Алгоритм быстрого построения минимального охватывающего дерева для множества точек в конечномерном псевдометрическом пространстве

Д. В. Полевой, В. В. Постников, А. В. Усков

Данная статья посвящена вопросу эффективного построения минимального охватывающего дерева для конечного множества точек в конечномерном метрическом или псевдометрическом пространстве. Предлагаются алгоритм и вспомогательные структуры данных, которые позволяют вычислительно эффективно решать поставленную задачу. Временные характеристики предложенного алгоритма исследуется экспериментально с использованием искусственных и реальных данных.

1. Задача построения минимального остова на множестве точек

Пусть в n-мерном пространстве задана функция расстояния между точками этого пространства и конечное множество точек V, которые могут рассматриваться, как вершины графа. Отрезки, соединяющие точки из V могут рассматриваться, как ребра некоторого неориентированного графа, а расстояние между этими точками, весом соответствующих ребер. Тогда задача построения минимального охватывающего дерева имеет следующую формальную постановку.

Пусть имеется связный неориентированный граф $G = (V, E)$, в котором V — множество вершин, E — множество ребер. Для каждого ребра графа (u, v) задан неотрицательный вес w(u, v). Задача состоит в нахождении подмножества $T \subset E$, связывающего все вершины, для которого суммарный вес

$$\omega(T)= \sum_{(u,v)\in T} \omega(u,v)$$

минимален. Такое подмножество T можно считать деревом (в любом цикле одно ребро можно удалить, не нарушая связности). Связный подграф

графа G, являющийся деревом и содержащий все вершины, называют покрывающим деревом (*spanning tree*) этого графа. (Иногда используют термин «остовное дерево», или, короче, «остов».) Минимальным покрывающим деревом (*minimal spanning tree*) называют покрывающее дерево, имеющее минимально возможный вес.

Общая схема всех алгоритмов построения минимального покрывающего дерева такова. Искомый остов строиться постепенно: к изначально пустому множеству A ребер на каждом шаге будет добавляться одно ребро. Множество A всегда является подмножеством некоторого минимального покрывающего дерева. Ребро (u, v), добавляемое на очередном шаге, выбирается так, чтобы не нарушать этого свойства: $A \cup \{(u, v)\}$ тоже должно быть подмножеством минимального остовного дерева. Назовем такое ребро безопасным ребром для A.

MST-Generic (G, w)
1 $A \leftarrow \varnothing$
2 **while** (пока) A не является остовом
3 **do** найти безопасное ребро (u, v) для A
4 $A \leftarrow A \cup \{(u, v)\}$
5 **return** A

Наиболее известными алгоритмами построения минимального остовного дерева являются алгоритма Крускала и Прима. Подробный разбор этих алгоритмов можно найти, например, в работе [2], а здесь приведем уже готовые оценки времени работы. Лучшее время работы алгоритма Крускала составляет

$$O(|E| \log|E|),$$

а лучшее время работы алгоритма Прима

$$O(|E| + |V| \log|V|),$$

где |V| — количество вершин графа G, |E| — количество ребер графа G.

Граф называется полносвязным, если каждая его вершина соседствует с каждой. Полносвязный граф содержит $|E| = |V||V-1|/2$ ребер. Наиболее эффективная реализация алгоритма Крускала для вычисления минимального покрывающего дерева на полносвязном графе имеет время работы

$$O(|E| \log|E|) = O(|V|(|V| - 1)/2 \log|V|(|V| - 1)/2) = O(|V|^2 \log|V|).$$

Для алгоритма Прима лучшее время работы на полносвязном графе

$$O(|E| + |V|\log|V|) = O(|V|(|V| - 1)/2 + |V|\log|V|) = O(|V|^2).$$

Однако, во многих задачах, которые формально сводятся к построению минимального остовного дерева на полносвязном графе этот результат можно улучшить, если отвлечься от чисто графовой постановки задачи.

2. Обзор методов решения задачи поиска ближайшего соседа

Пусть $P = \{p_1, p_2, \ldots, p_n\}$ — множество из n точек в метрическом пространстве $M = (\nu, d(\cdot,\cdot))$, где ν — это d-мерное векторное пространство и $d : \nu \times \nu \rightarrow R$ обозначает меру близости (удаленности).

Пусть даны множество точек P из n точек в M и исходная точка поиска $q \in \nu$. Назовем ближайшим соседом для точки q такую точку $p \in P$, что для любой другой точки $p' \in P$ выполняется

$$d(q, p) \leq d(q, p').$$

Задача поиска ближайшего соседа (nearest neighbor search) состоит в нахождении ближайшего соседа для произвольной исходной точки поиска q. Если при этом разрешены добавление точек к множеству P, или их удаление, то задача называется динамической. Если множество точек данных фиксировано и не изменяется, то это статическая задача. Далее рассматривается только статическая задача поиска ближайшего соседа.

Использование вспомогательных поисковых структур позволяет значительно убыстрить процесс поиска ближайшего соседа. Проблема имеет оптимальное решение в случае 1-мерного пространства с использованием B-деревьев и их вариантов [5]. Для пространств большей размерности разработаны различные подходы [6], [15], [16], [17], [17]. Наиболее популярными вспомогательными структурами данных являются различные индексные деревья [4], [11], [12], [14], [18].

Рассмотрим подробнее один из возможных подходов. HS алгоритм, предложенный Hjaltson и Samet [7], использует простую идею, корни которой восходят ко многим работам (таким как k-d-деревья [8], или BBD-деревья). Пусть дана исходная точка поиска q, идея состоит в том, чтобы разбить все пространство на блоки, и искать в этих блоках в порядке воз-

растания расстояния до точки q. Блок — это область пространства, которая может разбиваться на более мелкие блоки, в зависимости от количества точек, попавших в данный блок. Более формально: пусть MINDIST(q, B) обозначает минимальное расстояние от точки q до любой точки блока B. MINDIST(q, B) = 0 если точка лежит внутри блока B. Функцию MINDIST(q, B) мы считаем расстояние от точки q до блока B. Заметим, что по определению в блоке B нет может быть точек с расстоянием до точки q меньшим, чем MINDIST(q, B).

Алгоритм ищет в иерархической структуре индексов, чтобы найти блок, который содержит точку q. Спускаясь вниз по дереву будем помещать блоки, встречающиеся на пути поиска, в очередь с приоритетом. В очереди блоки сортируются в соответствии с функцией MINDIST(q, B). Алгоритм посещает блоки в порядке уменьшения расстояния от дочки q. Если блок соответствует внутреннему узлу дерева поиска, тогда все соответствующие ему блоки помещаются в очередь. Если блок является терминальным — вычислим расстояние от q до каждой точки блока. Расстояние до ближайшей точки хранится в переменной l_{min}. Алгоритм заканчивает свою работу, когда расстояние до очередного блока, извлеченного из очереди, станет больше, чем l_{min}. HS алгоритм является оптимальным в том смысле, что он посещает только те блоки, которые пересекаются с шаром радиуса d_{min} (расстояние до ближайшего соседа) и центром в исходной точке q. Однако, в худшем случае, алгоритм вынужден просмотреть все точки в каждом блоке.

3. Алгоритм построения минимального остова

Задачи нахождения k-ближайших соседей и построения минимального охватывающего дерева решались в рамках исследования возможности применения кластеризации компонент связности на ранних этапах сегментации изображения страницы. Поскольку исследовались как локальные, так и глобальные групповые свойства компонент связности, то было принято решение разработать гибкий алгоритм, легко применимый ко всем основным метрикам.

Очевидно, что каждой компоненте связности соответствует точка в 4-мерном вещественном пространстве.

В этой работе при реализации алгоритмов в качестве функции расстояния между компонентами связности, использовалось выражение

$$w(a,b) = |dx|+|dy|.$$

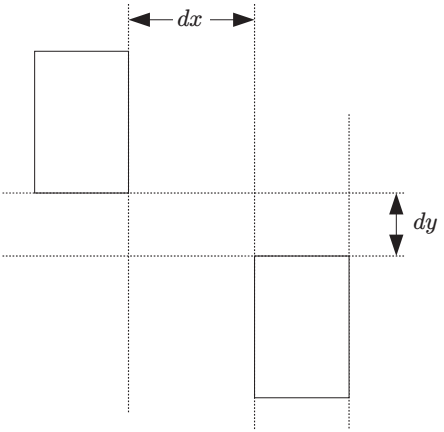

Рис. 1. Определение расстояния

Такая функция расстояния не является метрикой, поскольку не удовлетворяет следующим аксиомам:

$$w(a, b) = 0 \Leftrightarrow a = b \quad \text{и } w(a, b) \le w(a, c) + w(b, c), \quad \forall\, a, b, c.$$

Однако эта функция обладает одним важным свойством, которым мы и воспользуемся в дальнейшем для построения эффективной индексной структуры данных.

Так же, как и алгоритмы Крускала и Прима, предлагаемый алгоритм построения минимального покрывающего дерева на полносвязном графе следует общей схеме алгоритма построения минимального остовного дерева. Этот алгоритм является модификацией алгоритма Прима. Так же, как и в алгоритме Прима, растущая часть остова представляет собой дерево (множество ребер которого есть A). Формирование дерева начинается с произвольной корневой вершины r. На каждом шаге добавляется ребро наименьшего веса среди ребер, соединяющих вершины этого дерева с вершинами не из дерева. В работе [2] показано, что такие ребра являются безопасными для A, так что в результате получается минимальное покрывающее дерево.

При реализации алгоритма важно быстро выбирать легкое ребро и принципиальным отличием от алгоритма Прима является способ выбора такого ребра. Функция Find-Nearest(u, A, G) осуществляет поиск и возвращает ближайшего соседа для вершины u из множества вершин $G \setminus A$.

Если такового не существует, возвращается NIL. Доопределим весовую функцию следующим образом: $w(u, NIL) = \infty$ для любой вершины u. Алгоритм получает на вход связный граф G и корень r минимального остова. В ходе алгоритма в очереди с приоритетами Q хранятся все вершины, которые уже вошли в минимальное остовное дерево. Приоритет вершины v в очереди Q определяется значением ключа key[v], которое равно минимальному весу ребер, соединяющих v с вершинами не из дерева A. Поле $\pi[v]$ для вершины дерева $v \in Q$ указывает на вершину $u \in G \setminus A$, являющуюся ближайшей к вершине v в момент определения $\pi[v]$ ($(v,\pi[v])$ — одно из таких ребер, если их несколько). Вершины из очереди не удаляются. Минимальный остов A хранится явно. Построение минимального остовного дерева начинается с произвольной вершины графа r.

MST-KD(G, w, r)
1 A ← r
2 Q ← r
3 π[r] ← Find-Nearest(r, A, G)
4 key[r] ← w(r, π[r])
5 **while** A ≠ V[G]
6 **do** u ← Minimum(Q)
7 **if** π[u]∈ A
8 **then** π[u] ← Find-Nearest(u, A, G)
9 key[u] ← w(u, π[u])
10 **else** A ← A∪{(u, π[u])}
11 Q ← π[u]
12 π[π[u]] ← Find-Nearest(π[u], A, G)
13 key[π[u]] ← w(π[u], π[π[u]])
14 π[u] ← Find-Nearest(u, A, G)
15 key[u] ← w(u, π[u])
16 **return** A

После исполнения строк 1–4 дерево состоит из единственной вершины r, а в очереди Q содержится единственная вершина r. Поле π[r] ссылается на ближайшую к r вершину, вес до которой есть key[r] = w(r, π[r]). В цикле (строки 5–15) мы проверяем, лежат ли концы ребра в минимальном покрывающем дереве. Если да, то ребро нельзя добавить к минималь-

ному остовному дереву (не создавая цикла), и оно заменятся на ребро, обеспечивающее отсутствие циклов. Если нет, то ребро добавляется к A (строка 10). Таким образом, проверка условия в строке 7 обеспечивает сохранение инварианта (дерево есть часть некоторого остова). Заметим, что строки 8–9 обеспечивают обновление $\pi[u]$ и key[u], но не приводят к расширению минимального покрывающего дерева A. Каждое выполнение строки 10 соответствует одному шагу алгоритма Прима и увеличивает множество A на одно ребро.

Оценим время работы алгоритма. Цикл 5–15 не может повторяться менее чем $|V| - 1$ раз. Более точно: строки 10–15 выполняются ровно $|V| - 1$ раз, а количество повторений строк 8–9 в общем случае определить нельзя. Очевидно, что время работы функции Find-Nearest в среднем имеет сложность не менее $\log|V|$ (реализация такой функции будет показана ниже). Таким образом, асимптотическая оценка сложности алгоритма в лучшем случае не менее $|V| \log|V|$. Дать оценку для сложности алгоритма в общем случае не представляется возможным, поэтому она будет изучаться экспериментально.

Ключевыми моментами предложенного алгоритма является отложенное вычисление ближайшего соседа для точек, уже попавших в дерево, и эффективный поиск ближайшего соседа на раскрашенном в два цвета множестве. Будем считать, что исходные точки первоначально окрашены и перекрашиваются в другой цвет по мере построения остовного дерева (очередная добавляемая вершина перекрашивается). Таким образом, процедура Find-Nearest в алгоритме MST-KD должна искать ближайшего соседа для данной точки среди еще не закрашенных точек. Задача поиска ближайшего соседа является полностью статической, в том смысле, что на момент создания поисковой структуры данных, координаты всех компонент связности известны и зафиксированы: в процессе вычисления ближайших соседей не происходит удаления и добавления новых компонент связности, координаты существующих компонент не изменяются. Более того, процесс раскраски исходных точек все время сужает множество, на котором ищется ближайший сосед. Еще одной особенностью является поиск ближайшего соседа только для точек исходных данных. Исходя из всего этого, для ответа на запрос о ближайших соседях было выбрана модификация R-дерева [3] и модифицированный HS-алгоритм [7].

Покажем, как можно построить эффективную индексную структуру поиска. Для любого множества компонент связности M, существует пря-

моугольник I(M) со сторонами параллельными осям координат, который «накрывает» все компоненты связности этого множества. Накрывание состоит в следующем:

$$w(a, I(M)) \le w(a, c), \qquad \forall a, \forall c \in M,$$

где a, c — компоненты связности, w — выбранная нами функция расстояния.

Таким образом w(a, I(M)) ограничивает снизу расстояние от произвольной компоненты связности до компонент выбранного множества M, однако в отличии от HS-алгоритма эта нижняя граница не обязана достигаться. Этим фактом мы и воспользуемся для создания бинарного дерева поиска.

Сделаем некоторое обобщение. Пусть S — некоторое конечное множество, элементы которого мы называем точками данных, $w : S \times S \to R$ — некоторая функция расстояния, удовлетворяющая условиям:

$w(a, b) \ge 0$
$w(a, a) = 0$
$w(a, b) = w(b, a)$
$w(\bullet, \bullet) \ne const$

Пусть функцию расстояния можно обобщить на измерение расстояния между точками данных, и подмножествами S так, чтобы она удовлетворяла

$$\forall a \in S, \qquad \forall S_2 \subseteq S_1 \subseteq S \to w(a, S_1) \le w(a, S_2)$$

и для одноточечных множеств совпадала с исходной функцией расстояния

$$\forall a \in S, \qquad \forall S_1 = \{b\} \subseteq S \to w(a, S_1) = w(a, b).$$

Очевидно, что

$$w(a, S_1 \cup S_2) \le w(a, S_1), \qquad w(a, S_1 \cup S_2) \le w(a, S_2),$$

$$\forall a \in S_1 \subseteq S \to w(a, S_1) = 0.$$

Покажем, что если $\forall a \in S$, $\forall S_1 \subseteq S$ операция вычисления $w(a, S_1)$ имеет сложность порядка сложности w(a, b), то можно построить структуру данных, позволяющую искать ближайшего соседа для точки данных за время меньшее, чем линейное (в лучшем случае).

Построим сбалансированное бинарное дерево. Его корень соответствует всему множеству S. Каждая вершина соответствует подмножеству S_i (и содержит информацию, позволяющую быстро считать $w(a, S_i)$), причем для братьев соответствующие им подмножества не пересекаются. Пусть терминальная вершина дерева ссылается на точки исходных данных (но не более чем на L штук). Очевидно, что терминальные вершины разбивают все множество данных S на непересекающиеся подмножества. Будем хранить для каждой точки данных указатель на терминальную вершину, которой она принадлежит. Структура узлов дерева поддерживает как спуск, так и подъем по дереву (вершина ссылается на своих двух потомков и родителя). Скажем, что точка данных принадлежат вершине, если она сама или кто-то из ее потомков ссылается на эту точку. Очевидно, что некоторой внутренней вершине принадлежат все точки ее потомков. Далее на этой структуре применим HS-алгоритм, который начинает свою работу с терминальной вершины, а затем проводит поиск в глубину с подъемом после исследования очередного поддерева.

Заметим, что эту же структуру можно использовать для поиска ближайшего соседа для точки $q \notin S$, $q \in Q$, $S \subset Q$, если на Q определены функции расстояния, а поиск начинать с корня дерева.

Покажем, что такая структура эффективно реализуется для компонент связности и выбранной функции расстояния $w(a,b) = |dx| + |dy|$.

В качестве расстояния между компонентой связности c и множеством компонент связности M предлагается взять расстояние между компонентой связности и прямоугольником, накрывающим M (то есть минимальным охватывающим прямоугольником)

$$w(c, M) = w(c, I(M)).$$

В вершинах индексного дерева будем хранить МОП для точек данных, принадлежащих этой вершине. Например, как на рис. 2. Сплошные прямоугольники — компоненты связности, на которые ссылаются терминальные вершины дерева поиска. Короткой штрих пунктирной линией обозначены минимальные охватывающие прямоугольники для терминальных вершин дерева поиска. Длинной штрих пунктирной линией обозначены минимальные охватывающие прямоугольники внутренних узлов дерева.

Улучшить время поиска при использовании раскрашенных точек данных, позволяет поддержание в узлах дерева поиска информации о

количестве принадлежащих ему точек данных, еще не включенных в остовное дерево.

Заметим, что построенное дерево поиска может использоваться для эффективного поиска k-ближайших соседей.

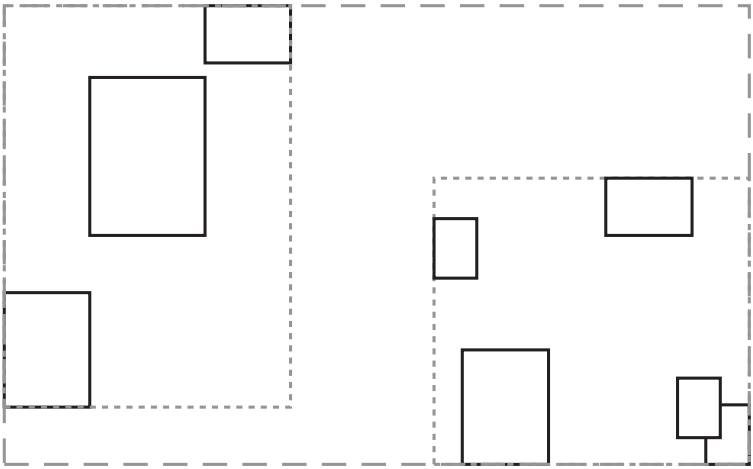

Рис. 2. Компоненты связности и МОП для двух терминальных вершин и их родителя

4. Программная реализация и экспериментальная проверка алгоритма

Задача построения минимального покрывающего дерева решалась согласно предложенному алгоритму, в котором процедура Find-Nearest является процедурой поиска ближайшего соседа на раскрашенном множестве вершин. Поскольку задача является статической в указанном выше смысле, сначала полностью строится дерево поиска, и только затем начинается построение минимального остова. Одним из параметров индексной структуры является значение L, ограничивающее количество точек данных в терминальной вершине.

Эффективность работы индексного дерева зависит от значения L. Исследование времени работы предложенного алгоритма показало, что единого оптимального значения нет, а оптимальное $L \in [4, 8]$ для разных распределений исходных данных. В этой работе при экспериментальной проверке использовалось $L = 6$.

Экспериментальная проверка характеристик предложенного алгоритма построения минимального покрывающего дерева проводилась на искусственных и реальных данных.

В качестве искусственных данных использовались регулярные возмущенные структуры.

Искусственный набор. Регулярная структура с возмущением

Рассмотрим регулярную решетку с косоугольной ячейкой. Расположим центры компонент в узлах решетки. С помощью датчика случайных чисел сгенерируем случайное отклонение центра компоненты от узла решетки (отклонение равномерное распределено на некотором интервале).

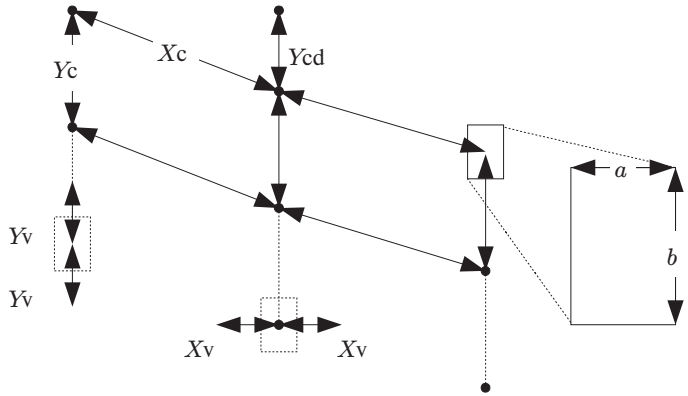

Рис. 3. Параметры регулярной структуры с возмущением.

a, b — размеры прямоугольников по оси OX и OY

Xc, Yc — расстояние между центрами прямоугольников по оси OX и OY

Xv, Yv — допустимое максимальное отклонение центра прямоугольника по оси OX и OY

Ycd — смещение центра прямоугольника по оси OY на одном шаге решетки по оси OX

Для эксперимента было сгенерировано несколько наборов исходных данных. Сравнительные характеристики искусственных наборов исходных

данных (с регулярной возмущенной структурой) для построения минимального покрывающего дерева приведем в таблице.

Таблица 1

Сводная таблица характеристики
искусственных наборов исходных данных
с регулярной возмущенной структурой

Название набора данных	Параметры						
	a	B	Xc	Yc	Ycd	Xv	Yv
Решетка	10	15	50	50	0	30	20
Зебра	10	15	50	150	0	15	110
Косая решетка 10	10	15	50	50	10	30	20
Косая решетка 50	10	15	50	50	50	30	20
Косая зебра 10	10	15	50	150	10	15	110
Косая зебра 50	10	15	50	150	50	15	110

Реальные данные

В качестве реальных данных брались компоненты связности, выделенные на изображениях отсканированных документов (от 400 до 12 000 компонент связности).

Результаты экспериментов

Для всех графиков по оси OX отложено количество компонент связности на графическом образе документа, а по оси OY — чистое (без подготовительных операций чтения исходных данных) время построения минимального остовного дерева, измеряемое в 1/1 000 секунды. На некоторых графиках используется логарифмическая шкала для времени, что указывается в подписи к графику.

Рассмотрим результаты эксперимента на реальных данных. Сравнивалось время построения минимального остовного дерева алгоритмом Прима (MST-Prim) и предложенным алгоритмом (MST-KD). На графике хорошо видна квадратичная зависимость для алгоритма Прима (в полном соответствии с теорией).

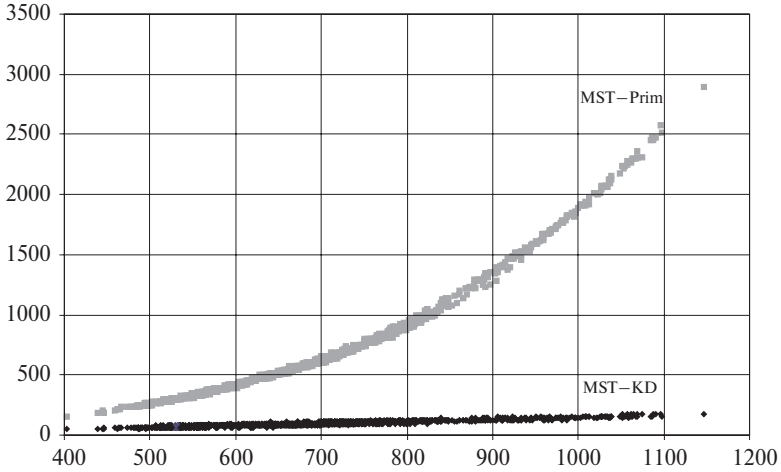

Рис. 4. Сравнительный график времени работы предложенного алгоритма и алгоритма Прима на реальных данных

Тот же график, но в логарифмической шкале времени.

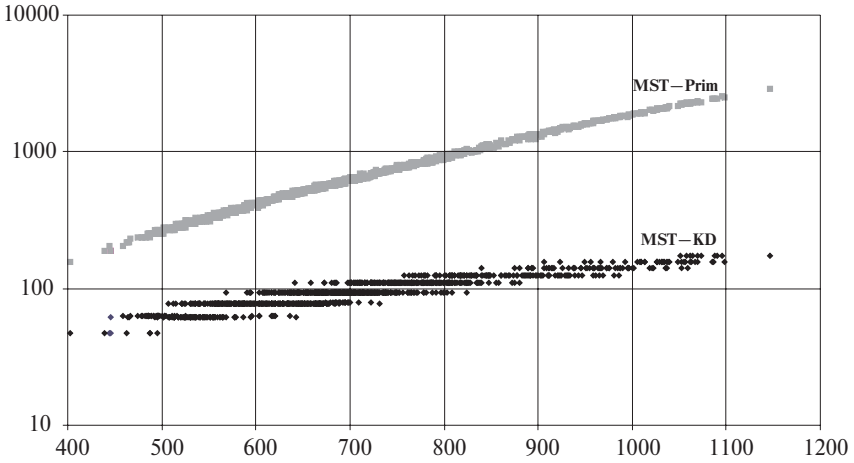

Рис. 5. Сравнительный график времени работы предложенного алгоритма и алгоритма Прима на реальных данных. Шкала времени логарифмическая

Для других наборов реальных и искусственных исходных данных получились аналогичные результаты: с ростом количества компонент связ-

ности время работы алгоритма Прима растет значительно быстрее времени работы предложенного алгоритма.

На всех искусственных наборах данных, исследуемый алгоритм показал схожие результаты. Теоретическое предположение, что отклонения от регулярной прямоугольной (параллельной осям) структуры данных приводят к увеличению времени работы, подтвердились практически (практически все точки графика, соответствующие набору «Косая зебра 50» лежат выше точек, соответствующих набору «Квадрат»). Из графика видно, что искусственные наборы данных не очень хорошо моделируют работу алгоритма и дают заниженную оценку времени работы.

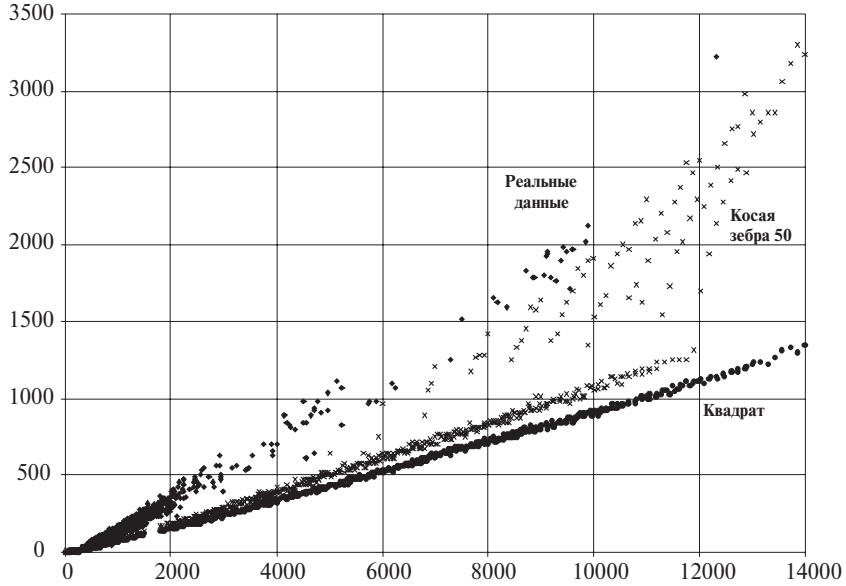

Рис. 6. Сравнительный график времени работы предложенного алгоритма на реальных и искусственных данных

5. Заключение

Лучшее время предложенного алгоритма построения минимального покрывающего дерева имеет асимптотику не лучше $|V| \log|V|$, а худшее — лучше прямого алгоритма Прима, причем преимущество предложенного

алгоритма растет с ростом объема исходных данных. Сформулировано общее требование к характеристикам исходных данных и функции расстояния, для которых возможна реализации эффективного алгоритма построения минимального остовного дерева.

Практические испытания программы показали, что алгоритм имеет хорошие временные характеристики по отношению к росту объема входных данных и годен для промышленного применения.

Результаты проведенных исследований апробированы и использованы в рамках системы массового ввода стандартных форм документов Cognitive Forms [1].

Литература

1. *Арлазаров В. В., Постников В. В., Шоломов Д. Л.* Cognitive Forms — система массового ввода структурированных документов / Сборник трудов ИСА РАН «Управление информационными потоками». 2002.
2. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: МЦНМО, 2001.
3. *Guttman A.* R-trees: a dynamic index structure for spatial searching // Proceedings of the ACM SIGMOD International Conference on Management of Data. 1984. V. 14. P. 47–57.
4. *White D. A.*, *Jain R.* Similarity indexing with SS-tree // Proceedings of the 24th International Conference on Very Large Data Bases, VLDB. 1998. P. 194–205.
5. *Comer D.* Ubiquitous B-tree // ACM Computing Surveys, 11(2): 121–137, June 1979.
6. *Dobkin D., Lipton R. J.* Multidimensional searching problems // SIAM Journal of Computing, 5(2): 181–186, 1976.
7. *Hjaltson G. R., Samet H.* Ranking in spatial databases // Lecture Notes in Computer Science, 951: 83–95, 1995.
8. *Friedman J. H., Bently J. L., Finkel R. A.* An algorithm for finding best matches in logarithmic expected time // ACM Transaction on Mathematical Software. 1977. V. 3. P. 209–226.
9. *Clarkson K.* A randomized algorithm for closest-point queries // SIAM Journal of Computing, 17: 830–847, 1988.
10. *Bennett K. P., Fayyad U., Geiger D.* Density-based indexing for nearest neighbor queries. Technical Report MSR-TR-98-58, Microsoft Research, 1998.
11. *Beckmann N., Kriegel H. P., Shneider R., Seeger B.* The R*-tree: An efficient and robust access method for points and rectangles // Proceedings of the ACM SIGMOD International Conference on Management of Data. 1990. P. 322–331.

12. *Katayama N., Satoh S.* The RS-tree: An index structure for high-dimensional nearest neighbor queries // Proceedings of the ACM SIGMOD International Conference on Management of Data. 1997. V. 26, 2. P. 369–380.
13. *Roussopolous N., Kelley S., Vincent F.* Nearest neighbor queries // Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data. 1995. P. 71–79.
14. *Sarnak N., Tarjan R. E.* Planar point location using persistent search trees // Commun. ACM, 26: 669–679, 1986.
15. *Tsaparas P.* Nearest Neigbor Search in Multidimensional Spaces. Depth Oral Report, June 10, 1999.
16. *Agarwal P. K., Edelsbrunner H., Schwarzkopf O., Welzl E.* Euclidean minimum spanning trees and bichromatic closest pairs. Proc. 6th ACM Symp. Comp. Geom., 1990. P. 189–201.
17. *Yianilos P. N.* Data structures and algorithms for nearest neighbor search in general metric space // Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA ‘93). 1993. P. 311–321.
18. *Berchtold S., Keim D. A., Kriegel H.-P.* The X-tree: An index structure for high-dimensional data // VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases. 1999. P. 28–39.