МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра дискретной математики и алгоритмики

ЗЯЗЮЛЬКИН Сергей Павлович

ПОСТРОЕНИЕ БОЛЬШИХ НЕПЕРЕСЕКАЮЩИХСЯ АЦИКЛИЧЕСКИХ ПОДГРАФОВ В ГЕОМЕТРИЧЕСКИХ ГРАФАХ

Магистерская диссертация

специальность 1-31 81 09 «Алгоритмы и системы обработки больших объемов информации»

Научный руководитель
Сарванов Владимир Иванович
кандидат физико-математических наук

Допущена к защите

«___» _____ 2019 г.

Зав. кафедрой дискретной математики и алгоритмики
_____ В.М. Котов
доктор физико-математических наук, профессор

Минск, 2019

ОГЛАВЛЕНИЕ

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Магистерская диссертация, 49 с., 13 рис., 11 источников, 1 приложение.

Ключевые слова: ГЕОМЕТРИЧЕСКИЙ ГРАФ, НЕПЕРЕСЕКАЮЩЕЕСЯ ОСТОВНОЕ ДЕРЕВО, НАИБОЛЬШИЙ НЕПЕРЕСЕКАЮЩИЙСЯ АЦИКЛИЧЕСКИЙ ПОДГРАФ, ПЕРЕСЕЧЕНИЕ МАТРОИДОВ, АЛГОРИТМ ЧАСТИЧНОГО ПЕРЕБОРА С ОТСЕЧЕНИЯМИ.

Объект исследования – проблема построения больших непересекающихся подграфов в геометрических графах. В частности, исследуются задачи распознавания и построения непересекающегося остовного дерева и наибольшего непересекающегося ациклического подграфа.

Цель работы – изучить основные сведения о проблеме построения больших непересекающихся подграфов в геометрических графах; исследовать классы геометрических графов, допускающие построение непересекающегося остовного дерева за полиномиальное время; разработать точные экспоненциальные алгоритмы построения больших непересекающихся подграфов в геометрических графах; реализовать разработанные алгоритмы.

Методы проведения работы – методы теории графов, комбинаторной вычислительной геометрии, комбинаторной оптимизации и использование техники TDD разработки программного обеспечения.

Результаты – новый класс геометрических графов, допускающий решение задачи построения непересекающегося остовного дерева за полиномиальное время; разработанный и реализованный на языке программирования Java полиномиальный алгоритм построения непересекающегося остовного дерева в представленном классе геометрических графов; разработанные и реализованные на языке программирования Java алгоритмы частичного перебора с отсечениями для решения задачи построения больших непересекающихся подграфов в геометрических графах.

Область применения – проектирование интегральных схем, робототехника.

АГУЛЬНАЯ ХАРАКТАРЫСТЫКА ПРАЦЫ

Магістарская дысертацыя, 49 с., 13 мал., 11 крыніц, 1 дадатак.

Ключавыя словы: ГЕАМЕТРЫЧНЫ ГРАФ, НЕПЕРАСЯКАЛЬНАЕ ОСТАЎНАЕ ДРЭВА, НАЙБОЛЬШЫ НЕПЕРАСЯКАЛЬНЫ АЦЫКЛІЧНЫ ПАДГРАФ, ПЕРАСЯЧЭННЕ МАТРОІДАЎ, АЛГАРЫТМ ЧАСТКОВАГА ПЕРАБОРУ З АДСЯЧЭННЯМІ.

Аб’ект даследавання – праблема пабудовы вялікіх неперасякальных падграфаў у геаметрычных графах. У прыватнасці, даследуюцца задачы распазнавання і пабудовы неперасякальнага остаўнага дрэва і найбольшага неперасякальнага ацыклічнага падграфа.

Мэта працы – вывучыць асноўныя звесткі аб праблеме пабудовы вялікіх неперасякальных падграфаў у геаметрычных графах; даследаваць класы геаметрычных графаў, якія дапускаюць пабудову неперасякальнага остаўнага дрэва за паліномны час; распрацаваць дакладныя экспаненцыяльныя алгарытмы пабудовы вялікіх неперасякальных падграфаў у геаметрычных графах; рэалізаваць распрацаваныя алгарытмы.

Метады правядзення працы – метады тэорыі графаў, камбінаторнай вылічальнай геаметрыі, камбінаторнай аптымізацыі і выкарыстанне тэхнікі TDD распрацоўкі праграмнага забеспячэння.

Вынікі – новы клас геаметрычных графаў, які дапускае рашэнне задачы пабудовы неперасякальнага остаўнага дрэва за паліномны час; распрацаваны і рэалізаваны на мове праграмавання Java паліномны алгарытм пабудовы неперасякальнага остаўнага дрэва ў прадстаўленым класе геаметрычных графаў; распрацаваныя і рэалізаваныя на мове праграмавання Java алгарытмы частковага перабору з адсячэннямі для вырашэння задачы пабудовы вялікіх неперасякальных падграфаў у геаметрычных графах.

Галіна прымянення – праектаванне інтэгральных схем, робататэхніка.

ABSTRACT

Master thesis, 49 p., 13 fig., 11 sources, 1 appendix.

Keywords: GEOMETRIC GRAPH, NON-CROSSING SPANNING TREE, MAXIMUM NON-CROSSING ACYCLIC SUBGRAPH, MATROID INTERSECTION, PARTIAL SEARCH ALGORITHM WITH CLIPPING.

Object of research – the problem of constructing large non-crossing subgraphs in geometric graphs. In particular, the problem of constructing non-crossing spanning tree and the problem of constructing maximum non-crossing acyclic subgraph.

Objective – to study the basic information about the problem of constructing large non-crossing subgraphs in geometric graphs; to research classes of geometric graphs that allow construction of non-crossing spanning tree in polynomial time; to develop exact exponential algorithms for constructing large non-crossing subgraphs in geometric graphs; to implement the developed algorithms.

Methods – methods of graph theory, combinatorial computational geometry, combinatorial optimization and the usage of TDD technique for software development.

Results – a new class of geometric graphs that allows construction of non-crossing spanning tree in polynomial time; developed and implemented in Java programming language polynomial algorithm for constructing non-crossing spanning tree in the presented class of geometric graphs; developed and implemented in Java programming language algorithms for constructing large non-crossing subgraphs in geometric graphs.

Application area – development of integrated circuits, robotics.

ВВЕДЕНИЕ

Тематика диссертации относится к интенсивно развивающейся области исследований, находящейся «на стыке» теории графов и комбинаторной вычислительной геометрии. Задачи построения непересекающихся подграфов возникают, в частности, в автоматизации проектирования интегральных схем и в робототехнике. Кроме того, они имеют прямое отношение к задачам построения оптимальных по различным критериям плоских триангуляций, играющих ключевую роль в ряде прикладных областей.

Диссертационная работа направлена на исследование и разработку алгоритмов решения задач построения больших непересекающихся ациклических подграфов в геометрических графах. Основное внимание уделено непересекающемуся остовному дереву и наибольшему по числу ребер непересекающемуся ациклическому подграфу.

В рамках диссертации изучены основные аспекты проблемы построения непересекающегося остовного дерева: сложность задачи, необходимые и достаточные условия существования, полиномиально разрешимые случаи проблемы, параметризованные алгоритмы, оптимизационные версии задачи. Представлен новый класс геометрических графов, допускающий полиномиальные алгоритмы распознавания и построения непересекающегося остовного дерева. Для этого класса графов разработан алгоритм построения непересекающегося остовного дерева с временем работы $O(n^3)$. Выполнена разработка точных экспоненциальных алгоритмов частичного перебора с отсечениями для решения задач построения непересекающегося остовного дерева и наибольшего по числу ребер непересекающегося ациклического подграфа.

Программная реализация всех разработанных алгоритмов выполнена с использованием языка программирования Java, реализованные алгоритмы покрыты unit-тестами и размещены в публичном репозитории на github. Проведены вычислительные эксперименты, целью которых является демонстрация работы реализованных алгоритмов и их сравнение.

ГЛАВА 1
ОСНОВНЫЕ СВЕДЕНИЯ О ПРОБЛЕМЕ ПОСТРОЕНИЯ НЕПЕРЕСЕКАЮЩЕГОСЯ ОСТОВНОГО ДЕРЕВА В ГЕОМЕТРИЧЕСКОМ ГРАФЕ

1.1 Геометрические графы

Геометрическим графом (*рисунок 1.1*) называется граф, уложенный на плоскости так, что каждое его ребро является отрезком. Напомним, что под укладкой графа на плоскости понимается такое взаимно-однозначное отображение φ его вершин в точки плоскости, а ребер в дуги (кривые, являющиеся гомеоморфными образами отрезка [0,1]), соединяющие эти точки, при котором:

1. никакая дуга не содержит образов вершин, отличных от ее конечных точек;
2. две смежные дуги содержат только одну общую конечную точку;
3. для любых двух несмежных дуг существует не более одной точки, в которой они пересекаются (такое пересечение дуг вне образов вершин называется собственным).

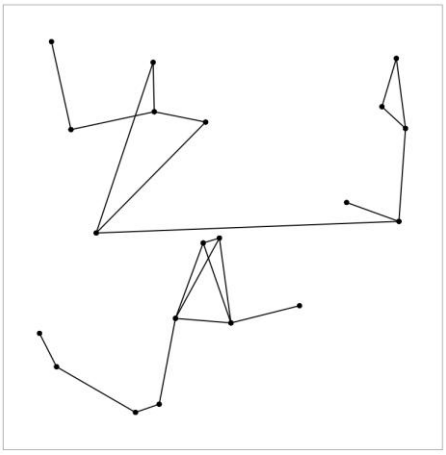

Рисунок 1.1 – Геометрический граф

Далее всюду будем называть образы вершин и ребер при отображении φ соответственно вершинами и ребрами соответствующего геометрического графа.

Индексом пересечения ребра геометрического графа назовем число собственных пересечений данного ребра. Наибольший индекс пересечения

ребра среди всех ребер геометрического графа назовем *индексом пересечения геометрического графа*. Геометрический граф, индекс пересечения которого равен нулю, называется *непересекающимся*.

В дальнейшем будем рассматривать только геометрические графы, вершины которых находятся в *общем положении*, т.е. никакие три вершины графа не лежат на одной прямой.

Выпуклую оболочку множества вершин геометрического графа будем называть *выпуклой оболочкой* этого *графа*. Говорят, что вершины геометрического графа G находятся в *выпуклом положении*, если они лежат на выпуклой оболочке графа G. Такой геометрический граф G называют *выпуклым*. Геометрический граф, не являющийся выпуклым, называют, соответственно, *невыпуклым*. Вершины невыпуклого геометрического графа, лежащие внутри его выпуклой оболочки, называются *внутренними*.

Для краткости изложения непересекающееся остовное дерево будем обозначать через NST (*Non-crossing Spanning Tree*), а наибольший непересекающийся ациклический подграф – через $MNAS$ (*Maximum Non-crossing Acyclic Subgraph*).

1.2 Задача построения непересекающегося остовного дерева и ее сложность

Рассмотрим *задачу распознавания NST в геометрическом графе*:

Вход: геометрический граф G.

Вопрос: существует ли NST в графе G?

В дальнейшем эту задачу будем называть *задачей распознавания NST*. Соответственно, задачу построения NST в геометрическом графе будем называть просто *задачей построения NST*. В работе [1] представлен следующий важный результат.

Теорема 1.1. *Задача распознавания NST в геометрическом графе G является NP-трудной, даже если степень любой вершины не более трех, а индекс пересечения графа G не превосходит двух.*

Из приведенной теоремы следует, что задача построения NST является NP-трудной. Тем не менее, эти факты не исключают возможность разработки эффективных приближений NST с константной относительной погрешностью. Имеется два естественных способа построения приближения NST, которые исследованы в работе [2].

Первый способ заключается в построении остовного дерева с минимальным числом пересечений. Обозначим через $\varphi(G)$ минимальное число

пересечений в NST геометрического графа G, увеличенное на единицу. О сложности задачи приближения параметра $\varphi(G)$ говорит следующая теорема.

Теорема 1.2. *Пусть G – геометрический граф, содержащий k пар пересекающихся ребер. Задача приближения параметра $\varphi(G)$ с относительной погрешностью $k^{1-\varepsilon}$ является NP-трудной для любого $\varepsilon > 0$.*

Второй способ приближения NST заключается в построении непересекающегося остовного леса с минимальным числом компонент. Обозначим через $\psi(G)$ минимальное число компонент в непересекающемся остовном лесе геометрического графа G. Как и в случае параметра $\varphi(G)$, эффективного приближения параметра $\psi(G)$ не существует при условии, что $P \neq NP$.

Теорема 1.3. *Пусть G – геометрический граф, содержащий k пар пересекающихся ребер. Задача приближения параметра $\psi(G)$ с относительной погрешностью $k^{1-\varepsilon}$ является NP-трудной для любого $\varepsilon > 0$.*

Из *теорем 1.2* и *1.3* следует, что:

1. не только задача построения NST, но и задача построения эффективного приближения NST, является NP-трудной;
2. при условии неравенства классов P и NP не существует полиномиального алгоритма, позволяющего приблизить NST с относительной погрешностью лучше, чем $O(k)$.

1.3 Параметризованные алгоритмы

Одним из направлений разработки эффективных алгоритмов для NP-трудных задач является построение параметризованных алгоритмов. Ключевым моментом при разработке параметризованного алгоритма является выбор подходящего параметра.

Самым простым способом решения задачи построения непересекающегося подграфа специального вида является полный перебор всех подграфов. Такой перебор может быть осуществлен за экспоненциальное от числа ребер в графе время. Однако, если число пересекающихся пар ребер значительно меньше общего числа ребер в графе и необходимое свойство (в данном случае, что подграф содержит остовное дерево) может быть проверено для непересекающегося подграфа за полиномиальное время, то существует более оптимальная, но все еще наивная стратегия: перебрать все непересекающиеся подграфы, получаемые из исходного графа удалением одного из ребер для каждой пары пересекающихся ребер. Число перебираемых при использовании данного подхода подграфов не превосходит 2^k, где k –

число пар пересекающихся ребер. Таким образом, используя наивный подход, задача построения NST для геометрического графа может быть решена за время $O^*(2^k) = O(p(k)2^k)$, где $p(k)$ – некоторый полином.

Первое усовершенствование наивного подхода было предложено в 2005-ом году. В работе [2] предложено решение задачи построения NST с трудоемкостью $O^*(1.9999996^k)$. Несмотря на то, что улучшение являлось незначительным, оно имело теоретический интерес и стимулировало дальнейшее развитие переборных алгоритмов для решения задачи построения NST.

В работе [3] была предложена идея сведения задачи построения NST в геометрическом графе G к задаче построения NST в геометрическом графе G' меньшего размера, при этом должно существовать биективное отображение между множествами NST графов G и G'. Такое сведение в дальнейшем будем называть *кернелизацией.* Такой граф G' может быть получен из графа G последовательным стягиванием ребер с индексом пересечения, равным 0. Отметим, что стягивание непересекающихся ребер не нарушает общую структуру пересечений в графе. Легко заметить, что кернелизация может быть выполнена за полиномиальное (на самом деле, линейное) время. Для ряда графов кернелизация может существенно уменьшить размер задачи. На основе идеи кернелизации и фиксирования порядка перебора ребер получен алгоритм решения задачи построения NST, имеющий время работы $O^*(1.9276^k)$ [3].

Теорема 1.4. *Пусть G − геометрический граф, содержащий k пар пересекающихся ребер. За время $O^*(1.9276^k)$ можно построить NST в графе G, если оно существует.*

Задача распознавания NST может быть решена за полиномиальное время для выпуклых геометрических графов. В работе [4] разработаны параметризованные алгоритмы, использующие в качестве параметра число внутренних вершин геометрического графа.

Теорема 1.5. *Пусть G − геометрический граф на n вершинах. Задача распознавания NST в графе G может быть решена за время $O\left(2^{33\sqrt{k}\log k}k^2n^3 + n^3\right)$. В качестве параметра k выступает число внутренних вершин графа G.*

Авторы теоремы также предлагают более «практичный» алгоритм, асимптотика которого, однако, несколько хуже.

Теорема 1.6. *Пусть G − геометрический граф на n вершинах. Задача распознавания NST в графе G может быть решена за время $O(2^{7.5k}k^2n^3 + n^3)$. В качестве параметра k выступает число внутренних вершин графа G.*

Для решения задачи распознавания NST также применялся вероятностный подход. Так, в работе [2] представлен следующий результат.

Теорема 1.7. *Пусть G − геометрический граф, содержащий k пар пересекающихся ребер. Существует вероятностный алгоритм типа Монте-Карло с односторонней ошибкой для решения задачи распознавания NST, имеющий вероятность успеха $p > 0$ и трудоемкость $O^*(1.968^k)$.*

1.4 Полиномиально разрешимые случаи проблемы

Случаев геометрических графов, допускающих проверку наличия и построение NST за полиномиальное время, очень мало.

Некоторые NP-трудные геометрические задачи на плоскости решаются за полиномиальное время для случая, когда точки находятся в выпуклом положении. Примером такой задачи является задача построения минимальной триангуляции множества точек на плоскости. Не исключением является и задача построения NST.

Теорема 1.8 [3]. *Пусть G – выпуклый геометрический граф на n вершинах. Задача построения NST в графе G может быть решена за время $O(n^3)$ с использованием метода динамического программирования.*

Еще один полиномиально разрешимый случай задачи построения NST рассмотрен в диссертации [5].

Теорема 1.9. *Пусть G такой геометрический граф, что для произвольной тройки e_i, e_j, e_k его ребер выполняется условие $e_i \cap e_j \neq \emptyset$ и $e_i \cap e_k \neq \emptyset \Rightarrow e_j \cap e_k \neq \emptyset$. Тогда задача построения NST в графе G является полиномиально разрешимой.*

Полиномиальная разрешимость данного случая обеспечивается сводимостью к известной полиномиально разрешимой задаче о пересечении двух матроидов. Из данной теоремы следует более простой полиномиально разрешимый случай задачи построения NST.

Следствие. *Проблема построения NST в геометрическом графе с индексом пересечения, равным единице, является полиномиально разрешимой.*

Данное следствие создает предпосылки к созданию параметризованного алгоритма построения NST, где в качестве параметра выступает число ребер, имеющих не менее двух собственных пересечений.

Достаточные и необходимые условия существования NST в геометрическом графе порождают случаи геометрических графов, допускающие решение задачи распознавания NST за полиномиальное время. Эти достаточные условия представлены в двух следующих разделах.

1.5 Достаточные условия существования непересекающегося остовного дерева

Рассмотрим три вершины u, v, w геометрического графа G и образуемый ими треугольник Δuvw. Отметим, что стороны треугольника могут не являться ребрами графа G. Треугольник Δuvw называют *пустым*, если внутри него не содержится ни одна из вершин графа G. Если подграф геометрического графа G, порожденный тремя вершинами u, v, w, является несвязным, то треугольник Δuvw также называют *несвязным*. Число несвязных пустых треугольников в геометрическом графе G обозначим через $s(G)$. В работе [6] представлены следующие результаты.

Теорема 1.10. *Если геометрический граф G на n вершинах, $n \geq 3$ удовлетворяет условию $s(G) \leq n - 3$, то в графе G существует NST.*

Напомним, что *дополнением* геометрического графа G называется геометрический граф G' на том же множестве вершин, такой, что две вершины в графе G' смежны тогда и только тогда, когда они не смежны в графе G.

Теорема 1.11. *Дополнение остовного дерева, индекс пересечения которого больше нуля, содержит NST.*

Случай, когда вместо дополнения остовного дерева рассматривается дополнение NST, оказывается значительно сложнее. Поэтому он вводится дополнительное ограничение на выпуклость NST.

Теорема 1.12. *Дополнение NST T, вершины которого находятся в выпуклом положении, содержит NST тогда и только тогда, когда T обладает по крайней мере двумя различными компонентами на границе своей выпуклой оболочки.*

Ряд работ посвящен исследованию локальных достаточных условий существования NST. В статье [7] сформулирована следующая гипотеза.

Гипотеза. *Пусть $m \geq 2$ – произвольное целое число и G – геометрический граф с по крайней мере m вершинами. Тогда, если для любого множества U из m вершин графа G подграф, индуцированный множеством U, содержит NST, то и граф G содержит NST.*

Доказательство или опровержение этой гипотезы пока не получено. В работе [8] приведено подтверждение гипотезы в частном случае, когда $m = 6$.

Теорема 1.13. *Если для любого подмножества $U \subset V$ из 6 вершин геометрического графа $G = (V, E)$ порядка $n \geq 6$ подграф, индуцированный множеством U, содержит NST, то и сам граф G содержит NST.*

1.6 Необходимые условия существования непересекающегося остовного дерева

Пусть $G = (V, E)$ – полный геометрический граф. Рассмотрим такой минимальный подграф $B = (V', E')$ графа G, что любое NST графа G содержит хотя бы одно ребро подграфа B. Очевидно, что удаление всех ребер подграфа B приводит к тому, что в получаемом графе $G' = (V, E \setminus E')$ не существует NST. Такой подграф B будем называть *блокатором* NST или, в дальнейшем, просто *блокатором*. Из определения блокатора следует, что NST существует в геометрическом графе только тогда, когда его дополнение не содержит блокатор в качестве подграфа.

Подробное исследование различных характеристик блокаторов проведено в работе [9]. Основными являются следующие результаты.

Теорема 1.14. *Пусть $G = (V, E)$ – полный геометрический граф, а $B = (V', E')$ – подграф графа G такой, что в графе $G' = (V, E \setminus E')$ не существует NST, диаметр которого не превосходит 4. Тогда подграф B является блокатором.*

В случае выпуклого геометрического графа получен более сильный результат.

Теорема 1.15. *Пусть $G = (V, E)$ – выпуклый полный геометрический граф, а $B = (V', E')$ – подграф графа G такой, что в графе $G' = (V, E \setminus E')$ не существует NST, диаметр которого не превосходит 3. Тогда подграф B является блокатором.*

Рассматривалась еще одна проблема для полного геометрического графа: как много произвольных ребер может быть удалено из полного геометрического графа на n вершинах, чтобы оставшийся граф по-прежнему содержал NST. Основные результаты по этой проблеме представлены в работе [10].

Теорема 1.16. *Для любого k, $2 \leq k \leq n - 1$ и всех полных геометрических графов G на n вершинах, а также для всех подграфов H графа G, содержащих не более $\lceil kn/2 \rceil - 1$ вершин, геометрический граф $G - H$ содержит непересекающееся дерево на $n - k + 1$ вершинах. Данная оценка является строгой относительно числа ребер в подграфе H: каждый полный геометрический граф G содержит подграф H с $\lceil kn/2 \rceil$ ребрами такой, что любое непересекающееся дерево графа $G - H$ содержит не более $n - k$ вершин.*

1.7 Оптимизационные версии задачи

В большинстве приложений геометрических графов наличие пересекающихся ребер является нежелательной характеристикой. Некоторые геометрические структуры, например триангуляция, не содержат пересекающихся ребер по определению. Остовные деревья этим свойством не обладают. Рассмотрим две оптимизационные проблемы, а именно построение NST минимальной и максимальной длины в полном геометрическом графе. Под длиной NST подразумевается сумма длин всех ребер, входящих в дерево.

Отсутствие пересекающихся ребер в остовном дереве минимальной длины следует из неравенства треугольника. Таким образом, любое остовное дерево минимальной длины автоматически будет являться непересекающимся. Построить такое дерево, очевидно, можно любым алгоритмом для построения минимального остовного дерева.

В случае проблемы построения NST максимальной длины ситуация значительно сложнее. Максимизация длины входящих в остовное дерево ребер вступает в противоречие с необходимостью обеспечить отсутствие пересекающихся ребер. Есть предположение, что проблема построения NST максимальной длины в полном геометрическом графе является NP-трудной, однако доказательство или опровержение этого предположения пока не получено. В статье [11] произведен анализ данной проблемы и предложен приближенный алгоритм ее решения.

Теорема 1.17. *Пусть G − полный геометрический граф на n вершинах. Существует приближенный алгоритм решения задачи построения NST в геометрическом графе с относительной погрешностью* 0.502 *и трудоемкостью* $O(n \log n)$.

На текущий момент данный алгоритм является лучшим из известных приближений NST максимальной длины в полном геометрическом графе.

ГЛАВА 2
ЗАДАЧА ПЕРЕСЕЧЕНИЯ ДВУХ МАТРОИДОВ

2.1 Матроиды

Матроидом называют упорядоченную пару (E, X), где E – непустое конечное множество, а X – непустое множество его подмножеств, удовлетворяющее следующим условиям:

1. $\emptyset \in X$;
2. Если $I \in X$ и $I' \subseteq I$, то $I' \in X$;
3. Если $I_1 \in X$, $I_2 \in X$ и $|I_1| < |I_2|$, то существует элемент $e \in I_2 \setminus I_1$ такой, что $I_1 \cup \{e\} \in X$.

Если M – матроид (E, X), то M называют *матроидом на множестве* E. Элементы множества X называются *независимыми множествами* матроида M, а множество E – *носителем* матроида M. Множества E и X матроида M обозначаются через $E(M)$ и $X(M)$ соответственно. Подмножество множества E, которое не содержится в множестве X, называется *зависимым*. Говорят, что множество C является *циклом* матроида M, если C – минимальное по включению зависимое множество матроида M. Максимальные по включению независимые множества матроида M называются *базисами* или *базами*.

Утверждение 2.1 [12]. Если B_1 и B_2 – базисы матроида M, то $|B_1| = |B_2|$.

Утверждение 2.2 [12]. Пусть I – независимое множество матроида $M = (E, X)$, $e \in E$ и $I \cup \{e\}$ – зависимое множество. Тогда в множестве $I \cup \{e\}$ содержится единственный цикл матроида M, причем этот цикл содержит элемент e.

Пусть $G = (V, E)$ – геометрический граф. Рассмотрим два примера матроидов $M = (E, X)$ на множестве ребер графа G.

1. *Матроид циклов* или *графический матроид*. Независимое множество матроида представляет собой ациклическое подмножество ребер графа G, цикл – простой цикл графа G, а базис – остовный лес графа G.
2. *Матроид разбиения*. Пусть задано некоторое разбиение P множества E ребер графа, т.е. задано семейство непересекающихся подмножеств множества E, покрывающих E. Подмножество I ребер графа называется независимым в том и только в том случае, если никакие два ребра из I не лежат в одном и том же множестве разбиения P. Циклом матроида разбиения является любое множество, состоящее из двух ребер

одного и того же множества разбиения P. Множество ребер, содержащее ровно по одному ребру из каждого множества разбиения P, является базисом матроида разбиения. Пусть множество ребер графа G можно разбить на непересекающиеся подмножества так, что любые два ребра одно и того же подмножества пересекаются, а любые два ребра из разных подмножеств – нет. Будем называть такой матроид разбиения *матроидом пересечений*.

2.2 Постановка задачи

Пусть заданы два матроида $M_1 = (E, X_1)$ и $M_2 = (E, X_2)$, базирующихся на одном и том же множестве элементов E. *Задача пересечения двух матроидов* состоит в отыскании наибольшего по мощности множества X', которое является независимым в обоих матроидах. Т.е. для такого множества X' выполняются следующие условия:

1. $X' \subseteq E$;
2. $X' \in X_1$;
3. $X' \in X_2$;
4. X' является наибольшим по мощности множеством среди всех множеств, удовлетворяющих условиям 1−3.

Для краткости изложения задачу пересечения двух матроидов будем называть *задачей TMI (Two Matroids Intersection)*.

Задача TMI является полиномиально разрешимой. Алгоритм решения этой задачи будет рассмотрен в следующем разделе. Отметим, что задача пересечения трех матроидов уже является NP-трудной.

2.3 Алгоритм решения задачи

Рассмотрим два матроида $M_1 = (E, X_1)$, $M_2 = (E, X_2)$ и последовательность $S = [e_1, e_2, \dots, e_{2k+1}]$ элементов множества E. Говорят, что последовательность S является *увеличивающим путем* для некоторого множества X, независимого в матроидах M_1 и M_2, если применение этой последовательности к множеству X переводит это множество в множество X', также являющееся независимым в матроидах M_1 и M_2, такое, что $|X'| = |X| + 1$. Под *применением последовательности* к множеству X понимается добавление в него всех элементов, находящихся на нечетных позициях, и удаление из него всех элементов, находящихся на четных позициях. При этом на добавляемые и

удаляемые элементы накладываются следующие ограничения: добавляемый элемент должен отсутствовать в множестве, а удаляемый – присутствовать в нем. Отметим, что для того, чтобы выполнялось условие $|X'| = |X| + 1$, увеличивающий путь всегда содержит нечетное число элементов.

Идея алгоритма решения задачи TMI заключается в следующем. Пусть известно некоторое множество X, являющееся независимым в каждом из пересекаемых матроидов. Для начала можно взять пустое множество, т.к. оно является независимым по определению. Затем, пока это возможно, строятся увеличивающие пути. Полученное в результате увеличений множество и будет искомым.

Открытым остается вопрос построения увеличивающего пути для имеющегося множества X. Эта проблема решается построением двудольного орграфа G специального вида, вершины которого соответствуют элементам множества E, и двух множеств Q и T. Множество Q содержит элементы (вершины), с которых может начинаться увеличивающий путь, множество T – элементы (вершины), которыми может заканчиваться увеличивающий путь. Построение увеличивающего пути сводится к нахождению кратчайшего пути, ведущего из вершины из множества Q в вершину из множества T. Нахождение кратчайшего пути выполняется простой модификацией алгоритма поиска в ширину.

Алгоритм решения задачи TMI [12].

Вход: два матроида $M_1 = (E, X_1)$ и $M_2 = (E, X_2)$, базирующихся на одном и том же множестве элементов E.

Выход: максимальное по мощности множество I, являющееся независимым для матроидов M_1 и M_2.

1. Множество I инициализируется пустым множеством.
2. Выполняется поиск увеличивающего пути для множества I.
   - 2.1. Выполняется инициализация структур данных.
     - 2.1.1. Очередь Q инициализируется пустой очередью. Эта очередь будет использоваться для хранения элементов, с которых может начинаться увеличивающий путь.
     - 2.1.2. Множество T инициализируется пустым множеством. В этом множестве будут храниться элементы, которыми может заканчиваться увеличивающий путь.
     - 2.1.3. Список смежности A инициализируется пустым списком. Этот список смежности будет использоваться для хранения дуг вспомогательного двудольного орграфа, используемого для поиска увеличивающего пути.
     - 2.1.4. Словарь P инициализируется пустым словарем. В этом словаре для элементов будет храниться родительский

элемент, т.е. элемент, из которой мы пришли в данный во время поиска увеличивающего пути. Этот словарь нужен для восстановления увеличивающего пути.

2.2. Выполняется построение вспомогательного двудольного орграфа.

2.2.1. Итерируемся по всем элементам множества $E \setminus I$.

2.2.2. Пусть на текущей итерации просматривается элемент e_i. Обозначим множество $I \cup \{e_i\}$ через I'.

2.2.3. Если множество I' является независимым в матроидах M_1 и M_2, то увеличивающий путь найден – он состоит из единственного элемента e_i. Переходим к *шагу 3*.

2.2.4. Если множество I' является независимым только в матроиде M_1, то добавляем элемент e_i в хвост очереди Q. В противном случае, согласно *утверждению 2.2*, множество I' содержит единственный цикл C в матроиде M_1. Добавляем в список смежности A дуги из элементов цикла C в элемент e_i, исключая петлю (e_i, e_i).

2.2.5. Если множество I' является независимым только в матроиде M_2, то добавляем элемент e_i в множество T. В противном случае, согласно *утверждению 2.2*, множество I' содержит единственный цикл D в матроиде M_2. Добавляем в список смежности A дуги из элемента e_i в элементы цикла D, исключая петлю (e_i, e_i).

2.3. Выполняем поиск увеличивающего пути.

2.3.1. Если очередь Q пуста, то увеличивающий путь для множества I не существует. Переходим к *шагу 4*.

2.3.2. Извлекаем элемент e из головы очереди Q.

2.3.3. Итерируемся по всем элементам e', в которые ведут дуги из элемента e (используем список смежности A).

2.3.4. Если элемент e' еще не просматривался, т.е. словарь P не хранит предка данного элемента, то добавляем в словарь P e в качестве предка вершины e'.

2.3.5. Если элемент e' принадлежит множеству T, то увеличивающий путь существует, переходим к восстановлению увеличивающего пути (*шаг 2.4*). В противном случае увеличивающий путь пока не найден, возвращаемся к *шагу 2.3.1*.

2.4. Восстанавливаем увеличивающий путь, итерируясь по предкам элементов, используя словарь P. Увеличивающий путь найден, переходим к *шагу 3*.

3. Если увеличивающий путь найден, то применяем его к множеству I и возвращаемся к *шагу 2*.

4. Если увеличивающий путь не найден, то алгоритм завершает свою работу, возвращая в качестве ответа множество I.

Трудоемкость данного алгоритма зависит от вида пересекаемых матроидов. В частности, ключевую роль играет трудоемкость алгоритма поиска цикла в зависимом множестве матроида.

Теорема 2.1 [12]. *Пусть на вход алгоритма решения задачи TMI подаются матроиды* $M_1 = (E, X_1)$ *и* $M_2 = (E, X_2)$*, а задача поиска цикла в зависимом множестве может быть решена за время* $O\big(C(|E|)\big)$ *для каждого из матроидов* M_1*,* M_2*. Тогда представленный алгоритм корректно решает задачу TMI за время* $O\big(|E|^3 C(|E|)\big)$.

2.4 Особенности программной реализации

Для реализации алгоритма решения задачи *TMI* был выбран язык программирования *Java*. Реализация алгоритма велась с использованием интегрированной среды разработки *IntelliJ IDEA*, системы автоматической сборки *Gradle*, библиотеки для модульного тестирования *JUnit*.

Алгоритм решения задачи TMI был реализован в общем виде без привязки к каким-либо конкретным видам пересекаемых матроидов. Класс, отвечающий за решение задачи TMI, представлен интерфейсом всего из одного метода, принимающего на вход два матроида и возвращающего наибольшее по мощности независимое множество в заданных матроидах. Сами матроиды также представлены интерфейсом, содержащим два метода. Первый метод позволяет получить носитель матроида, второй – находит в заданном подмножестве носителя матроида цикл, если таковой имеется.

Для представления графа, а также его ребер и вершин, было принято решение вместо использования сторонних библиотек разработать собственную реализацию с целью упрощения работы с графом и оптимизации реализации под нужды разрабатываемого алгоритма.

Интерфейс вершин графа крайне прост, он содержит всего два метода: получение идентификатора вершины (элемента, которому соответствует вершина; как правило, в качества идентификатора вершины используется число) и получение координат вершины на плоскости. На реализацию интерфейса вершины накладывается ограничение: вершины с одинаковым идентификатором должны быть равны между собой.

Интерфейс ребра позволяет получить его альтернативные представления: в виде пары вершин, в виде отрезка на плоскости, в виде потока вершин. Как и в случае вершины, на реализацию интерфейса ребра накладывается дополнительное ограничение: ребра между одними и теми же вершинами с сохранением направленности должны быть равны между собой.

Важно отметить, что реализации интерфейса вершин и ребер должны быть immutable (неизменяемыми), что позволит безопасно передавать их без клонирования за пределы классов, которым они принадлежат, а также использовать их в многопоточной среде без дополнительной синхронизации.

Интерфейс графа оставляет за конкретной реализацией ответы на следующие вопросы: являются ли ребра направленными, допускаются ли в графе петли, допускаются ли в графе кратные ребра. Помимо стандартных методов получения, добавления и удаления вершин и ребер графа, интерфейс содержит методы, позволяющие получить индекс пересечения графа, ребро с наибольшим индексом пересечения, связные компоненты и мосты.

Ввиду того, что конкретная реализация интерфейсов вершин и ребер для графа скрыта от пользователя, интерфейс графа предоставляет factory-методы для создания вершин и ребер.

Для реализации алгоритма были разработаны следующие имплементации описанных интерфейсов: *SimpleVertex* (вершина, соответствующая точке на плоскости), *SimpleUndirectedEdge* (неориентированное ребро), *UndirectedGraphWithIntersections* (неориентированный граф, не допускающий петли и кратные ребра), *BaseMatroidIntersectionAlgorithm* (реализация алгоритма пересечения двух матроидов, представленного в предыдущем разделе).

Реализация алгоритма является однопоточной, т.к. в дальнейшем предполагается использовать алгоритм пересечения двух матроидов при разработке алгоритмов решения задачи построения непересекающихся подграфов в геометрических графах и выполнять распараллеливание на более высоком уровне.

Ссылка на реализацию разработанного алгоритма находится в *приложении А*.

2.5 Применение для решения задачи построения наибольшего непересекающегося ациклического подграфа

Пусть задан некоторый геометрический граф. Рассмотрим два матроида на множестве ребер этого графа: графический матроид и матроид пересечений.

Напомним, что графический матроид на множестве ребер геометрического графа может быть построен всегда, а матроид пересечений – лишь в случае, когда множество ребер графа может быть разбито на непересекающиеся подмножества так, что никакие два ребра из разных подмножеств не пересекаются, любые два ребра из одного и того же подмножества пересекаются. Пример геометрического графа, допускающего построение матроида пересечений, представлен на *рисунке 2.1*.

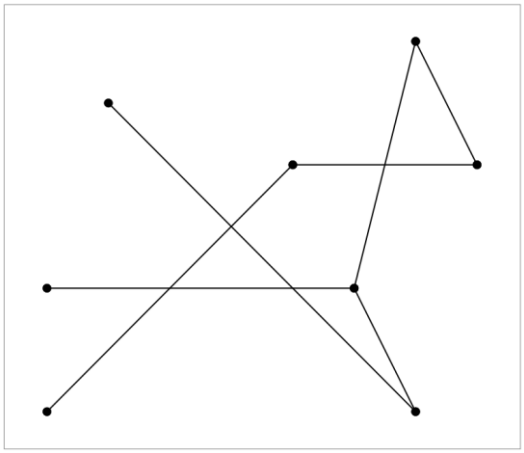

Рисунок 2.1 – Геометрический граф, допускающий построение матроида пересечений

Независимость множества в графическом матроиде означает ацикличность множества ребер, независимость в матроиде пересечений – отсутствие пересекающихся ребер. Пересечение графического матроида и матроида пересечений представляет собой $MNAS$. Если число ребер в этом подграфе ровно на единицу меньше числа вершин в исходном графе, то такой подграф является NST. Таким образом, если можно построить матроид пересечений на множестве ребер геометрического графа, то можно проверить наличие и построить NST в случае его существования за полиномиальное время, используя алгоритм решения задачи TMI.

Теорема 2.2. *Пусть $G = (V, E)$ – геометрический граф, на множестве ребер E которого может быть построен матроид пересечений. Тогда задача построения NST и задача построения $MNAS$ могут быть решены для графа G за полиномиальное время.*

Примером геометрического графа, допускающего построение матроида пересечений на множестве ребер, является геометрический граф, имеющий индекс пересечения, равный единице. Пример такого графа изображен на *рисунке 2.2*.

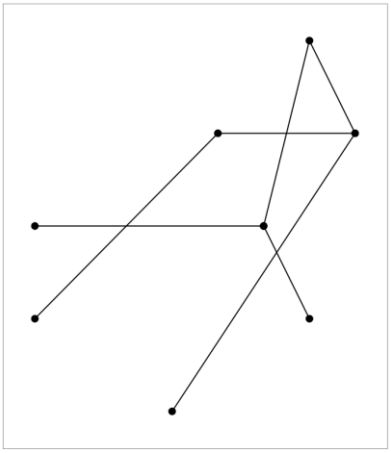

Рисунок 2.2 – Геометрический граф с индексом пересечения, равным 1

2.6 Применение для решения задачи построения наибольшего непересекающегося ациклического подграфа с множеством зафиксированных ребер

Рассмотрим более общий случай задачи построения NST и $MNAS$. Пусть вместе с геометрическим графом $G = (V, E)$ задано некоторое фиксированное подмножество $E' \subseteq E$ его ребер. Необходимо построить в графе G NST ($MNAS$), которое(-ый) содержит в себе E'.

В данном разделе будет показано, что эта задача также может быть решена за полиномиальное время для геометрических графов, допускающих построение матроида пересечений на множестве ребер.

Пусть задан матроид $M = (E, X)$ и независимое множество N этого матроида. Рассмотрим упорядоченную пару $M' = (E', X')$, где $E' = E \backslash N$, $X' = \{N' : N' \subseteq E',\ N' \cup N \in X\}$. Докажем, что упорядоченная пара M' также является матроидом. Для этого покажем, что матроид M удовлетворяет каждому из трех условий определения матроида. Доказательство будем производить методом «от противного».

1. Предположим, что $\emptyset \notin X'$. Тогда $N \notin X$. Получаем противоречие с тем, что N по определению является независимым множеством матроида M.
2. Предположим, что $I \in X'$, $I' \subseteq I$ и $I' \notin X'$. Обозначим $J = I \cup N$, $J' = I' \cup N$. Из определения упорядоченной пары M' следует, что $J \in X$, $J' \subseteq J$ и $J' \notin X$. Получаем, что матроид M не удовлетворяет второму условия из определения матроида – противоречие.
3. Предположим, что $I_1 \in X'$, $I_2 \in X'$, $|I_1| < |I_2|$ и не существует элемент $e \in I_2 \backslash I_1$ такой, что $I_1 \cup \{e\} \in X'$. Обозначим $J_1 = I_1 \cup N$, $J_2 = I_2 \cup N$. Из определения упорядоченной пары M' следует, что $J_1 \in X$,

$J_2 \in X$, $|J_1| < |J_2|$ и не существует элемент $e \in J_2 \setminus J_1 = I_2 \setminus I_1$ такой, что $J_1 \cup \{e\} \in X$. Получаем, что матроид M не удовлетворяет третьему условию из определения матроида – противоречие.

Таким образом, упорядоченная пара M', полученная из матроида M путем фиксирования независимого множества N, также является матроидом. Такой матроид M' будем называть *матроидом с зафиксированным множеством* N. Заметим, что каждому независимому множеству I' матроида M' соответствует независимое множество $I = I' \cup N$ матроида M.

Пусть задан графический матроид $M_g = (E, X_g)$, матроид пересечений $M_i = (E, X_i)$ и множество зафиксированное ребер N такое, что $N \in X_g$ и $N \in X_i$. Используя алгоритм решения задачи TMI, построим пересечение матроидов M'_g и M'_i, полученных из матроидов M_g и M_i соответственно путем фиксирования независимого множества N. Добавив к полученному пересечению множество N, получим пересечение исходных матроидов M_g и M_i, причем это пересечение содержит фиксированное множество ребер N.

Теорема. 2.3. *Пусть $G = (V, E)$ – геометрический граф, на множестве ребер E которого может быть построен матроид пересечений. Пусть $E' \subseteq E$ – зафиксированное подмножество ребер графа G. Тогда задача построения NST и $MNAS$ с множеством зафиксированным ребер E' могут быть решены для графа G за полиномиальное время путем сведения этих задач к задаче пересечения двух матроидов.*

Эта теорема будет использована в дальнейшем в работе при разработке алгоритмов частичного перебора с отсечениями для решения задачи построения NST и задачи построения $MNAS$.

ГЛАВА 3
ТОЧНЫЕ АЛГОРИТМЫ ПОСТРОЕНИЯ НАИБОЛЬШИХ НЕПЕРЕСЕКАЮЩИХСЯ АЦИКЛИЧЕСКИХ ПОДГРАФОВ

В данной главе будут представлены точные экспоненциальные алгоритмы решения задачи построения NST и $MNAS$, базирующиеся на методе частичного перебора с отсечениями.

При переборе для ребра будем рассматривать две возможные ситуации: ребро не входит в искомый подграф, ребро входит в искомый подграф. В первом случае происходит удаление рассматриваемого ребра. Во втором – ребро фиксируется и становится обязательным для включения. Во время фиксирования ребра также происходит удаление ребер, которые не могут входить в искомый подграф вместе с зафиксированным. Такими, например, являются ребра, которые пересекают зафиксированное ребро. Во время перебора для каждого варианта возможны два случая:

1. вариант имеет специальный вид, допускающий проверку наличия и построение искомого подграфа за полиномиальное время;
2. вариант порождает некоторое множество подвариантов, которые необходимо проверить.

3.1 Свойства геометрического графа, используемые в алгоритмах частичного перебора с отсечениями

Очевидно, что несвязный геометрический граф не содержит NST. При этом несвязный геометрический граф не допускает построение $MNAS$ за полиномиальное время.

Непересекающиеся геометрические графы допускают проверку наличия и построение NST и $MNAS$ за полиномиальное время. Любой подграф непересекающегося геометрического графа будет непересекающимся по определению. Поэтому для построения NST и $MNAS$ можно применить любой жадный алгоритм. Большинство жадных алгоритмов могут быть легко модифицированы таким образом, чтобы строить NST и $MNAS$ с заранее заданным множеством зафиксированных ребер, которое должно входить в искомых подграф.

Воспользуемся тем свойством, что любой мост графа обязательно входит в остовное дерево этого графа при условии, что оно существует. Таким

образом, если геометрический граф содержит один или более мостов, то задача построения NST может быть разбита на подзадачи меньшего размера. Удалим из геометрического графа все мосты и ребра, пересекающие эти мосты. Если число полученных компонент связности превосходит число мостов, увеличенное на единицу, то исходный граф не содержит NST. Если число компонент связности в точности равно числу мостов, увеличенному на единицу, то решим задачу построения NST для каждой из компонент связности. Если хотя бы одна из компонент связности не содержит NST, то и исходный геометрический граф не содержит NST. Если для каждой компоненты связности удалось построить NST, то NST исходного геометрического графа может быть получено добавлением удаленных мостов к построенным NST. Отметим, что такое разбиение на подзадачи применимо для задачи построения NST, но не для задачи построения $MNAS$, т.к. мост геометрического графа может не входить в его $MNAS$.

Задачи построения NST и $MNAS$ могут быть решены за полиномиальное время для геометрического графа, допускающего построение матроида пересечений на множестве ребер. Проверка, можно ли построить матроид пересечений на множестве ребер геометрического графа, является довольно трудоемкой. Напомним, что геометрический граф с индексом пересечения, равным единице, допускает построение матроида пересечений. Проверка величины индекса пересечений геометрического графа значительно проще проверки возможности построения матроида пересечений.

3.2 Алгоритм частичного перебора с отсечениями для решения задачи построения непересекающегося остовного дерева

В данном разделе представлен точный экспоненциальный алгоритм частичного перебора с отсечениями для решения задачи построения NST с опциональным множеством зафиксированных ребер.

Алгоритм частичного перебора с отсечениями для решения задачи построения NST с опциональным множеством зафиксированных ребер.

Вход: геометрический граф G, опциональное множество N зафиксированных ребер.

Выход: NST, содержащее множество N зафиксированных ребер, если такое NST существует в графе G.

1. Множество зафиксированных ребер N инициализируется пустым множеством, если оно не было подано на вход.

2. Очередь задач Q инициализируется очередью из одного элемента – пары (G, N). Очередь Q будет использоваться для хранения подзадач, которые необходимо решить.
3. Если очередь Q пуста, то в геометрическом графе G отсутствует NST. Алгоритм завершает свою работу.
4. Извлекаем из очереди Q пару (G_i, N_i).
5. Если геометрический граф G_i является несвязным, то он не содержит NST, возвращаемся на *шаг 3*.
6. Если геометрический граф G_i является непересекающимся, то выполняем построение NST с фиксированным множеством ребер N_i, используя любую модификацию алгоритма поиска обычного остовного дерева, поддерживающую задание ребер, которые обязаны входить в остовное дерево. Если удалось построить NST, то алгоритм завершает свою работу, возвращая построенное NST в качестве ответа. Если построить NST не удалось, то возвращаемся на *шаг 3*.
7. Если возможно построить матроид пересечений на множестве ребер геометрического графа G_i (например, индекс пересечения графа G_i равен 1), то выполняем построение NST с фиксированным множеством ребер N_i, используя алгоритм решения задачи TMI. Если удалось построить NST, то алгоритм завершает свою работу, возвращая построенное NST в качестве ответа. Если построить NST не удалось, то возвращаемся на *шаг 3*.
8. *Опциональный шаг.* Если геометрический граф G_i содержит хотя бы один мост, то решаем задачу построения NST с указанием соответствующих фиксированных ребер для каждой из компонент связности, которые получаются из графа G_i путем удаления мостов и всех ребер, пересекающих мосты. Если после удаления мостов и пересекающих их ребер число образованных компонент связности превосходит число мостов, увеличенное на единицу, то NST в графе G_i отсутствует, возвращаемся на *шаг 3*. Если для каждой из компонент связности удалось построить NST, то строим NST графа G_i, объединяя NST компонент связности с мостами графа G_i. Алгоритм завершает свою работу, возвращая построенное NST в качестве ответа. Если хотя бы для одной компоненты связности NST построить не удалось, то возвращаемся на *шаг 3*. Отметим, что решать задачу построения NST для компонент связности можно двумя способами. Первый (самый простой) способ заключается в рекурсивном вызове данного алгоритма для каждой из компонент связности. Второй способ, который может быть полезен для параллельной реализации алгоритма, предполагает добавление в очередь Q подзадач (G_{ij}, N_{ij}), где G_{ij} – j-ая компонента связности графа G_i,

$N_{ij} = N_i \cap E(G_{ij})$. При этом необходимо модифицировать соответствующим образом данный алгоритм для обработки таких подзадач и выполнения агрегирования результатов после их решения.

9. Выбираем некоторое ребро e_i графа G_i, имеющее индекс пересечения, больший нуля. Такое ребро всегда можно выбрать, т.к. в противном случае алгоритм остановился бы на *шаге 6*. Алгоритм выбора ребра e_i допускает вариации и зависит от конкретной реализации алгоритма. Например, можно брать ребро с наибольшим индексом пересечения или случайное ребро.

10. Если ребро e_i не образует цикл с зафиксированными ребрами N_i, то строим граф G_{i1}, удаляя из графа G_i все ребра, что пересекают ребро e_i, и добавляем в очередь Q подзадачу $(G_{i1}, N_i \cup \{e_i\})$.

11. Строим граф G_{i2} удалением ребра e_i и добавляем в очередь Q подзадачу (G_{i2}, N_i).

12. Возвращаемся на *шаг 3*.

Представленный алгоритм оставляет за реализацией решение следующих вопросов:

1. Реализация очереди Q. Задачи могут извлекаться в порядке поступления, случайным образом, в соответствии с некоторым приоритетом или иным образом.
2. Конкретный алгоритм поиска NST с множеством зафиксированных ребер, используемый на *шаге 6*.
3. Включение в алгоритм *шага 8*.
4. Алгоритм выбора ребра на *шаге 9*.
5. Распараллеливание алгоритма: пары (G_i, N_i) могут обрабатываться независимо друг от друга.
6. Мемоизация и кэширование с целью исключения повторных вычислений.

Оценим трудоемкость представленного алгоритма при следующих условиях: произвольная реализация очереди Q, любой полиномиальный алгоритм поиска NST для *шага 6*, отсутствие *шага 8*, выбор ребра с наибольшим индексом пересечения на *шаге 9*.

Пусть на вход алгоритму подан граф G, содержащий k пар пересекающихся ребер, время работы *шагов 5-7* алгоритма составляет $O(P_1(n,m))$, время работы *шагов 9-11* – $O(P_2(n,m))$, где $n = |V(G_i)|$, $m = |E(G_i)|$, P_1 и P_2 – некоторые полиномы.

Подзадачи (G_i, N_i) будем называть *простыми*, если они могут быть решены без дальнейшего разбиения на подзадачи, т.е. будут обработаны на *шагах 5-7*. Подзадачи, не являющиеся простыми, будем называть *составными*. Отметим, что обработка составных задач выполняется на *шагах 5-7* и *шагах 9-*

11. Под *размером подзадачи* (G_i, N_i) будем понимать число пар пересекающихся ребер в графе G_i. Граф G_i будем называть *графом подзадачи*. Обработка простых подзадач может быть произведена за время $O\big(P_1(n_i, m_i)\big)$, составных – за время $O\big(P_1(n_i, m_i) + P_2(n_i, m_i)\big)$.

Очевидно, что подзадачи размера 0 являются простыми, т.к. графы подзадач являются непересекающимися. Простыми также являются подзадачи (G_i, N_i), где граф G_i имеет индекс пересечения, равный 1, т.к. такие подзадачи будут обработаны на *шаге 6*. Следовательно, простыми также являются подзадачи размера 1, потому что индекс пересечения графов таких подзадач будет также равен 1.

Подзадача либо является простой, либо разбивается на не более чем две подзадачи, каждая из которых, в свою очередь, может быть простой или составной. Заметим, что размер подзадач, образуемых при разбиении составной задачи, не менее чем на 2 меньше размера исходной составной подзадачи. Действительно, если подзадача является составной, то индекс пересечения ее графа равен минимум 2. Отсюда следует, что и индекс пересечения ребра e_i, выбираемого на *шаге 9*, также будет не меньше 2. При удалении или фиксировании ребра e_i размер получаемой подзадачи уменьшается на число пересекаемых ребром e_i ребер, т.е. не менее чем на 2.

Исходная задача (G, N) имеет размер k. Исходя из того, что при разбиении размер образуемых подзадач не менее чем на 2 меньше размера исходной составной подзадачи, а также учитывая тот факт, что число образуемых подзадач не превосходит 2, получаем, что задача размера k может породить не более чем $2^{k/2}$ простых подзадач и не более чем $2^{k/2}$ составных подзадач. Следовательно, общее время работы алгоритма составляет $O\left(2^{k/2}P_1(n,m) + 2^{k/2}\big(P_1(n,m) + P_2(n,m)\big)\right) = O\left(2^{k/2}(2P_1(n,m) + P_2(n,m)\right) = O^*\left(2^{k/2}\right) = O^*\left(\sqrt{2}^k\right) \approx O^*(1.4143^k)$. Отметим, что полученный алгоритм значительно улучшает предыдущую оценку времени работы $O^*(1.9276^k)$, представленную в работе [3].

Теорема 3.1. *Пусть G – геометрический граф, имеющий k пар пересекающихся ребер. Задача построения NST может быть решена для графа G за время $O^*(1.4143^k)$.*

Влияние *шага 8* на время работы алгоритма является объектом для дальнейшего исследования. С одной стороны, разбиение на подзадачи по компонентам связности после удаления мостов и пересекающих мосты ребер может увеличивать общее число подзадач. И число образуемых простых подзадач, и число образуемых составных подзадач может превышать $2^{k/2}$. В то же время размеры (число ребер и вершин) графов подзадач могут быть

значительно меньше, что потенциально ускоряет обработку подзадач и может приводить к меньшему итоговому времени работы алгоритма.

3.3 Алгоритм частичного перебора с отсечениями для решения задачи построения наибольшего непересекающегося ациклического подграфа

Алгоритм частичного перебора с отсечениями для решения задачи построения NST с опциональным множеством зафиксированных ребер может быть адаптирован для решения задачи построения $MNAS$ с опциональным множеством зафиксированных ребер.

Алгоритмы построения NST для простых подзадач могут быть легко модифицированы для построения $MNAS$. Несвязного подграфа, в котором явно отсутствует NST, может содержать $MNAS$ исходного графа, поэтому также должен обрабатываться. Следовательно, отбрасывать его, как это делается в алгоритме решения задачи построения NST, нельзя. Мост геометрического графа обязательно входит в NST, если оно существует, но может не входить в $MNAS$, поэтому *шаг 8* алгоритма решения задачи построения NST нельзя применять при построении $MNAS$. Также стоит отметить, что ранняя остановка перебора может быть осуществлена только в случае нахождения NST, т.к. в противном случае не гарантируется, что оставшиеся для перебора варианты не содержат большего непересекающегося ациклического подграфа.

Алгоритм частичного перебора с отсечениями для решения задачи построения $MNAS$ с опциональным множеством зафиксированных ребер.

Вход: геометрический граф G, опциональное множество N зафиксированных ребер.

Выход: $MNAS$ графа G, содержащий множество N зафиксированных ребер.

1. Множество фиксированных ребер N инициализируется пустым множеством, если оно не было подано на вход.
2. $MNAS$ G_{max} инициализируется графом $(V(G), N)$.
3. Очередь задач Q инициализируется очередью из одного элемента – пары (G, N). Очередь Q будет использоваться для хранения подзадач, которые необходимо решить.
4. Если очередь Q пуста или G_{max} является NST, то алгоритм завершает свою работу, возвращая G_{max} в качестве ответа.
5. Извлекаем из очереди Q пару (G_i, N_i).

6. Если геометрический граф G_i является непересекающимся, то выполняем построение $LNAS$ в подграфе G_i' с фиксированным множеством ребер N_i. Если подграф G_i' больше подграфа G_{max}, то $G_{max} \coloneqq G_i'$. Переходим на *шаг 4*.

7. Если возможно построить матроид пересечений на множестве ребер геометрического графа G_i (например, индекс пересечения графа G_i равен 1), то выполняем построение $MNAS$ в подграфе G_i' с фиксированным множеством ребер N_i, используя алгоритм решения задачи TMI. Если подграф G_i' больше подграфа G_{max}, то $G_{max} \coloneqq G_i'$. Переходим на *шаг 4*.

8. Выбираем некоторое ребро e_i графа G_i, которое имеет индекс пересечения, больший 0. Такое ребро всегда можно выбрать, т.к. в противном случае алгоритм остановился бы на *шаге 6*. Алгоритм выбора ребра e_i допускает вариации и зависит от конкретной реализации алгоритма. Например, можно брать ребро с наибольшим индексом пересечения или случайное ребро.

9. Если ребро e_i не образует цикл с зафиксированными ребрами N_i, то строим граф G_{i1}, удаляя из графа G_i все ребра, что пересекают ребро e_i, и добавляем в очередь Q подзадачу $(G_{i1}, N_i \cup \{e_i\})$.

10. Строим граф G_{i2} удалением ребра e_i и добавляем в очередь Q подзадачу (G_{i2}, N_i).

11. Возвращаемся на *шаг 4*.

Разработанный алгоритм оставляет за реализацией решение следующих вопросов:

1. Реализация очереди Q. Задачи могут извлекаться в порядке поступления, случайным образом, в соответствии с некоторым приоритетом или иным образом.

2. Конкретный алгоритм поиска $MNAS$ с множеством зафиксированных ребер, используемый на *шаге 6*.

3. Алгоритм выбора ребра на *шаге 8*.

4. Распараллеливание алгоритма: пары (G_i, N_i) могут обрабатываться независимо друг от друга.

5. Мемоизация и кэширование с целью исключения повторных вычислений.

Алгоритм решения задачи построения $MNAS$ с опциональным множеством зафиксированных ребер имеет ту же трудоемкость, что и алгоритм из *раздела 3.2*.

Теорема 3.2. *Пусть G – геометрический граф, имеющий k пар пересекающихся ребер. Задача построения $MNAS$ может быть решена для графа G за время $O^*(1.4143^k)$.*

3.4 Особенности программной реализации

Для разработки представленных в данной главе алгоритмов использовалась та же программная среда, что и для разработки алгоритма решения задачи TMI (см. *раздел 2.4*).

Алгоритмы были реализованы в следующем варианте: для хранения подзадач используется несколько потокобезопасных очередей с использованием work-stealing алгоритма извлечения подзадач из очереди; на *шаге 6* используется алгоритм построения NST и $MNAS$, основанный на системе непересекающихся множеств и работающий за время $O(m)$, где m – число ребер в графе; *шаг 8* включен в реализацию алгоритма решения задачи построения NST с опциональным множеством зафиксированных ребер, причем подзадачи решаются не рекурсивно, а с использованием общих с другими подзадачами очередей и дополнительных механизмов синхронизации; на *шаге 9 (8)* выбирается ребро с наибольшим индексом пересечения; реализация является параллельной, причем распараллеливание возможно на произвольное число потоков, не превышающее общее число образуемых подзадач; мемоизация и кэширование не производятся, однако используется ряд вспомогательных классов, позволяющих ускорить некоторые вычисления; на *шаге 7* лишь проверяется, имеет ли граф подзадачи индекс пересечения, равный 1, т.к. полная проверка, можно ли построить матроид пересечений на множестве ребер графа подзадачи, слишком трудоемка.

Параллельная реализация алгоритмов основана на общих очередях подзадач и использовании ForkJoinPool-а. Для упрощения реализации и повышения уровня параллелизации алгоритма решения задачи построения NST с опциональным множеством зафиксированных ребер между подзадачами не делается различия, т.е. подзадачи, формируемые на *шагах 8*, *10* и *11*, хранятся в одних и тех же очередях подзадач и обрабатываются единообразно. ForkJoinPool представляет собой реализацию пула потоков из стандартной библиотеки Java. Его ключевой особенностью являются эффективные механизмы порождения дочерних подзадач и ожидания завершения их обработки для агрегирования результатов, а также пониженное по сравнению с другими стандартными реализациями пула потоков потребление ресурсов процессора при решении задач, порождающих большое число подзадач. Такая эффективность ForkJoinPool-а обеспечивается за счет использования work-stealing механизма обработки подзадач: «простаивающие» потоки «воруют» (отсюда и название «work-stealing») подзадачи, порожденные задачами, обрабатываемыми другими потоками. Это позволяет одновременно

обрабатывать число задач, значительно превышающее число потоков в ForkJoinPool-е.

С целью уменьшения объема используемой алгоритмами памяти, а также уменьшения числа механизмов синхронизации, многие объекты являются immutable. Таковыми, например, являются реализации интерфейсов ребра, вершины, матроида. Это позволяет передавать и использовать эти объекты сразу несколькими потоками без использования дополнительных механизмов синхронизации. К тому же это позволяет использовать одни и те же immutable объекты в различных подзадачах (например, одни и те же инстансы вершин графа во всех графах подзадач), что избавляет от необходимости клонировать эти объекты и уменьшает объем потребляемой памяти.

Кэширование и мемоизация в явном виде не применяются, однако разработан и используется ряд вспомогательных классов, позволяющих ускорить некоторые вычисления, в том числе за счет исключения повторных вычислений. Например, для множества фиксированных ребер поддерживается система непересекающихся множеств, представляющая разбиение вершин на компоненты связности и позволяющая быстро ответить на вопрос, приводит ли добавление нового фиксированного ребра к образованию цикла.

Реализованные алгоритмы покрыты *unit*-тестами.

Ссылка на программный код реализованных алгоритмов находится в *Приложении А*.

3.5 Вычислительный эксперимент

Продемонстрируем работу разработанных алгоритмов на нескольких примерах.

На *рисунке 3.1* представлен граф, имеющий 20 вершин, 127 ребер, индекс пересечения 53, 1370 пар пересекающихся ребер. Решив 1150090 подзадач за 43.784 секунды, алгоритм частичного перебора с отсечениями для решения задачи построения NST построил в этом графе NST, представленное на *рисунке 3.2*.

Теперь рассмотрим задачу построения $MNAS$. На *рисунке 3.3* изображен граф со следующими параметрами: 40 вершин, 115 ребер, индекс пересечения 50, 1142 пары пересекающихся ребер. Алгоритм частичного перебора с отсечениями для решения задачи построения $MNAS$ построил $MNAS$ (*рисунок 3.4*) на 35 ребрах, решив 581420 подзадач за 33.934 секунды.

Заметим, что число решаемых алгоритмами подзадач и общее время работы алгоритмов в рассмотренных примерах выглядят очень оптимистично

на фоне оценки числа трудоемкости $O^*(1.4143^k)$. Действительно, $1.4143^{1000} > 10^{150}$.

Стоит отметить, что при одних и тех же параметрах графа (число вершин и ребер, индекс пересечения, число пар пересекающихся ребер) число решаемых подзадач и время работы алгоритма могут сильно варьироваться. Это обусловлено тем, что общее число рассматриваемых подзадач в значительной степени зависит от структуры пересечений ребер в геометрическом графе.

Также стоит отметить, что в случае существования NST в поданном на вход графе происходит ранняя остановка работы алгоритма, когда это NST обнаруживается. В случае отсутствия NST в заданном графе оба алгоритма вынуждены выполнить исчерпывающий поиск по всем подзадачам.

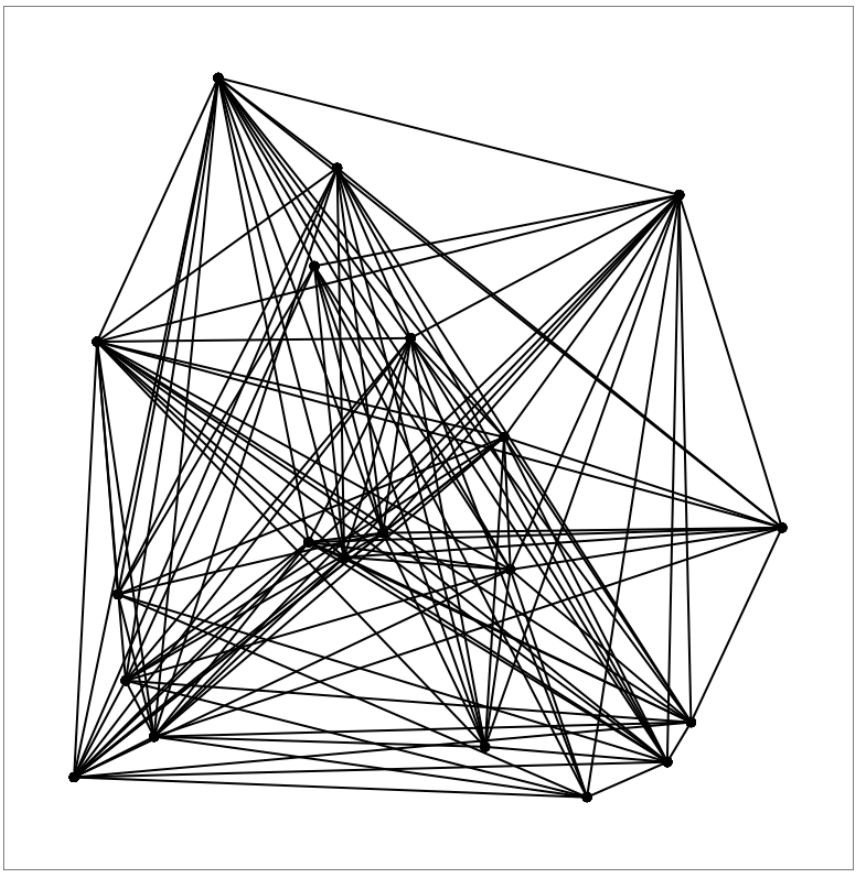

Рисунок 3.1 – Геометрический граф, содержащий *NST*

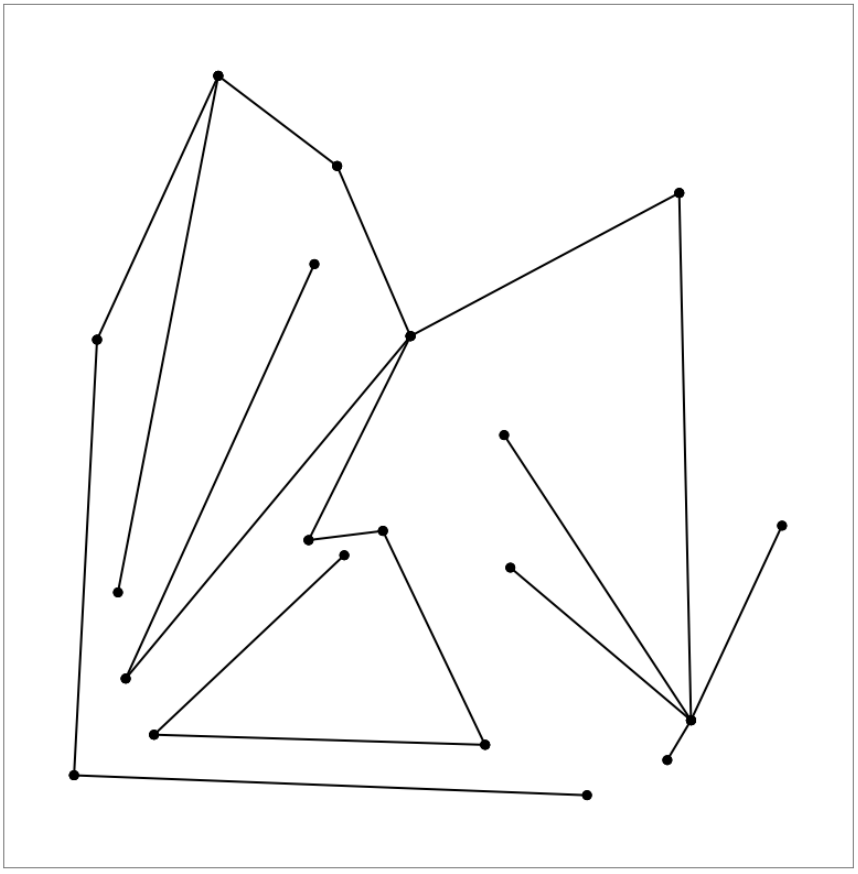

Рисунок 3.2 – *NST*

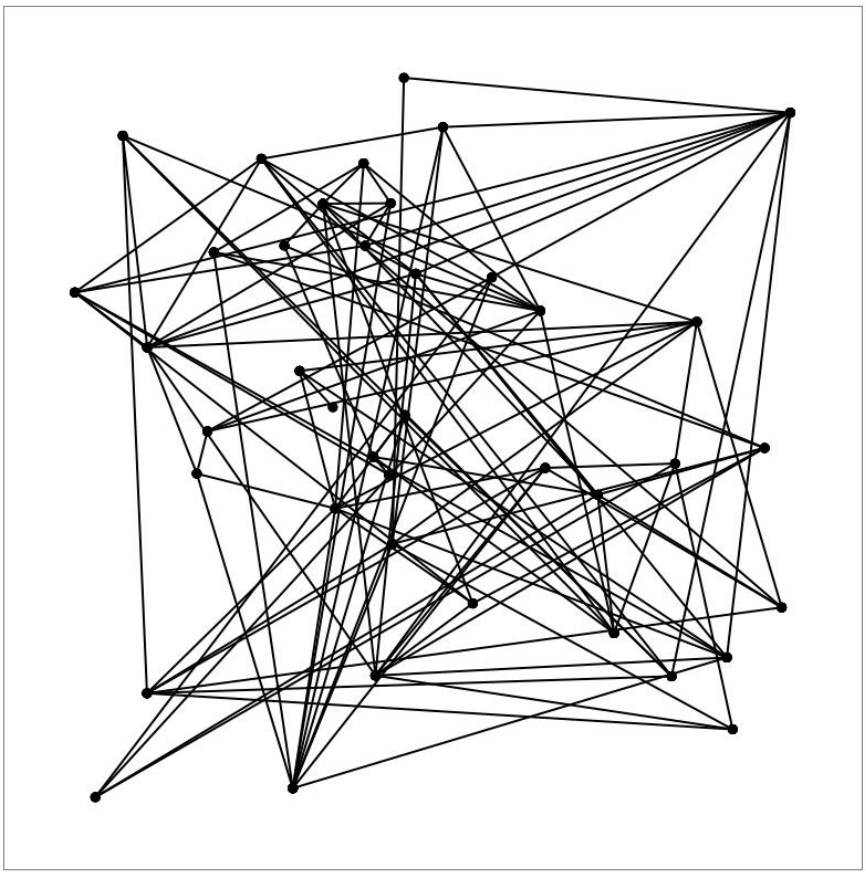

Рисунок 3.3 – Геометрический граф, не содержащий NST

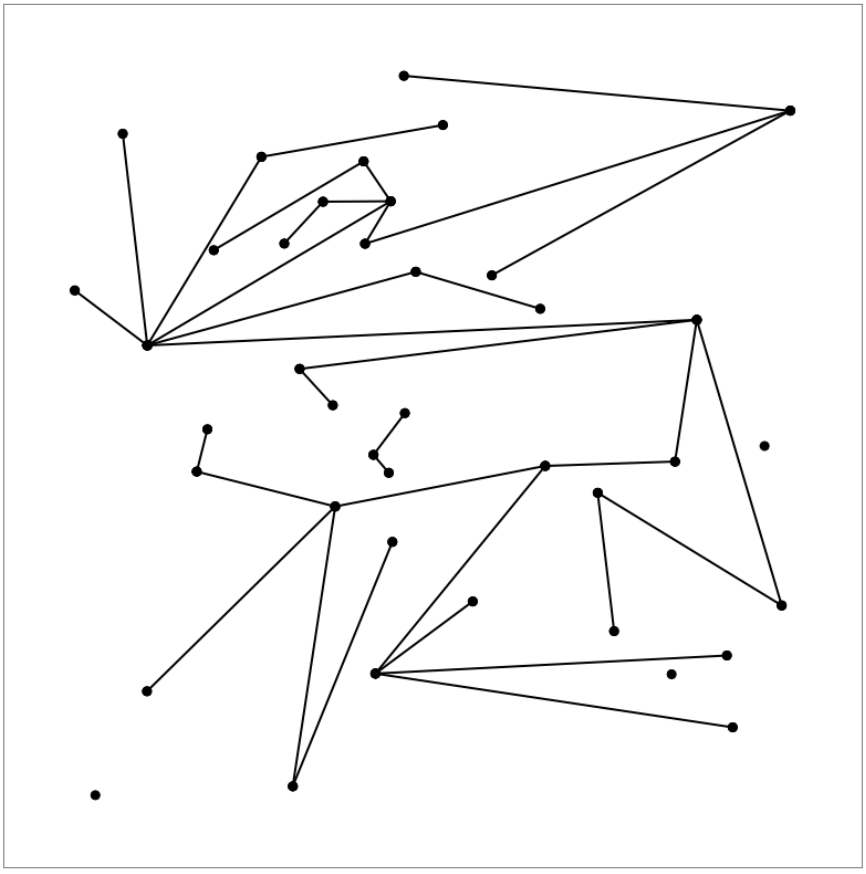

Рисунок 3.4 – $MNAS$

ГЛАВА 4
ЗАДАЧА ПОСТРОЕНИЯ НЕПЕРЕСЕКАЮЩЕГОСЯ ОСТОВНОГО ДЕРЕВА В ГЕОМЕТРИЧЕСКОМ ГРАФЕ, ОГРАНИЧЕННОМ МНОГОУГОЛЬНИКОМ

Напомним, что задача построения NST в выпуклом геометрическом графе может быть решена за время $O(n^3)$. В настоящей главе представлен алгоритм решения данной задачи за то же время, но для существенно более общего случая.

4.1 Геометрические графы, ограниченные многоугольником

Многоугольником называется часть плоскости, ограниченная замкнутой ломаной без самопересечений. Эту ломаную называют *границей многоугольника*. Вершины ломаной называются *вершинами многоугольника*, а ее отрезки – *сторонами многоугольниками*. Отрезки, соединяющие несмежные вершины многоугольника, называют *диагоналями*. Многоугольник называется *выпуклым*, если он лежит по одну сторону от любой прямой, содержащей его сторону (т.е. продолжения сторон многоугольника не пересекают других его сторон). В противном случае многоугольник называется *невыпуклым*. Диагональ многоугольника будем называть *допустимой*, если она лежит строго внутри данного многоугольника. В противном случае диагональ будем называть *недопустимой*. Очевидно, что все диагонали выпуклого многоугольника являются допустимыми.

Рассмотрим геометрический граф, вершины которого являются вершинами некоторого многоугольника, не обязательно выпуклого, при этом все ребра геометрического графа лежат внутри или на границе этого многоугольника. Такой геометрический граф будем называть *графом, ограниченным многоугольником*, или *ОМ-графом*. Соответствующий графу многоугольник будем называть *многоугольником* этого *графа*. Пример ОМ-графа изображен на *рисунке 4.1*. Отметим, что выпуклый геометрический граф, содержащий не менее 3 вершин, является ОМ-графом, т.к. его вершины лежат на выпуклой оболочке.

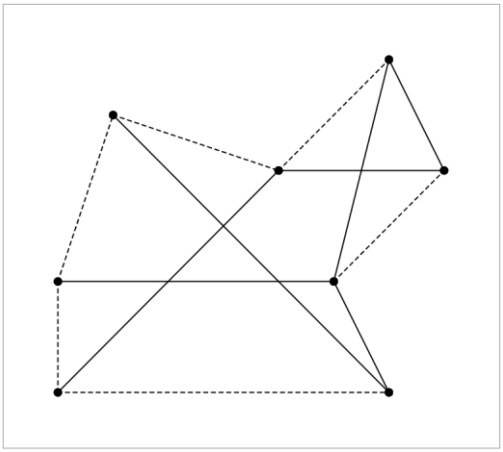

Рисунок 4.1 – Граф, ограниченный многоугольником

Триангуляцией многоугольника называется плоский геометрический граф, у которого:

1. множеством вершин является множество вершин многоугольника;
2. каждая внутренняя грань ограничена треугольником;
3. граница внешней грани совпадает с границей многоугольника.

4.2 Метод решения задачи

Заметим, что если в ОМ-графе существует NST, то существует и триангуляция многоугольника этого графа, для которой NST является подграфом. Таким образом, перебрав все триангуляции многоугольника ОМ-графа, можно проверить существование NST в этом графе.

Рассмотрим некоторый ОМ-граф $G = (V, E)$, $|V| = n$, $n \geq 3$ и соответствующий ему многоугольник M. Обозначим через S множество сторон многоугольника M. Пусть $T(M)$ – множество всех триангуляций многоугольника M.

Очевидно, что каждая сторона многоугольника M содержится ровно в одной внутренней грани для каждой триангуляции из множества $T(M)$. Пронумеруем вершины графа числами от 1 до n вдоль границы многоугольника M. Рассмотрим произвольную сторону этого многоугольника. Не нарушая общности, пусть это будет сторона C с вершинами 1 и 2. Проанализируем все треугольники, лежащие внутри многоугольника и содержащие сторону C. Таковым является множество треугольников $R_{1,2}$, содержащее треугольники $\Delta_{1,2,j}$, где $j \in \{3,4,\dots,n\}$, а диагонали $1,j$ и $2,j$ являются допустимыми. Пусть $T(\Delta_{p,q,r})$ – множество триангуляций из $T(M)$, содержащих $\Delta_{p,q,r}$. Не трудно заметить, что $\bigcup T(\Delta_i) = T(M)$, $\Delta_i \in R_{1,2}$.

Зафиксируем некоторый треугольник $\Delta_{1,2,j}$. Он разбивает множество вершин V графа на два множества $V_1 = \{2,3,\dots,j\}$ и $V_2 = \{j, j+1,\dots,n,1\}$, $V_1 \cup V_2 = V$. Рассмотрим два подграфа $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$ ОМ-графа G, индуцируемые множествами вершин V_1 и V_2 соответственно. Заметим, что каждый из подграфов G_1, G_2 является или геометрическим графом на двух вершинах, или ОМ-графом, допускающим дальнейшее аналогичное разбиение. Возьмем произвольные непересекающиеся остовные леса F_1 и F_2 в подграфах G_1 и G_2 соответственно. Непересекающийся остовный лес графа $G' = (V, E_1 \cup E_2)$ получается из лесов F_1 и F_2 объединением компонент, содержащих вершину j, а также вершины 1 и 2 соответственно при условии наличия ребра $e_{1,2}$ в ОМ-графе G. В отношении подграфов G_1 и G_2 возможны следующие случаи:

1. В подграфах G_1, G_2 существуют NST T_1, T_2 соответственно. Очевидно, что тогда и графы G', G содержат NST $T_1 \cup T_2$;
2. Один из подграфов содержит NST T, а другой – такой непересекающийся остовный лес F, состоящий из двух компонент, что вершины $1, j$ или $2, j$ принадлежат разным компонентам. Если в графе G существует ребро $e_{1,2}$, то и в графе G', и в графе G существует NST $T \cup F \cup e_{1,2}$. В противном случае в графе G' существует такой непересекающийся остовный лес $\mathrm{T} \cup F$, состоящий из двух компонент, что вершины $1, 2$ принадлежат разным компонентам.
3. Если подграфы G_1, G_2 не удовлетворяют условиям 1 и 2, то не трудно заметить, что в графе G' не существует NST. Более того, в графе G' также не существует такого непересекающегося остовного леса, состоящего из двух компонент, что вершины 1 и 2 принадлежат разным компонентам.

Таким образом, существование NST в ОМ-графе G может быть проверено следующим образом. Фиксируется некоторая сторона C многоугольника M. Перебираются на множестве вершин V все допустимые треугольники Δ_i, содержащие сторону C. Каждый треугольник Δ_i разбивает граф G на два подграфа G_1, G_2, для которых задача решается рекурсивно. Для тривиального случая, когда геометрический граф содержит ровно две вершины, существование NST или непересекающегося остовного леса, состоящего из двух компонент, определяется наличием ребра между этими двумя вершинами. Если для хотя бы одного разбиения существует NST, то NST существует и для графа G. В противном случае не существует триангуляции многоугольника M, содержащей NST ОМ-графа G в качестве подграфа, а значит, ОМ-граф G не содержит NST.

4.3 Описание алгоритма

Для ОМ-графа G задача построения NST может быть решена с использованием метода динамического программирования. Рассмотрим матрицу S размера $n \times n$. Элементу $s_{i,j}$, $i < j$ поставим в соответствие подграф $G_{i,j}$ графа G, индуцируемый множеством вершин с номерами $\{i, i+1, \dots, j\}$. Если диагональ i,j является допустимой, то в $s_{i,j}$ будем хранить 2, если подграф $G_{i,j}$ содержит NST; 1, если подграф $G_{i,j}$ содержит такой непересекающийся остовный лес, состоящий из двух компонент, что вершины с номерами i и j принадлежат разным компонентам; 0 в противном случае. Если диагональ i,j является недопустимой, то в $s_{i,j}$ будем хранить -1. На первом, восходящем этапе алгоритма производится последовательное заполнение матрицы S. Затем проверяется существование NST в графе G. NST в графе G существует тогда и только тогда, когда $s_{1,n} = 2$. Если NST существует, то оно может быть восстановлено на втором, нисходящем этапе алгоритма, предполагающем рекурсивно восстанавливать NST путем пробега по матрице S и определения разбиений, приводящих к построению NST.

Алгоритм построения NST в ОМ-графе.

Вход: ОМ-граф $G = (V, E)$, $|V| = n, n > 3$ и многоугольник M графа G.

Выход: NST графа G, если оно существует в графе G.

1. Выполняется нумерация вершин геометрического графа G числами от 1 до n вдоль границы многоугольника.
2. Если многоугольник M ОМ-графа G не является выпуклым, то определяем для каждой диагонали многоугольника M, является ли она допустимой. В выпуклом многоугольнике все диагонали являются допустимыми.
3. Выполняем *шаги 4-6* в цикле по i от $n-1$ до 1.
4. Если $e_{i,i+1} \in E$, то $s_{i,i+1} \coloneqq 2$. Иначе $s_{i,i+1} \coloneqq 1$.
5. Выполняем *шаг 6* в цикле по j от $i+2$ до n.
6. Если диагональ i,j является недопустимой, то $s_{i,j} \coloneqq -1$. Иначе $s_{i,j} \coloneqq \max_{i<k<j} f(s_{ik}, s_{kj})$, где $f(a,b)$, $a,b \in \{-1,0,1,2\}$ принимает значение -1, если хотя бы одно из значение a, b равно -1; 2, если $a = b = 2$ или $e_{i,j} \in E$ и ($a = 1$, $b = 2$ или $a = 2$, $b = 1$); 1, если $e_{i,j} \notin E$ и ($a = 1$, $b = 2$ или $a = 2$, $b = 1$); 0 в противном случае.
7. Если $s_{1,n} \neq 2$, то в ОМ-графе G отсутствует NST. Алгоритм завершает свою работу.
8. Изначально множество ребер NST пусто: $E_T \coloneqq \emptyset$.
9. $i \coloneqq 1, j \coloneqq n$.
10. Если $j = i + 1$ и $e_{i,j} \in E$, то $E_T \coloneqq E_T \cup e_{i,j}$.

11. Если $j > i + 1$, то находим $k' = \text{argmax}_{i<k<j} f(s_{ik}, s_{kj})$, где $f(a, b)$ определена на шаге 6. Если $s_{i,j} = 2$ и ($s_{i,k'} = 1$, $s_{k',j} = 2$ или $s_{i,k'} = 2$, $s_{k',j} = 1$), то $E_T \coloneqq E_T \cup e_{i,j}$. Повторяем *шаги 10-11* для $i = i$, $j = k'$ и $i = k', j = j$.

12. Возвращается E_T в качестве ответа.

Оценим трудоемкость представленного алгоритма. Нумерация вершин ОМ-графа G на *шаге* 1 может быть выполнена за время $O(n)$. Является ли многоугольник ОМ-графа G выпуклым, можно также определить за время $O(n)$. Трудоемкость проверки всех диагоналей многоугольника на допустимость составляет $O(n^3)$ для невыпуклого многоугольника. Таким образом, *шаг 2* выполняется за время $O(n)$ для ОМ-графа, многоугольник которого является выпуклым, $O(n^3)$ в противном случае. Если информация о выпуклости многоугольника заданного графа подается на вход алгоритму, то для ОМ-графа с выпуклым многоугольником *шаг 2* будет выполнен за время $O(1)$. *Шаги 3-6* представляют собой 3 вложенных цикла по вершинам графа и имеют трудоемкость $O(n^3)$. *Шаги 7-9* и *12* являются элементарными и занимают время $O(1)$. Не трудно заметить, что шаги *10* и *11* будут вызываться $O(n)$ раз каждый. Трудоемкость *шага 10* составляет $O(1)$, *шага 11* – $O(n)$. Отметим, что трудоемкость *шага 11* можно уменьшить до $O(1)$, если предпросчитывать и сохранять k' на *шаге 6*. При этом, правда, потребуется $O(n^2)$ дополнительной памяти для хранения k' для всех пар i, j.

Теорема 4.1. *Пусть G – ОМ-граф на n вершинах. Задача построения NST в графе G может быть решена за время $O(n^3)$ с использованием метода динамического программирования.*

4.4 Особенности программной реализации

Для разработки представленного в данной главе алгоритма использовалась та же программная среда, что и для разработки алгоритма решения задачи TMI (см. *раздел 2.4*).

Из особенностей реализации стоит отметить использование библиотеки *JTS* (JTS Topology Suite). *JTS* является java-библиотекой с открытым исходным кодом. Она предоставляет объектную модель для $2D$ геометрии вместе с набором основных геометрических функций.

JTS библиотека используется для определения допустимых диагоналей в многоугольнике ОМ-графа. Для этого сначала строится замкнутая кривая (класс *LinearRing*) на вершинах ОМ-графа в порядке их нумерации, на основе которой выполняется построение объекта класса *Polygon*. Данный класс

библиотеки *JTS* позволяет проверить, лежит ли диагональ (объект класса *LineString*) внутри многоугольника, при помощи метода *covers*. Этот метод используется для проверки каждой из диагоналей многоугольника геометрического графа на допустимость.

Стоит отметить, что во время выполнения *шагов 3-6* алгоритма будет использована каждая из диагоналей, причем не однократно. Ленивая проверка допустимости диагонали с мемоизацией не даст выигрыша в производительности. Вычисление допустимости диагонали по необходимости без мемоизации позволит сэкономить $O(n^2)$ памяти, но при этом ухудшит время работы алгоритма до $O(n^4)$ из-за увеличения трудоемкости проверки диагонали на допустимость на этапе решения подзадач (*шаги 3-6*) с $O(1)$ до $O(n)$.

Алгоритм реализован в варианте, когда k' каждый раз вычисляется на *шаге 11*, а не предпросчитывается на *шаге 6* с последующим сохранением. Также стоит отметить, что для хранения представления многоугольника ОМ-графа используется список вершин в порядке обхода многоугольника по или против часовой стрелки.

Реализованный алгоритм покрыт *unit*-тестами.

Ссылка на программный код реализованного алгоритма находится в *Приложении А*.

4.5 Вычислительный эксперимент

Продемонстрируем работу алгоритма для решения задачи построения NST в ОМ-графе на примере.

Представленный на *рисунке 4.2* ОМ-граф имеет 100 вершин и 1509 ребер. Его индекс пересечения равен 941, а суммарное число пересекающихся пар ребер составляет 401981. Отметим, что многоугольник рассматриваемого ОМ-графа не является выпуклым, т.е. этот геометрический граф является невыпуклым. Заметим также, что алгоритму частичного перебора с отсечениями решения задачи построения NST с трудоемкостью $O^*(1.4143^k)$, представленному в *разделе 3.2*, не под силу за разумное время построить NST в таком графе. Напомним, что k в оценке трудоемкости обозначает число пар пересекающихся ребер, т.е. 401981 для рассматриваемого ОМ-графа.

Реализованный в данной главе алгоритм построил NST для поданного на вход ОМ-графа всего за 1.7 секунды. Полученное в результате работы алгоритма NST изображено на *рисунке 4.3*.

Сравним скорость работы разработанных в *главах 3* и *4* алгоритмов на более простом примере, решаемом алгоритмом частичного перебора с отсечениями за разумное время.

На *рисунке 4.4* представлен OM- граф, имеющий 20 вершин и 133 ребра. Индекс пересечения этого графа равен 67, а число пар пересекающихся ребер – 2438.

Полиномиальный алгоритм для решения задачи построения NST в OM-графе построил NST для заданного OM-графа, изображенное на *рисунке 4.5*, за 0.01 секунды.

Экспоненциальный алгоритм частичного перебора с отсечениями для решения задачи построения NST в произвольном геометрическом графе построил NST для заданного графа, решив за 6.212 секунд 193208 подзадач. Это NST, отличающееся от построенного первым алгоритмом, представлено на *рисунке 4.6*.

Алгоритм частичного перебора с отсечениями имеет экспоненциальную трудоемкость, поэтому, что ожидаемо, работает значительно медленнее полиномиального алгоритма. Разница во времени работы для рассмотренного OM- графа составила 621.2 раза. При этом важно отметить, что время работы алгоритма частичного перебора с отсечениями значительно меньше худшего ожидаемого времени работы, которое дает оценка трудоемкости $O^*(1.4143^k)$.

Несмотря на значительно большее время работы, алгоритм частичного перебора с отсечениями, в отличие от полиномиального алгоритма, решает задачу построения NST для произвольного геометрического графа, а не только для класса OM-графов, а также позволяет фиксировать множество ребер, которое обязано входить в NST.

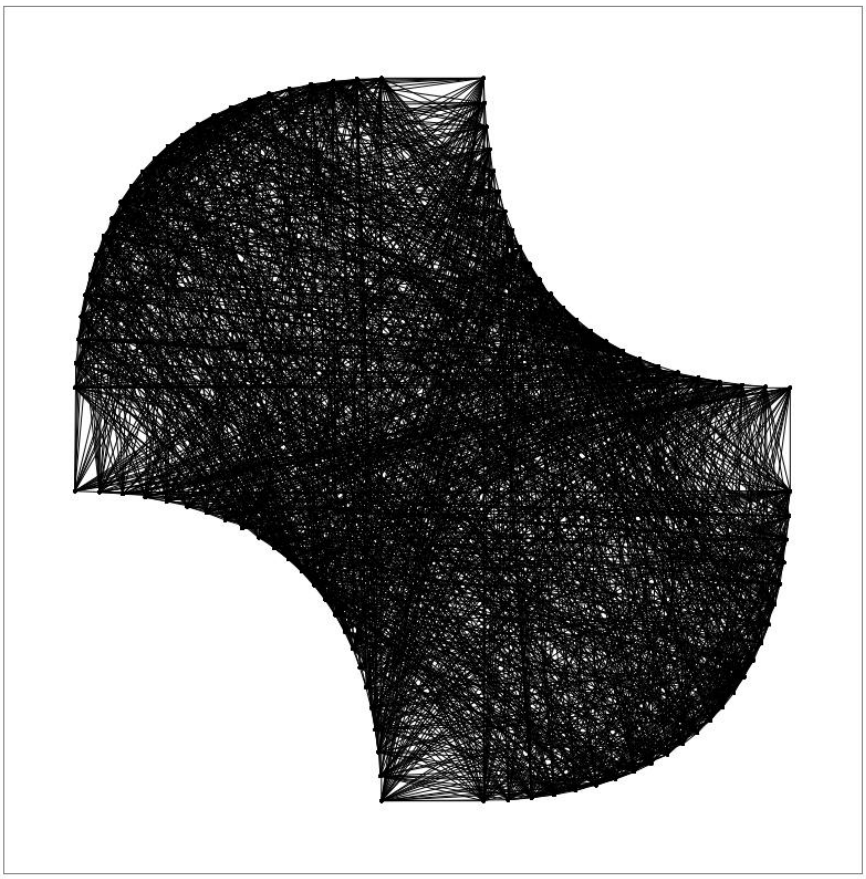

Рисунок 4.2 – OM-граф

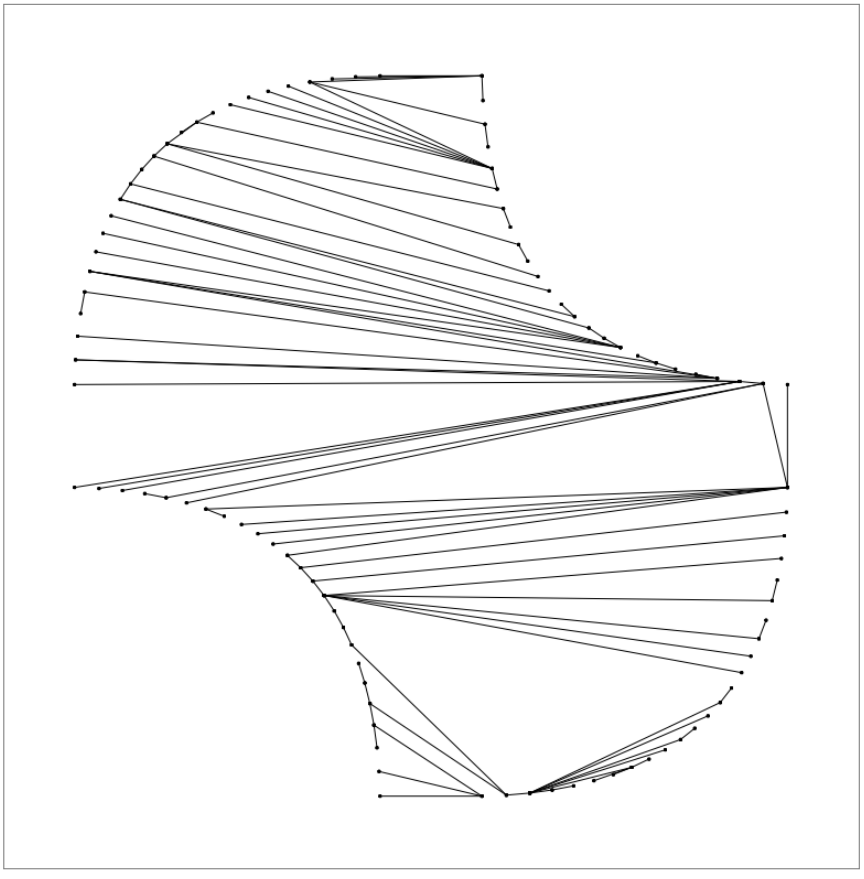

Рисунок 4.3 – NST в OM-графе

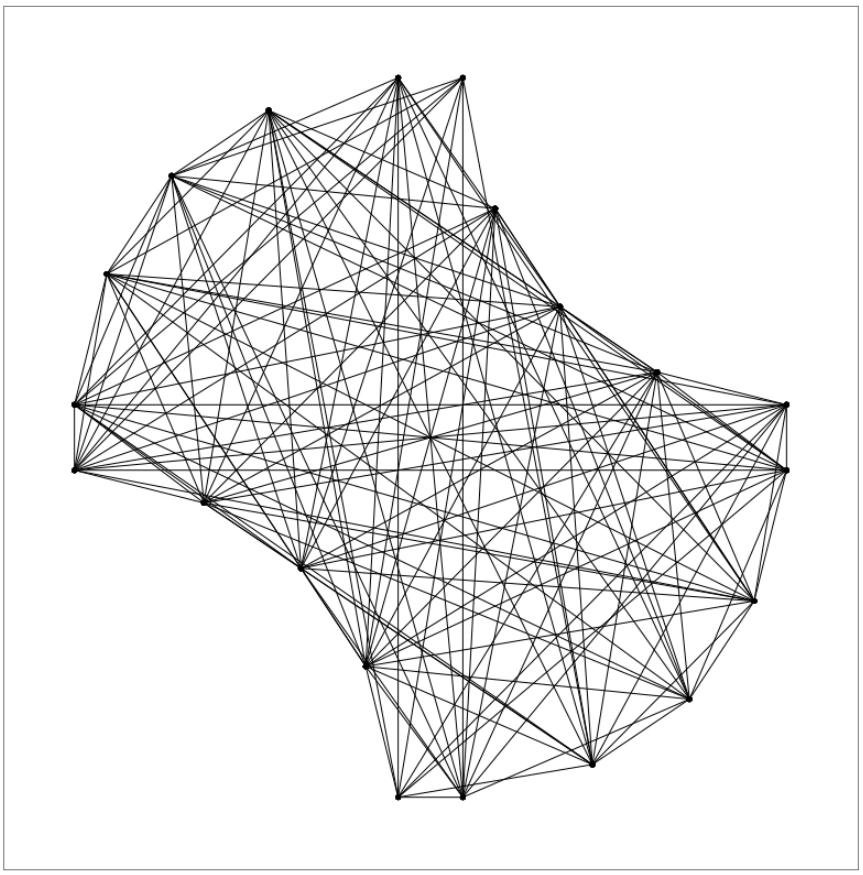

Рисунок 4.4 – OM-граф

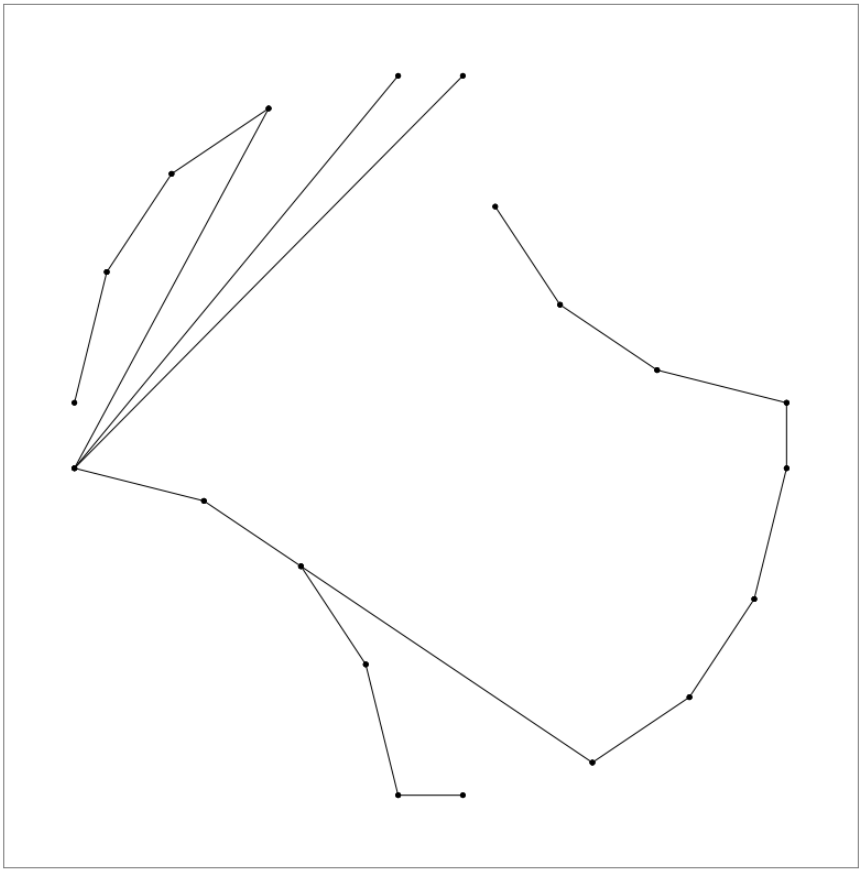

Рисунок 4.5 – NST, построенное полиномиальным алгоритмом

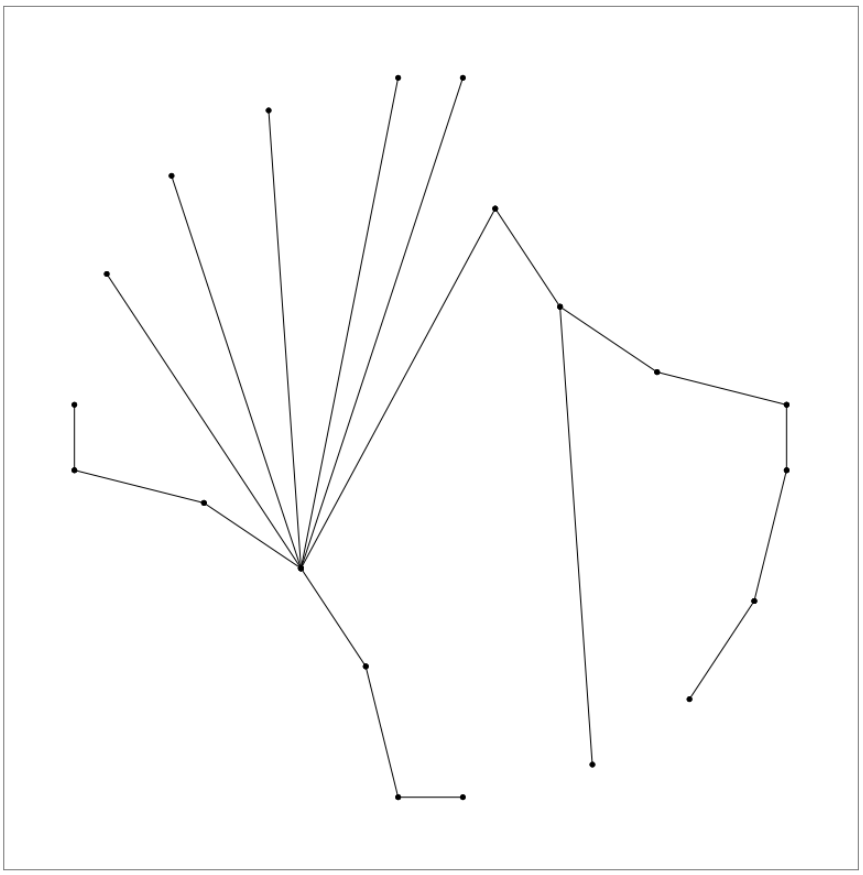

Рисунок 4.6 – NST, построенное алгоритмом частичного перебора с отсечениями

ЗАКЛЮЧЕНИЕ

В рамках диссертационной работы изучены материалы по проблеме построения больших непересекающихся ациклических подграфов в геометрических графах, включая:

– точные экспоненциальные алгоритмы построения непересекающихся ациклических подграфов в геометрических графах;
– параметризованные алгоритмы построения непересекающихся ациклических подграфов в геометрических графах;
– полиномиально разрешимые случаи задачи построения непересекающихся ациклических подграфов в геометрических графах;
– задача пересечения двух матроидов.

Основные результаты работы состоят в следующем:

– разработан и программно реализован алгоритм решения задачи пересечения двух матроидов с возможностью фиксирования множества элементов, которое обязано входить в пересечение;
– предложен способ применения алгоритма решения задачи пересечения двух матроидов с возможностью фиксирования множества обязательных элементов для решения задачи построения непересекающегося остовного дерева и задачи построения наибольшего непересекающегося ациклического подграфа в геометрическом графе, допускающем построение матроида пересечений на множестве ребер, с возможностью указания множества обязательных для вхождения в искомый подграф ребер;
– разработан и программно реализован алгоритм частичного перебора с отсечениями для решения задачи построения непересекающегося остовного дерева в геометрическом графе, дана оценка трудоемкости разработанного алгоритма;
– разработан и программно реализован алгоритм частичного перебора с отсечениями для решения задачи построения наибольшего непересекающегося ациклического подграфа в геометрическом графе, дана оценка трудоемкости разработанного алгоритма;
– введено понятие графа, ограниченного многоугольником, разработан и программно реализован точный полиномиальный алгоритм решения задачи построения непересекающегося остовного дерева в графе, ограниченном многоугольником;
– проведены вычислительные эксперименты, целью которых является демонстрация работы разработанных алгоритмов и их сравнение.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Jansen, K. The complexity of detecting crossingfree configurations in the plane / K. Jansen, G. J. Woeginger. – 1993. – P. 580–595.
2. Knauer, C. Configurations with few crossings in topological graphs / C. Knauer, E. Schramm, A. Spillner, A. Wolff // International. Symposium on Algorithms and Computations (ISAAC), 2005. – Springer, 2005. – v. 3827. − P. 604−613.
3. Halldorsson, M. Parameterized algorithms and complexity of non-crossing spanning trees / M. Halldorsson, C. Knauer, A. Spillner, T. Tokuyama // 10th Workshop on Algorithms and Data Structures (WADS), Canada, 2007 – Halifax, 2007. – P. 410−421.
4. Knauer, C. Fixed-parameter algorithms for finding crossing-free spanning trees in geometric graphs / C. Knauer, A. Spillner. – Jena, 2006. – 11 p.
5. Бенедиктович, В.И. Алгоритмы и сложность построения некоторых комбинаторно-геометрических конфигураций: дис. канд. физ.-мат. наук: 01.01.09 / В.И. Бенедиктович. – Минск, 2001. – 97 с.
6. Rivera-Campo, E. A sufficient condition for the existence of plane spanning trees on geometric graphs / E. Rivera-Campo, V. Urrutia-Galicia // Computational Geometry: Theory and Applications. – 2013. – № 1. – P. 1–6.
7. Rivera-Campo, E. Proceedings of the Japanese Conference on Discrete and Computational Geomentry / E. Rivera-Campo // Conference on Discrete and Computational Geometry, Japan, 1998. – 2000. – v. 1763. – P. 274-277.
8. Бенедиктович, В.И. Локальный признак существования плоского остовного дерева в геометрическом графе / В.И. Бенедиктович // Известия Национальной академии наук Беларуси. Серия Физико-математических наук. – 2014. − № 2. – С. 58−63.
9. Keller, C. Blockers for non-crossing spanning trees in complete geometric graphs. / C. Keller, M.A. Perles, E. Rivera-Campo, V. Urrutia-Galicia // Thirty Essays on Geometric Graph Theory. – Springer, 2013. – P. 383-398.
10. Aichholzer, O. Edge-Removal and Non-Crossing Configurations in Geometric Graphs / O. Aichholzer, S. Cabello, R. Fabila-Monroy, D. Flores-Penaloza, T. Hackl // Discrete Mathematics and Theoretical Computer Science. – 2010. – №12 (1). – P. 75-86.
11. Alon, N. Long non-crossing configurations in the plane / N. Alon, S. Rajagopalan, S. Suri // Fundamenta Informaticae 22. – 1995. – P. 385-394.
12. Oxley, J.G. Matroid Theory / J.G. Oxley. – Oxford University Press, 2006. – 532 p.

ПРИЛОЖЕНИЕ А

Ссылка на реализацию разработанных алгоритмов

https://github.com/SergeyZyazyulkin/NonCrossingSubgraphs