УДК 519.178
ББК 22.176

АЛГОРИТМ РЕШЕНИЯ ДИНАМИЧЕСКОЙ ЗАДАЧИ ПОИСКА КРАТЧАЙШИХ РАССТОЯНИЙ В ГРАФЕ

Тимеряев Т.В.[1], **Ураков А.Р.**[2]
(Уфимский государственный авиационный технический университет, Уфа)

В статье рассматривается полностью динамическая задача поиска кратчайших расстояний между всеми парами вершин неориентированного графа. Для рассматриваемой задачи предлагается метод решения, учитывающий все возможные изменения расстояний в графе с помощью процедур добавления и удаления ребра. Для процедуры удаления ребра предложен алгоритм, использующий понятие точек равноудаленных от инцидентных удаляемому ребру вершин, что позволяет существенно уменьшить время решения и объем используемой памяти в практических сценариях. Произведено сравнения предложенного метода решения с известными, показывающее уменьшение времени счета в среднем в 30 раз на исследованных разреженных графах размерности 10^4 *вершин.*

Ключевые слова: динамические кратчайшие расстояния, актуализация графа, коррекция графа, равноудаленные точки.

Введение

Задачи, связанные с поиском кратчайших путей (и расстояний, ассоциированных с этими путями), являются одними из самых исследованных в теории графов. В классической (*статической*) задаче поиска кратчайших путей между всеми вершинами графа (задача APSP) предполагается, что исследуемый граф —

[1] *Тимеряев Тимофей Валерьевич (timeryaev@yandex.ru).*
[2] *Ураков Айрат Ренатович, кандидат физико-математических наук, доцент (urakov@ufanet.ru).*

его структура и веса ребер (дуг) — не меняются с течением времени. В этом случае кратчайшие пути находятся в графе один раз и в дальнейшем используются без изменений. Для статической задачи разработано большое число изящных алгоритмов с различными областями эффективного применения, среди самых известных из которых алгоритмы Беллмана — Форда, Флойда — Уоршелла, Дейкстры и др. В качестве меры эффективности алгоритма, как правило, используют скорость нахождения решения.

Другой, менее исследованной является *динамическая* разновидность задачи. В динамической задаче предполагается, что подлежащий граф — его структура и веса ребер (дуг) — изменяются с течением времени, в связи с чем необходимо поддерживать вычисленные кратчайшие пути в актуальном состоянии с учетом происходящих изменений. Различают две разновидности динамической задачи: *полностью динамическая* — допустимо увеличение и уменьшение весов ребер, *частично динамическая* — допустимо либо увеличение, либо уменьшение весов ребер. Здесь, как и в случае со статической задачей, важна скорость решения, поэтому эффективное применение методов решения статической задачи — расчет кратчайших путей заново при изменении графа — сильно ограничено (размерностью графа, частотой изменений графа). Динамические задачи и методы их решения в разное время исследовались, например, в работах [1, 6, 5, 8].

В статье предлагается метод решения полностью динамической задачи поиска кратчайших расстояний между всеми вершинами ненаправленного графа с положительными вещественными весами ребер. Следует подчеркнуть, что при решении данной задачи нас интересует лишь актуальность кратчайших расстояний между вершинами, информация о соответствующих путях не актуализируется. Метод решения основан на использовании двух алгоритмов: удаления и добавления ребра, с помощью которых учитываются все возможные модификации графа. Практическая ценность изложенных в статье метода и алгоритмов заключается в том, что в сравнении с известными быстрыми [3] алгоритмами [7, 4] они позволяют решать задачу на разреженных графах быст-

рее и при этом не используют никакой дополнительной информации кроме расстояний, что позволяет существенно уменьшить сложность решения по памяти.

1. Термины и обозначения

В статье используются следующие термины и обозначения.

$G\,(V, E, w)$ — граф G с множеством вершин V, множеством ребер E и весовой функцией на ребрах w.

$e\,(a_0, b_0)$ — ребро между вершинами a_0 и b_0.

$w\,(a_0, b_0)$ — вес ребра между вершинами a_0 и b_0.

Расстояние между a_0 и b_0 — кратчайшее расстояние между вершинами a_0 и b_0.

$l\,(a_i, b_j)$ — расстояние между вершинами a_0 и b_0.

$[a_i, \ldots, b_j]$ — путь от вершины a_0 до вершины b_0.

Свойство оптимальности кратчайшего пути — каждый подпуть кратчайшего пути является также кратчайшим.

2. Метод коррекции графа

Изменение взвешенного графа состоит из операций следующего вида: изменение множества вершин, изменение множества ребер и изменение весов ребер. Требуемое изменение графа легко разложить на последовательность простейших операций, т. е. таких, которые затрагивают только одну вершину или ребро. Если мы будем отслеживать изменения расстояний при каждой простейшей операции, мы будем знать результирующие расстояния рассматриваемого графа.

Добавление/удаление вершины a, несвязанной с другими, изменяет расстояния в графе тривиальным образом — добавляются/удаляются равные бесконечности расстояния между a и всеми остальными вершинами графа. Операцию изменения списка вершин, связанных с другими вершинами графа, можно свести к операции изменения списка ребер.

Заметим, что если пара вершин a и b связана ребром веса w_1, то удаление или добавление ребра, связывающего a и b веса w_2,

не повлияет на расстояния, если $w_2 > w_1$. Отсюда следует, что уменьшение веса ребра можно свести к процедуре добавления ребра, а увеличение веса ребра можно свести к процедуре удаления ребра. В первом случае, сначала добавляется новое ребро меньшего веса, далее удаляется существующее ребро большего веса, что не повлияет на расстояния (и не требует пересчета). Во втором случае, сперва добавляется новое ребро большего веса, что не влияет на расстояния (и не требует пересчета), после этого удаляется существующее ребро меньшего веса.

Таким образом, рассматриваемая задача актуализации расстояний в динамическом графе может быть решена с помощью процедур удаления и добавления ребра. Далее дается описание алгоритмов этих процедур.

3. Добавление ребра

Рассматривается добавление ребра $e(a_0, b_0)$ веса $w(a_0, b_0)$ между вершинами a_0 и b_0. Необходимо пересчитать расстояния $l(a_i, b_j)$, которые в результате добавления данного ребра уменьшатся. Предполагается, что вес добавляемого ребра меньше текущего расстояния между вершинами $w(a_0, b_0) < l(a_0, b_0)$, т. к. в противном случае пересчет расстояний производить не нужно.

Добавляемое ребро $e(a_0, b_0)$ можно представлять как «мост», соединяющий множества вершин $A = \{a_i\}$ и $B = \{b_j\}$, которые разбивают множество вершин графа V на вершины, которые ближе к a_0, и вершины, которые ближе к b_0:

$$A = \{v | l(v, a_0) \leqslant l(v, b_0)\},$$
$$B = \{v | l(v, b_0) \leqslant l(v, a_0)\}. \quad (1)$$

В случае равенства расстояний до a_0 и до b_0 вершина v может быть помещена в любое из множеств A, B. Будем считать, что $a_0 \in A$ и $b_0 \in B$. При разбиении вершин графа (1), добавляя ребро $e(a_0, b_0)$, достаточно пересчитать расстояния лишь между парами вершин одна из которых принадлежит A, а другая B, т. е. $(v, u) : v \in A, u \in B$. Если обе вершины принадлежат одному из множеств, например, $v, u \in A$, то расстояние между ними не

изменится. Это обусловлено двумя факторами. Во-первых, добавление ребра $e\,(a_0, b_0)$ не изменит принадлежность вершин множеству A: расстояние от них до a_0 все так же будет не больше, чем до b_0. Действительно, если бы расстояние от, например, v до b_0 изменилось, соответствующий путь имел бы вид $[v, \ldots, a_0, b_0]$, что означало бы, что $l\,(v_i, a_0) < l\,(v_i, b_0)$. Во-вторых, кратчайший путь между v и u не может содержать добавляемое ребро $e\,(a_0, b_0)$, т. к. расстояние от обоих v и u до a_0 не больше, чем до b_0, и наличие ребра $e\,(a_0, b_0)$ в кратчайшем пути противоречило бы свойству оптимальности кратчайшего пути.

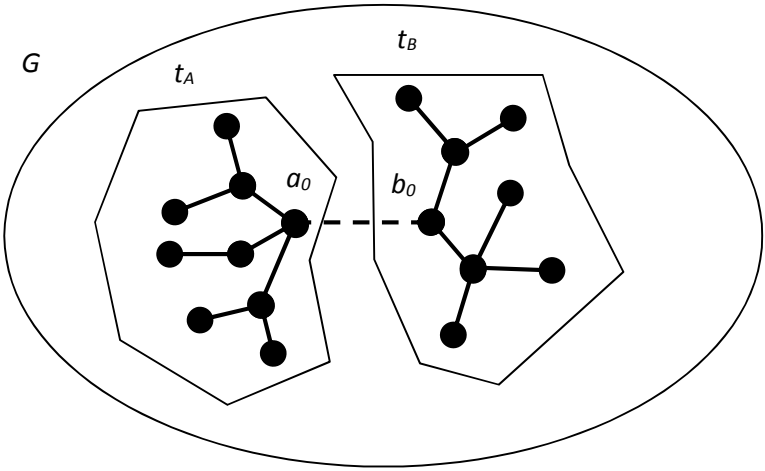

Рис. 1. Деревья t_A, t_B, между парами вершин которых при добавлении ребра $e\,(a_0, b_0)$ пересчитываются расстояния

Вершины графа можно поделить между множествами A и B так, что элементы этих множеств вместе со связывающими их ребрами в графе будут образовывать два связных подграфа. Действительно, это можно сделать, например, отнеся все вершины, которые имеют одинаковое расстояние до a_0 и b_0, либо к A, либо к B. Более того, в этом случае можно представить объединение вершин из множеств A и B в виде двух деревьев t_A и t_B с корнями a_0 и b_0, соответственно. Для каждого из множеств A и B (при

наличии достаточного числа вершин в них) существует несколько вариантов возможных деревьев с корнями a_0 и b_0. Среди этих вариантов существуют один обладающий полезным свойством.

В *дереве кратчайших путей* с корнем — некоторой вершиной v — длина пути от v до любой другой вершины дерева u равна расстоянию между этими вершинами $l(v,u)$. Если выбрать в качестве деревьев t_A и t_B на множествах A и B деревья кратчайших путей с корнями a_0 и b_0, соответственно, то, используя свойство этих деревьев, можно сократить число пар вершин (a_i, b_j), между которыми необходимо пересчитать расстояния.

Предположим, что вершина a_i дерева t_A в направлении от корня дерева связана ребрами с вершинам $a_{i+1}, a_{i+2}, \ldots, a_{i+k}$. Также предположим, что в данный момент времени мы пересчитываем расстояние между a_i и некоторой вершиной b_j дерева t_B. Расстояние между a_i и b_j необходимо пересчитать, если длина пути через добавляемое ребро $e(a_0, b_0)$ будет меньше текущего расстояния между a_i и b_j, т. е., если выполняется условие:

$$l(a_i, a_0) + w(a_0, b_0) + l(b_0, b_j) < l(a_i, b_j). \tag{2}$$

Если неравенство (2) не выполняется, величина $l(a_i, b_j)$ не пересчитывается, т. к. существует путь между a_i и b_j, не проходящий через $e(a_0, b_0)$ и обладающий не большей длиной. Но в этом случае не надо также пересчитывать расстояния между b_j и вершинами $a_{i+1}, a_{i+2}, \ldots, a_{i+k}$.

Действительно, т. к. дерево t_A является деревом кратчайших путей, то в случае пересчета расстояния между, например, a_{i+1} и b_j соответствующий кратчайший путь имел бы вид

$$[a_{i+1}, a_i, \ldots, a_0, b_0, \ldots, b_j]. \tag{3}$$

Если расстояние между a_i и b_j не было пересчитано, потому что выражение (2) имело форму равенства, то нет необходимости пересчитывать и расстояние между a_{i+1} и b_j, т. к. оно не изменится (не изменится длина подпути $[a_i, \ldots, b_j]$). Если же путь между a_i и b_j через $e(a_0, b_0)$ оказался длиннее, чем путь не проходящий через $e(a_0, b_0)$, то кратчайший путь между a_{i+1} и b_j не может иметь вида (3), т. к. в этом случае нарушалось бы свойство оптимальности кратчайшего пути.

Таким образом, если расстояние между данной a_i и некоторой b_j не было пересчитано, то не следует пересчитывать и расстояния между b_j и всеми вершинами исходящими из a_i в сторону от корня дерева t_A вплоть до листовых вершин. На основании этого правила можно уменьшить размеры деревьев t_A и t_B, исключая вершины дерева t_A, которым не следует пересчитывать расстояния до b_0 и вершины дерева t_B, которым не следует пересчитывать расстояния до a_0. Формально в t_A и t_B включаются, соответственно, вершины a_i и b_j, удовлетворяющие неравенствам

$$\begin{aligned} l(a_i, a_0) + w(a_0, b_0) < l(a_i, b_0), \\ l(b_j, b_0) + w(a_0, b_0) < l(b_j, a_0). \end{aligned} \tag{4}$$

Алгоритм 1 (Построение деревьев кратчайших путей)**.**

`ПостроитьДеревоДобавления` (G, l, v_1, v_2)

1. t = Дерево(Узел(v_1)) *//дерево с корнем с индексом v_1*
2. q = Очередь(t.Корень()) *//q — структура типа FIFO*
3. d = Множество(v_1) *//d — мн-во посещенных вершин*
4. **Пока** q.Размер() > 0
5. n = q.Выбрать()
6. v = n.Индекс()
7. **Для всех** ребер $e(v, u) \in G.V()$
8. **Если** d.Содержит(u) **то**
9. **Продолжить** *//continue*
10. **Если** $l(v_1, u) = l(v_1, v) + G.w(v, u)$ **то**
11. d.Добавить(u)
12. **Если** $l(u, v_1) + G.w(v_1, v_2) < l(u, v_2)$ **то**
13. c = Узел(u)
14. n.ДобавитьПотомка(c)
15. q.Добавить(c)
16. **Вернуть** t

Схематическое изображение деревьев t_A и t_B представлено на рис. 1. Описание алгоритма пересчета расстояний при добавлении ребра дано в листингах алгоритм 1, алгоритм 2. Алгоритмы в данных листингах предполагают, что текущие расстояния между всеми парами вершин графа до добавления ребра $e(a_0, b_0)$

известны.

Алгоритм 2 (Пересчет l при добавлении $e\,(a_0, b_0)$ веса w_{ab})**.**

ДобавитьРебро (G, l, a_0, b_0, w_{ab})
1. $G.E$.Добавить($e\,(a_0, b_0)$)
2. $G.w\,(a_0, b_0) = w_{ab}$
3. $G.w\,(b_0, a_0) = w_{ab}$
4. **Если** $l\,(a_0, b_0) \leqslant w_{ab}$ **то**
5. **Вернуть**
6. $t_A =$ ПостроитьДерево(G, l, a_0, b_0)
7. $t_B =$ ПостроитьДерево(G, l, b_0, a_0)
8. $q_a =$ Очередь(t_A.Корень())
9. **Пока** q_a.Размер() > 0
10. $n_a = q_a$.Выбрать()
11. $a_i = n_a$.Индекс()
12. $q_b =$ Очередь(t_B.Корень())
13. **Пока** q_b.Размер() > 0
14. $n_b = q_b$.Выбрать()
15. $b_j = n_b$.Индекс()
16. $l = l\,(a_i, a_0) + G.w\,(a_0, b_0) + l\,(b_0, b_j)$
17. **Если** $l < l\,(a_i, b_j)$ **то**
18. $l\,(a_i, b_j) = l$
19. $l\,(b_j, a_i) = l$
20. **Для** $k = 1$ **до** b_j.ЧислоПотомков()
21. q_b.Добавить(b_j.Потомок(k))
22. **Для** $k = 1$ **до** a_i.ЧислоПотомков()
23. q_a.Добавить(a_i.Потомок(k))

4. Алгоритм поиска кратчайших расстояний путем добавления ребер к остовному дереву

Алгоритм пересчета расстояний при добавлении ребра может использоваться для определения расстояний между всеми парами вершина графа (задача APSD). Для этого в графе выбирается произвольное остовное дерево t_f, для которого поиском в ширину находятся расстояния между всеми вершинами. После

чего по очереди добавляются $|E| - (|V| - 1)$ не вошедших в t_f ребер графа, и для каждого добавленного ребра выполняется пересчет расстояний.

Алгоритм 3 (Расчет расстояний для дерева)**.**

`ВычислитьРасстоянияДерева` (t_f, l)

1. $q =$ Очередь(t_f.Корень())
2. $d =$ Множество(t_f.Корень())
3. **Пока** q.Размер() > 0
4. $\quad n = q$.Выбрать()
5. $\quad v_j = n$.Индекс()
6. $\quad$ **Для** $m = 1$ **до** n.ЧислоПотомков()
7. $\quad\quad c = n$.Потомок(m)
8. $\quad\quad$ **Если не** d.Содержит(c) **то**
9. $\quad\quad\quad v_i = c$.Индекс()
10. $\quad\quad\quad$ **Для всех** $t \in d$
11. $\quad\quad\quad\quad v_k = t$.Индекс()
12. $\quad\quad\quad\quad l(v_k, v_i) = l(v_k, v_j) + w(v_j, v_i)$
13. $\quad\quad\quad\quad l(v_i, v_k) = l(v_k, v_i)$
14. $\quad\quad\quad d$.Добавить(c)
15. $\quad\quad\quad q$.Добавить(c)

При обходе дерева t_f расстояния от каждой новой посещаемой вершины v_i до уже посещенных v_k считаются через родительскую вершину v_j вершины v_i:

$$l(v_k, v_i) = l(v_k, v_j) + w(v_j, v_i).$$

В разделе 6 приведены результаты вычислительного эксперимента, в котором сравнивается время работы алгоритма добавления ребра и скорость выполнения известных алгоритмов поиска кратчайших расстояний. Согласно результатам, можно сделать вывод: использование приведенного алгоритма будет эффективно только при очень малом числе добавляемых ребер. Так как скорость добавления каждого ребра увеличивается пропорционально квадрату числа вершин, как и в других методах поиска кратчайших расстояний, то эффективность для широкого диапазона числа вершин будет ограничена некоторым абсолютным значением. Если

опираться на результаты в таб. 1, то можно сделать вывод, что алгоритм добавления ребер к дереву быстрее алгоритма Дейкстры, если добавляемых ребер не больше сотни, и быстрее алгоритма разборки-сборки, если добавляемых ребер не больше нескольких десятков.

Описание алгоритма определения расстояний между всеми парами вершин графа с использованием алгоритма пересчета расстояний при добавлении ребра приведено в листингах алгоритм 3, алгоритм 4.

Алгоритм 4 (Расчет расстояний между всеми вершинами)**.**
`ВычислитьРасстояния` (G, l)
1. $u =$ Очередь() *//содержит не посещенные ребра*
2. $t_f =$ ВыбратьДерево(G, u) *//u заполняется*
3. ВычислитьРасстоянияДерева (t_f, l)
4. **Пока** u.Размер() > 0
5. $\quad e = u$.Выбрать()
6. $\quad a_0 = e$.Вершина1()
7. $\quad b_0 = e$.Вершина2()
8. $\quad w_{ab} = e$.Вес()
9. $\quad$ ДобавитьРебро (G, l, a_0, b_0, w_{ab})

5. Удаление ребра и равноудаленные точки

Рассматривается удаление ребра $e\,(a_0, b_0)$ веса $w\,(a_0, b_0)$ между вершинами a_0 и b_0. Необходимо пересчитать расстояния $l\,(a_i, b_j)$, которые в результате удаления данного ребра увеличатся. Предполагается, что вес удаляемого ребра равен текущему расстоянию между вершинами $w\,(a_0, b_0) = l\,(a_0, b_0)$, т. к. в противном случае пересчет расстояний производить не нужно.

Как и в случае с добавлением ребра, используя выражения (1), множество вершин графа V можно поделить на множества A и B. На этих множествах так же можно построить деревья кратчайших путей t_A и t_B с корнями a_0 и b_0. Эти деревья будут обладать тем же полезным свойством: если расстояние между a_i и b_j не следует пересчитывать, то не следует пересчитывать и расстояния между всеми потомками a_i и b_j. Размеры деревьев

t_A и t_B можно уменьшить, но для этого вместо неравенств (4) следует использовать следующие равенства:

$$l(a_i, a_0) + w(a_0, b_0) = l(a_i, b_0), \quad l(b_j, b_0) + w(a_0, b_0) = l(b_j, a_0). \tag{5}$$

Равенства (5) оставляют в деревьях t_A и t_B только те вершины a_i и b_j, для которых существует кратчайший путь до, соответственно, вершин b_0 и a_0, проходящий через ребро $e(a_0, b_0)$. Так как в графе между любой парой вершин может существовать несколько кратчайший путей, не обязательно, что между вершинами деревьев t_A и t_B, полученных таким образом, произойдет хотя бы один пересчет расстояний.

Существенным отличием процедуры удаления ребра от случая добавления ребра является отсутствие информации о длине кратчайшего альтернативного пути. При добавлении ребра для пересчета расстояния между вершинами a_i и b_j используется неравенство (2), в котором длина известного (до добавления ребра $e(a_0, b_0)$) кратчайшего альтернативного пути $l(a_i, b_j)$ сравнивается с длиной пути через ребро $e(a_0, b_0)$. При удалении ребра известно лишь расстояние между a_i и b_j — длина кратчайшего пути (возможно проходящего через $e(a_0, b_0)$), второй по длине альтернативный путь между a_i и b_j неизвестен. Таким образом, возникает задача просмотра альтернативных путей и выбора из них наименьшего по длине.

На каждом из альтернативных путей между вершинами a_0 и b_0 существует точка, от которой расстояние до a_0 и до b_0 одинаково. Будем называть такие точки *равноудаленными точками* (РУТ). Используя РУТ можно обозначить все возможные варианты альтернативных путей между a_0 и b_0, ставя в соответствие каждому варианту отличную РУТ. Набор РУТ также определит и все возможные варианты альтернативных путей, не проходящих через ребро $e(a_0, b_0)$, между всеми парами вершин деревьев t_A и t_B. Действительно, пути между вершинами a_i и b_j через РУТ представляют собой пути, в которых вместо ребра $e(a_0, b_0)$ содержится подпуть через РУТ. Будем обозначать РУТ $C = \{c_k\}$, рис. 2 представляет схематическое изображение множества РУТ.

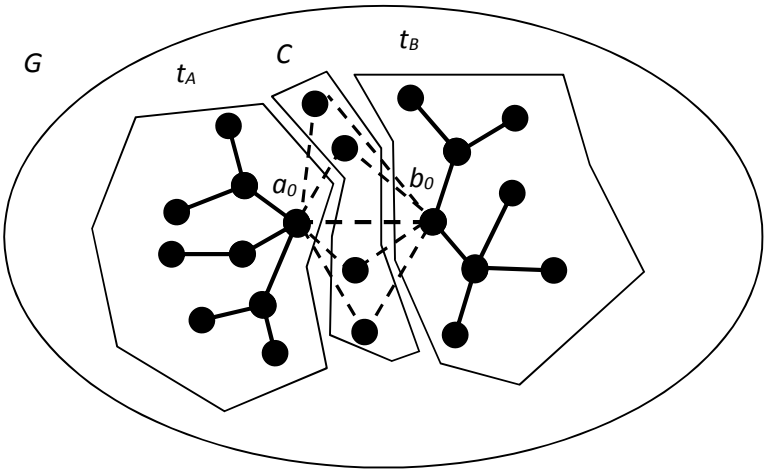

Рис. 2. Множество равноудаленных точек (РУТ) C при удалении ребра $e\,(a_0, b_0)$

Существует 2 варианта расположения РУТ. Во-первых, РУТ c_k, находящиеся в вершинах графа. Для таких справедливо равенство

$$l\,(c_k, a_0) = l\,(c_k, b_0)\,. \tag{6}$$

Во-вторых, РУТ, находящиеся на ребрах. Такие РУТ находятся на ребрах $e\,(v, u)$, удовлетворяющим всем трем неравенствам

$$\begin{gathered} |l\,(u, b_0) - l\,(v, a_0)| < w\,(v, u)\,, \\ l\,(v, a_0) < l\,(v, b_0)\,, \\ l\,(u, b_0) < l\,(u, a_0)\,. \end{gathered} \tag{7}$$

Расстояния между РУТ на ребрах c_k и некоторой вершиной z определяются формуле

$$l\,(z, c_k) = \begin{cases} l\,(z, v) + w'\,(v, c_k)\,, & \text{если } l\,(z, a_0) \leqslant l\,(z, b_0)\,; \\ l\,(z, u) + w'\,(u, c_k)\,, & \text{иначе.} \end{cases} \tag{8}$$

где $w'\,(v, c_k)$ и $w'\,(u, c_k)$ — длины отрезков ребра $e\,(v, u)$, определяемые по формулам

$$\begin{gathered} w'\,(v, c_k) = (l\,(u, b_0) + w\,(v, u) - l\,(v, a_0))\,/2, \\ w'\,(u, c_k) = w\,(v, u) - w'\,(v, c_k)\,. \end{gathered} \tag{9}$$

Так как РУТ имеют одинаковое расстояние до вершин, инцидентных удаляемому ребру, то при удалении ребра $e(a_0, b_0)$ расстояния от РУТ до всех остальных вершин графа не изменится. Действительно, если бы расстояние между РУТ c_k и некоторой вершиной z после удаления ребра $e(a_0, b_0)$ изменилось, то это значило бы, что до удаления ребра кратчайший путь между z и c_k имел вид $[c_k, \ldots, a_0, b_0, \ldots, z]$, что невозможно в силу свойства оптимальности кратчайшего пути и того, что ребро $e(a_0, b_0)$ имеет положительный вес. Таким образом, новое расстояние между вершинами a_i и b_j деревьев t_A и t_B может быть определено как минимальная сумма расстояний через РУТ: $l(a_i, b_j) = \min\limits_k (l(a_i, c_k) + l(c_k, b_j))$. Можно уменьшить число «холостых» проходов по деревьям t_A и t_B (без фактического пересчета расстояний), если использовать не расстояния до РУТ, а приращения (увеличения) расстояний через РУТ.

Будем обозначать $s(a_i, b_j)$ и называть *приращением расстояния* $l(a_i, b_j)$ увеличение расстояния $l(a_i, b_j)$ после удаления ребра $e(a_0, b_0)$. Тогда новое расстояние между a_i и b_j после удаления $e(a_0, b_0)$ определяется по формуле

$$l(a_i, b_j) = l(a_i, b_j) + s(a_i, b_j).$$

Будем обозначать $s^k(a_i, b_0)$, $s^k(b_j, a_0)$ и называть *приращениями расстояний* $l(a_i, b_0)$, $l(b_j, a_0)$ *через РУТ* c_k увеличение длин путей между соответствующими вершинами при прокладывании их через РУТ c_k при удалении ребра $e(a_0, b_0)$.

Для вершин, инцидентных удаляемому ребру, приращение определяется просто как минимальное увеличение пути через все существующие РУТ:

$$s(a_0, b_0) = \min_k s^k(a_0, b_0), \tag{10}$$

$$s^k(a_0, b_0) = l(a_0, c_k) + l(c_k, b_0) - w(a_0, b_0). \tag{11}$$

Для отличных от a_0,b_0 вершин деревьев t_A и t_B приращения через РУТ определяются как изменения приращений через РУТ от-

носительно родительского узла в дереве кратчайших путей:

(12)
$$s^k(a_{i+1}, b_0) = s^k(a_i, b_0) - w(a_{i+1}, a_i) + l(a_{i+1}, c_k) - l(a_i, c_k),$$
$$s^k(b_{j+1}, a_0) = s^k(b_j, a_0) - w(b_{j+1}, b_j) + l(b_{j+1}, c_k) - l(b_i, c_k),$$

где a_{i+1} и b_{j+1} — прямые потомки узлов a_i и b_j, соответственно, в деревьях t_A и t_B. Приращения расстояний для отличных от a_0, b_0 вершин a_i и b_j деревьев t_A и t_B определяются как сумма приращений через РУТ от a_i и от b_j с вычетом приращения через РУТ между вершинами удаляемого ребра, т. к. оно входит в сумму дважды:

(13)
$$s(a_i, b_j) = \min_k \left(s^k(a_i, b_0) + s^k(b_j, a_0) - s^k(a_0, b_0) \right).$$

Если для вершины a_i дерева t_A существует нулевое приращение расстояния через РУТ, т. е. $\min\limits_k s^k(a_i, b_0) = 0$, то для a_i и всех ее потомков в дереве t_A вплоть до листового уровня не нужно производить пересчет расстояний ни до одной вершины b_j дерева t_B. Действительно, в этом случае наличие нулевого приращения расстояния через РУТ означает существование альтернативного пути между a_i и b_0, не проходящего через удаляемое ребро $e(a_0, b_0)$, и имеющего такую же длину, как и путь через $e(a_0, b_0)$. Таким образом, расстояние $l(a_i, b_0)$ пересчитывать не нужно — оно не изменится. Так как t_A и t_B деревья кратчайших путей, кратчайший путь между любым a_x потомком a_i и любым b_y потомком b_0 содержит до удаления ребра $e(a_0, b_0)$ подпуть вида $[a_i, \ldots, b_0]$, т. к. длина $l(a_i, b_0)$ этого подпути не изменилась, то не изменится и расстояние $l(a_x, b_y)$. Аналогичное справедливо и для вершины b_j дерева t_B и ее потомков в случае $\min\limits_k s^k(b_j, a_0) = 0$.

Можно сократить и само число РУТ, используя два правила. $r1$) из C можно исключить те РУТ, находящиеся в вершинах, которые смежны только с вершинами, которые тоже являются РУТ в вершинах. В этом случае пути через исключаемые РУТ будут проходить также и через РУТ, которые будут не исключены. $r2$) если для деревьев t_A и t_B известны вершины a_f и b_f, находящиеся на наибольшем расстоянии от удаляемого ребра $e(a_0, b_0)$, т. е. $l(a_f, a_0) = \max\limits_i l(a_i, a_0)$ и $l(b_f, b_0) = \max\limits_j l(b_j, b_0)$, а также бли-

жайшая к удаляемому ребру РУТ c_n, т. е. $l(c_n, a_0) = \min\limits_i l(c_i, a_0)$, то нужно оставить в C только те РУТ c_k, для которых справедливо

$$l(c_k, a_0) < l(a_f, a_0) + l(b_f, b_0) + l(c_n, a_0).$$

Алгоритм 5 (Пересчет l при удалении $e(a_0, b_0)$ веса w_{ab})**.**

`УдалитьРебро` (G, l, a_0, b_0, w_{ab})

1. $G.E$.Удалить($e(a_0, b_0)$)
2. $G.w(a_0, b_0) = \infty$
3. $G.w(b_0, a_0) = \infty$
4. **Если** $l(a_0, b_0) < w_{ab}$ **то**
5. **Вернуть**
6. $t_A =$ ПостроитьДеревоУдаления(G, l, a_0, b_0)
7. $t_B =$ ПостроитьДеревоУдаления(G, l, b_0, a_0)
   //ПостроитьДеревоДобавления с (5) *вместо* (4)
8. $c =$ НайтиРУТ(G, l) *//формулы (6, 7), правила* $r1$, $r2$
9. $q_a =$ Очередь(t_A.Корень())
10. **Пока** q_a.Размер() > 0
11. $n_a = q_a$.Выбрать()
12. $a_i = n_a$.Индекс()
13. $s^k =$ НайтиПриращениеЧерезРут(G, l, c, a_i)
    //формулы (11, 12, 8, 9)
14. **Если** $s^k > 0$ **то**
15. $q_b =$ Очередь(t_B.Корень())
16. **Пока** q_b.Размер() > 0
17. $n_b = q_b$.Выбрать()
18. $b_j = n_b$.Индекс()
19. $s =$ НайтиПриращение(G, l, c, a_i, b_j)
    //формулы (10, 13)
20. **Если** $s > 0$ **то**
21. $l(a_i, b_j) = l(a_i, b_j) + s$
22. $l(b_j, a_i) = l(b_j, a_i) + s$
23. **Для** $k = 1$ **до** b_j.ЧислоПотомков()
24. q_b.Добавить(b_j.Потомок(k))
25. **Для** $k = 1$ **до** a_i.ЧислоПотомков()
26. q_a.Добавить(a_i.Потомок(k))

Описание алгоритма пересчета расстояний при удалении ребра приведено в листинге алгоритм 5.

6. Вычислительный эксперимент

Практическая эффективность предложенного метода актуализации расстояний в динамическом графе была оценена с помощью вычислительного эксперимента. В качестве тестовых данных использовались случайные связные подграфы заданной размерности графов автомобильных дорог европейских стран из проекта OpenStreetMap [10]. Эксперимент состоял в удалении случайного ребра, которое не нарушает связности графа, и последующем добавлении этого ребра обратно к графу, при этом фиксировалось время актуализации расстояний после каждой операции. Данная процедура повторялась 100 раз, после чего определялось среднее время актуализации расстояний каждым из сравниваемых алгоритмов. Исходный код программы был написан на языке C++, при компиляции использовался компилятор MS Visual Studio 2010 с флагом оптимизации O2.

Предложенные алгоритмы пересчета расстояний при добавлении ребра (уменьшении веса ребра) и удалении ребра (увеличение веса ребра) сравнивались с решением задачи статическими алгоритмами — полным пересчетом расстояний — n-кратным (по числу вершин графа) запуском алгоритма Дейкстры и алгоритмом разборки и сборки графа [2]. В эксперименте использовалась реализация алгоритма Дейкстры с двоичной кучей из библиотеки Boost Graph Library [11]. Полноценного сравнения с теоретически быстрейшими динамическими алгоритмами [7, 4] с использованием реализации, предоставленной авторами [9], произвести не удалось. Из-за большого потребления памяти данные алгоритмы не удалось использовать на графах размерности больше 10^3 вершин. На графах размерности 10^3 вершин лучший из данных алгоритмов показал в среднем в 10 раз большее время счета при потреблении около 150 МБ оперативной памяти (в сравнении с 6 МБ для предложенных в статье алгоритмов). Характеристики тестовых графов и результаты эксперимента представлены в табл.

1, 2.

Полученные результаты свидетельствуют о том, что предложенные алгоритмы позволяют существенно сократить время актуализации расстояний в сравнении с рассмотренными алгоритмами. С использованием предложенных алгоритмов актуализация расстояний при добавлении ребра происходит в среднем быстрее в 47 раза, а при удалении ребра — в 20 раз быстрее. Это позволяет, например, для исследованных графов с 10^4 вершин в среднем сэкономить 3 минуты вычислительного времени уже при 100 изменениях весов ребер графа.

Таблица 1. Характеристики тестовых графов

Группа графов (число графов)	Среднее число вершин	Средняя степень
G1(10)	10^3	2,19
G2(10)	$2 \cdot 10^3$	2,22
G3(10)	$3 \cdot 10^3$	2,23
G4(10)	$4 \cdot 10^3$	2,2
G5(10)	$5 \cdot 10^3$	2,21
G6(10)	$6 \cdot 10^3$	2,21
G7(10)	$7 \cdot 10^3$	2,23
G8(10)	$8 \cdot 10^3$	2,21
G9(10)	$9 \cdot 10^3$	2,22
G10(10)	10^4	2,2

7. Заключение

На практике взвешенные графы большого размера моделируют такие объекты (например, дорожные сети), для которых крайне маловероятно одновременное изменение весов у большого числа ребер. Скорее, наоборот — изменение ситуации (дорожной обстановки) должно сопровождаться постоянным и очень быстрым процессом пересчета расстояний между вершинами при

изменении весов отдельных ребер.

Приведенный динамический алгоритм коррекции кратчайших расстояний позволяет поддерживать актуальные расстояния между вершинами в режиме реального времени даже для графов большой размерности.

Литература

1. РОДИОНОВ В.В. *Параметрическая задача о кратчайших расстояниях* // Ж. вычисл. матем. и матем. физ. – 1968. – Т. 8, №. 5. – С. 1173-1177.
2. УРАКОВ А.Р. *Алгоритм поиска кратчайших путей для разреженных графов большой размерности* / А.Р. Ураков, Т.В. Тимеряев // Прикладная дискретная математика. – 2013. №. 1(19). – С. 84-92.
3. DEMETRESCU C. *Experimental analysis of dynamic all pairs shortest path algorithms* / C. Demetrescu, S. Emiliozzi, G.F. Italiano // Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms / Philadelphia : Society for Industrial and Applied Mathematics, 2004. – P. 362-371.
4. DEMETRESCU C. *A new approach to dynamic all pairs shortest paths* / C. Demetrescu, G.F. Italiano // Journal of the ACM. – 2004. –Vol. 51, № 6. – P. 968-992.
5. EVEN S. *Updating distances in dynamic graphs* / S. Even, H. Gazit // Methods of Operations Research. – 1985. № 49. – P. 371-387.
6. FRIGIONI D. *Fully dynamic algorithms for maintaining shortest paths trees* / D. Frigioni, A. Marchetti-Spaccamela, U. Nanni // Journal of Algorithms. – 2000. № 34. – P. 351-381.
7. RAMALINGAM G. *An incremental algorithm for a generalization of the shortest path problem* / G. Ramalingam, T. Reps // Journal of Algorithms. – 1996. № 21. – P. 267-305.
8. THORUP M. *Fully-Dynamic All-Pairs Shortest Paths: Faster and Allowing Negative Cycles* // Proceedings of the 9th

Scandinavian Workshop on Algorithm Theory / Berlin : Springer, 2004. – P. 384-396.
9. *Experimental Evaluation of Dynamic All Pairs Shortest Path Algorithms* [Электронный ресурс]. – Режим доступа: http://www.dis.uniroma1.it/ demetres/experim/dsp/
10. *OpenStreetMap* [Электронный ресурс]. – Режим доступа: http://wiki.openstreetmap.org/wiki/Planet.osm
11. *The Boost Graph Library* [Электронный ресурс]. – Режим доступа: www.boost.org/doc/libs/1_59_0/libs/graph/doc/index.html

ALGORITHM FOR DYNAMIC ALL-PAIRS SHORTEST DISTANCES IN GRAPH

Timeryaev Timofey, Ufa State Aviation Technical University, Ufa (timeryaev@yandex.ru).
Urakov Airat, Ufa State Aviation Technical University, Ufa, Candidate of Sciences, docent (urakov@ufanet.ru).

Abstract: Fully dynamic all-pairs shortest distances problem on undirected graphs is considered. A solution method taking into account all distance changes by the use of edge addition and deletion procedures is proposed. The developed algorithm for an edge deletion procedure uses the notion of points equidistant from the vertices incident to the edge being deleted what allows to significantly reduce time and memory complexity of solution in practical scenarios. The conducted computational experiment has shown reduction of solution time by an average of 45 times for sparse graphs with 10^4 *vertices compared to known solution methods.*

Keywords: dynamic shortest distances, graph update, equidistant points.

Статья представлена к публикации
членом редакционной коллегии ...

Поступила в редакцию ...
Дата опубликования ...

Таблица 2. Результаты вычислительного эксперимента

Группа графов	Добавление. Среднее время, с	Добавление. Макс. время, с	Удаление. Среднее время, с	Удаление. Макс. время, с	Среднее число РУТ	Макс. число РУТ	Дейкстра. Среднее время, с	Разборка –сборка. Среднее время, с
G1	$3,9 \cdot 10^{-4}$	0,016	0,001	0,016	4,94	29	0,058	0,012
G2	0,002	0,016	0,003	0,031	6,66	42	0,262	0,044
G3	0,004	0,047	0,008	0,078	9,77	61	0,618	0,121
G4	0,006	0,062	0,012	0,125	8,83	58	1,08	0,249
G5	0,006	0,062	0,015	0,234	11,5	54	1,82	0,378
G6	0,01	0,109	0,023	0,25	10,3	60	2,58	0,562
G7	0,014	0,187	0,037	0,452	16,3	78	3,75	0,793
G8	0,017	0,281	0,043	0,671	11,5	68	4,95	1,08
G9	0,025	0,218	0,065	0,64	15,2	102	6,37	1,41
G10	0,035	0,296	0,08	1,19	14,3	85	8,05	1,88