

УДК 519.17

ОБ АЛГОРИТМАХ ОПТИМАЛЬНОЙ РАЗМЕТКИ

М. Н. Алексеев^{1,a}, Ф. М. Алексеев^{2,b}

¹ Челябинский государственный университет, Челябинск, Россия

² Московский физико-технический институт (государственный университет), Долгопрудный, Россия

^a alexeev@csu.ru; ^b alekseev@phystech.edu

Задача о выборе оптимального порядка разметки рассмотрена как особый случай задачи коммивояжёра. Предложены алгоритмы точного и приближённого решений задачи. Обсуждены оценки сложности алгоритмов и возможные подходы к их улучшению.

Ключевые слова: задача коммивояжёра, неориентированный взвешенный граф, минимальное остовное дерево, гамильтонова цепь, динамическое программирование, жадный алгоритм, алгоритм Прима, алгоритм Краскала.

1. Постановка задачи

Рассмотрим работу автоматического устройства для нанесения разметки на плоскую металлическую поверхность ударным игольчатым механизмом. Прямоугольное рабочее поле разметки ограничено линейными размерами W и H . На этом поле устройство наносит ударной иглой последовательно несколько ($N \geq 1$) надписей, которые можно условно представить в виде односвязных областей с явно заданными координатами двух крайних точек $x_{i,1}, y_{i,1}$ и $x_{i,2}, y_{i,2}$ для i -й области ($1 \leq i \leq N$). Алгоритмы формирования составляющих надписи символов жёстко запрограммированы, время выполнения надписи зависит от её конфигурации и количества точек (следов ударов иглы), при этом надпись может быть за одно и то же время выполнена как в прямом — от 1-й точки до 2-й, так и в обратном порядке — от 2-й точки к 1-й. Время холостого перехода устройства от последней точки одной надписи к первой точке следующей надписи пропорционально расстоянию между этими точками. Для заданного набора надписей нужно выбрать последовательность их нанесения с минимальным холостым пробегом.

На самом деле специфика ударного механизма не является существенной. Точно так же можно рассматривать работу графопостроителя (плоттера) с одним пером.

Описанная задача похожа на классическую задачу коммивояжёра [1, с. 1138], которая в одной из распространённых интерпретаций заключается в отыскании самого выгодного маршрута, проходящего через все указанные города по одному разу, т. е. в поиске кратчайшей гамильтоновой цепи в неориентированном взвешенном графе. В общем случае задача относится к классу NP -полных, т. е. очень сложна и затратна по времени решения. Задача коммивояжёра неоднократно рассматривалась разными исследователями, для её точного или приближённого решения предлагаются различные методы — от метода полного перебора, предусматривающего рассмотрение всех $N!$ возможных маршрутов, до разнообразных эвристических подходов.

Отличие нашей задачи от классической задачи коммивояжёра в том, что, хотя граф полон, т. е. возможны переходы между любыми двумя точками, каждая точка v имеет ровно одну сопряжённую ей точку u , в том смысле, что за первым посещением точки v обязательно следует переход в её сопряжённую точку u . Из точки u затем можно снова уже произвольно выбрать следующую новую точку вместе с её сопряжённой. При этом отношение сопряжённости взаимно: из (v сопряжена u) следует (u сопряжена v). В дальнейшем сопряжённые точки будем также называть парными, а пары сопряжённых точек — просто парами. Нелишним будет пояснить, что пары сопряжённых точек и будут однозначно соответствовать концам отрезков, на которые нужно нанести надписи.

Первый и последний переходы маршрута — строго внутри пары точек. Стоит уточнить также, что для поиска оптимального варианта маршрута первой можно выбрать любую пару в любой ориентации.

Понятно, что каждый маршрут также имеет свой взаимно сопряжённый вариант, т. е. его можно с равной эффективностью проходить как от начала к концу, так и от конца к началу. При этом для любого маршрута сумма расстояний между точками из одной пары — чистое время выполнения надписей — является константой, не зависящей от порядка обхода, так что веса рёбер, связывающих сопряжённые точки, для выбора оптимального порядка нанесения надписей несущественны.

Ту же задачу можно сформулировать иными терминами теории графов. Пусть у нас есть максимальное паросочетание — набор пар точек, связанных «сильными» рёбрами (надписями), через которые нам следует достроить чередующуюся цепь, связав концы «сильных» рёбер «слабыми» рёбрами переходов вне надписей.

Можно также говорить о построении минимального остова графа с двумя ограничениями: все сильные рёбра должны присутствовать в остове, и ни для какой вершины её степень не должна превысить 2.

2. Точное решение

В контексте практической задачи разметки мы можем ограничиться разумным количеством надписей $N \leq 20$ и построить, например, следующий алгоритм точного решения задачи методом динамического программирования с экспоненциально растущим потреблением времени и памяти.

2.1. Теоретическое обоснование алгоритма

Обозначим за $d_{\text{mask}, \text{last}, \text{end}}$ минимальное суммарное расстояние холостых переходов, с которым можно разметить все надписи, входящие во множество mask , при этом последней разметив надпись номер last по направлению к концу end . Например, значение $d_{\{1,5,7\}, 5, 1}$ соответствует минимальному суммарному расстоянию холостых переходов, с которым возможно разметить надписи 1, 5 и 7, закончив надписью номер 5 в направлении от 2-й точки к 1-й. Это значит в том числе и то, что в рассмотренной ситуации игла закончит разметку в точке $(x_{5,1}, y_{5,1})$. Тогда верны следующие соображения:

1. Для любого номера надписи i верно

$$d_{\{i\}, i, 1} = d_{\{i\}, i, 2} = 0. \quad (1)$$

Действительно, единственную надпись можно нанести без лишних затрат по времени, если заранее установить иглу в один из концов надписи. В терминах динамического программирования это утверждение следует считать базой динамики.

2. Заметим, что для разметки некоторого множества надписей с фиксированной последней надписью и с фиксированным направлением нанесения последней надписи необходимо и достаточно разметить все надписи, кроме последней, затем вхолостую перенести иглу в её начало и, наконец, нанести последнюю надпись. Нужно только выбрать тот оптимальный способ разметки меньшего множества с известной последней надписью, с которым и исходное множество будет размечено оптимально.

Формально говоря, для всякого множества надписей mask величины не менее двух, принадлежащего ему номера надписи last и номера конца end значение $d_{\text{mask},\text{last},\text{end}}$ равно

$$\min_{\text{prev} \in \text{mask} \setminus \{\text{last}\}} \text{try}(\text{prev}), \quad (2)$$

где

$$\begin{aligned} \text{try}(\text{prev}) = & d_{\text{mask} \setminus \{\text{last}\}, \text{prev}, \text{prev_end}} + \\ & + \rho \left((x_{\text{prev}, \text{prev_end}}, y_{\text{prev}, \text{prev_end}}), (x_{\text{last}, 3-\text{end}}, y_{\text{last}, 3-\text{end}}) \right), \end{aligned}$$

где $\rho(A, B)$ — длина отрезка, соединяющего точки A и B . Это переход в динамике.

3. Ответ задачи — суммарное «штрафное время», равное минимуму среди всех значений $d_{\text{all},\text{last},\text{end}}$, где all — множество всех надписей, $1 \leq \text{last} \leq N$, $1 \leq \text{end} \leq 2$, то есть мы выбираем наиболее оптимальный из всех способов разметить все надписи, причём нам не важно, какую надпись мы нанесём последней и в каком направлении мы это сделаем.

2.2. Реализация алгоритма

Заведём трёхмерный массив \mathbf{d} , в котором и будем хранить все посчитанные значения функции d .

Для индексации в нём множество размеченных надписей будем представлять в виде битовой маски, т. е. числа, содержащего в своей двоичной записи в i -м разряде 1, если i -я надпись присутствует в множестве, и 0 в противном случае. Тогда множество $\{1, 5, 7\}$ будет представлено в виде числа $1010001_2 = 16110$. Множество всех надписей all будет представлено в виде числа, двоичная запись которого состоит из N единиц, заполняющих N младших разрядов, т. е. $2^N - 1$.

Будем заполнять массив \mathbf{d} , постепенно увеличивая битовую маску размеченных надписей от 1 до $2^N - 1$ включительно. Такой порядок гарантирует, что для всех подмножеств текущего множества размеченных надписей значения функции уже вычислены и хранятся в массиве \mathbf{d} , так как битовые маски подмножеств никогда не превысят битовой маски исходного множества. Итак, зафиксировав текущее множество размеченных надписей mask , переберём последнюю надпись last и направление end , чтобы найти все $d_{\text{mask},\text{last},\text{end}}$. Если mask состоит из одного элемента, воспользуемся равенствами (1) и присвоим соответствующему элементу массива значение 0. Иначе воспользуемся формулой (2), перебрав все возможные значения prev и prev_end .

Таким образом мы заполнили массив \mathbf{d} и можем восстановить «штрафное время» оптимального ответа, воспользовавшись соображением 3. Осталось восстановить сам ответ — перестановку номеров надписей с пометками, в каком направлении какую надпись размечать. Заметим, что, следуя соображению 3, мы найдём не только штраф, но и номер best_last последней надписи, и направление best_end её разметки в оптимальном алгоритме.

Воспользуемся следующим стандартным для динамического программирования приёмом. Пользуясь переходом динамики, по параметрам задачи $\langle \text{mask}, \text{last}, \text{end} \rangle$ мы можем не только найти $d_{\text{mask}, \text{last}, \text{end}}$, но и запомнить те значения переменных prev и prev_end , при которых функция try достигает минимума. Обозначим эти значения за result и result_end соответственно. Найденные значения — не что иное, как номер предпоследней надписи и направление её разметки в оптимальном алгоритме разметки множества mask с последней надписью last , размеченной в направлении к концу end .

Таким образом, зная best_last и best_end — последнее действие алгоритма, мы можем повторно применить соображение 2 и получить предпоследнее действие алгоритма. Затем, зная его, получить действие номер $N - 2$ и так далее, вплоть до самого первого действия.

Опишем алгоритм восстановления ответа. Будем хранить переменные **mask**, **last** и **end**, изначально равные соответственно $2^N - 1$, best_last и best_end , а также список номеров надписей **perm** и список пометок о направлении **directions**, в которых будет восстановлен ответ.

Изначально они оба будут состоять из одного элемента, это best_last для **perm** и best_end для **directions**.

Теперь до тех пор, пока в **mask** не останется единственный элемент, будем делать следующее: с помощью перехода динамики найдём result и result_end . Присвоим переменным **mask**, **last** и **end** значения $\text{mask} \oplus 2^{\text{last}-1}$, result и result_end соответственно. Здесь знаком \oplus обозначена операция побитового исключающего ИЛИ, т. е. в данном случае это обнуление бита **mask**, стоящего в разряде **last**, если пронумеровать разряды с единицы. Запишем новые значения **last** и **end** в начала списков **perm** и **directions** соответственно. Перейдём к следующей итерации.

2.3. Оценка сложности предложенного точного решения

Оценим число состояний динамики, т. е. количество возможных комбинаций параметров функции d . Из комбинаторики известно, что у всякого множества из N элементов существует ровно $2^N - 1$ различных непустых подмножеств. Значит, и вариантов параметра mask существует всего $2^N - 1$, т. е. $O(2^N)$. Для каждого варианта mask имеется в среднем $N/2$ вариантов last (это следует из того факта, что у множества из N элементов имеется C_N^K подмножеств, а $C_N^K = C_N^{N-K}$). Наконец, для каждой пары $\langle \text{mask}, \text{last} \rangle$ существует ровно 2 (т. е. $O(1)$) варианта end .

Итого имеем оценку числа состояний $O(2^N) \cdot O(N) \cdot O(1) = O(2^N \cdot N)$.

Пользуясь приведёнными рассуждениями, можно сказать, что трёхмерный массив **d** должен иметь размер порядка числа состояний, например $2^N \times N \times 2$ (для удобства индексации мы заведём примерно в два раза больше ячеек памяти, чем нужно, но эта жертва не повлияет на асимптотику требований алгоритма к памяти). Восстановление ответа потребует всего $O(N)$ памяти, поэтому алгоритм в целом требует порядка $O(2^N \cdot N)$ памяти.

Оценим теперь временную сложность работы. Заметим, что все состояния поделились на два типа: $O(N)$ состояний, составляющих базу, значение каждого из которых было получено за $O(1)$ с помощью соображения (1), и $O(2^N \cdot N)$ оставшихся, значение каждого из которых было получено за $O(N)$ с помощью перехода динамики (2). Итого массив **d** будет заполнен за $O(N) \cdot O(1) + O(2^N \cdot N) \cdot O(N) = O(2^N \cdot N^2)$ операций.

Восстановление ответа будет выполнено за $O(N)$ операций перехода, а значит, его сложность составит всего $O(N^2)$ операций.

Итоговая сложность алгоритма имеет оценку $O(2^N \cdot N^2)$.

3. Приближённое «жадное» решение на основе алгоритма Прима

Вернёмся к исходной постановке задачи. Идея решения состоит в том, чтобы искать гамильтонов путь, или остовную цепь, итеративно, начав с одного ребра, которое обязано входить в ответ, и постепенно наращивая текущую цепь, добавляя ребро в вершину, ближайшую к одному из концов цепи. Изложим этот подход более подробно.

Заведём дэк (deque), в который будем по мере построения остовой цепи помещать добавленные вершины. В начале построения в дэке лежит пара точек — концы одной произвольно выбранной надписи. Далее на каждой итерации из всех ещё не добавленных в дэк точек будем выбирать ближайшую к текущему началу либо к концу цепи. Выбранную точку добавим в дэк с соответствующей стороны.

После этого добавим с той же стороны и её пару (она ещё не была добавлена в дэк, иначе по алгоритму уже была бы добавлена и выбранная). Будем повторять итерации, пока все точки не будут добавлены в дэк.

Нетрудно заметить, что описанный алгоритм напоминает алгоритм Прима построения остовного дерева с двумя важными модификациями:

- вслед за одной точкой в дэк непременно добавляется и её пара;
- новые точки выбираются не по принципу «ближайшая к вершинам текущего остова», а по принципу «ближайшая к одному из концов текущей остовой цепи».

Эти модификации позволяют найти именно остовную цепь, а не минимальное остовное дерево, пожертвовав, впрочем, гарантией оптимальности.

На каждой итерации мы увеличиваем дэк, описывающий искомую остовную цепь, на два. Поэтому число итераций равно половине количества точек, т. е. количеству надписей.

На каждой итерации поиск ближайшей к концу цепи точки производится также за линейное время, поэтому при зафиксированной первой надписи алгоритм выполняет $O(N^2)$ операций.

На случайных графах перезапуск алгоритма от другой стартовой надписи нередко улучшает ответ. Можно перебрать все варианты стартовой надписи, тогда сложность возрастёт до $O(N^3)$, зато оптимальность ответа на случайном графе также улучшится.

4. Приближённое «жадное» решение на основе алгоритма Краскала

Если рассмотренный выше подход основан на алгоритме Прима, следующее решение повторяет классический алгоритм Краскала построения минимального остовного дерева также лишь с некоторыми отличиями:

- если обычно в начале алгоритма мы имеем набор вершин без рёбер, который в процессе добавления новых рёбер в порядке увеличения веса преобразуется в дерево, то здесь мы уже до начала добавления слабых рёбер имеем совершенное паросочетание из сильных рёбер (соединяющих концы одной надписи);
- новое ребро не может быть добавлено, если оно инцидентно хотя бы одной вершине, чья степень в построенном к текущему моменту остовном лесу уже равна двум.

Легко видеть, что эти отличия преодолимы без увеличения сложности алгоритма, а значит, при эффективной реализации системы непересекающихся множеств асимптотика этого подхода равна сложности предварительной сортировки списка рёбер, что составляет $O(N^2 \ln N)$.

Таким образом, мы привели два примера алгоритма, работающего за время, полиномиально зависящее от размера задачи, находящего некоторое «неплохое» решение.

На рис. 1–3 представлены примеры расчётов на основе описанных алгоритмов.

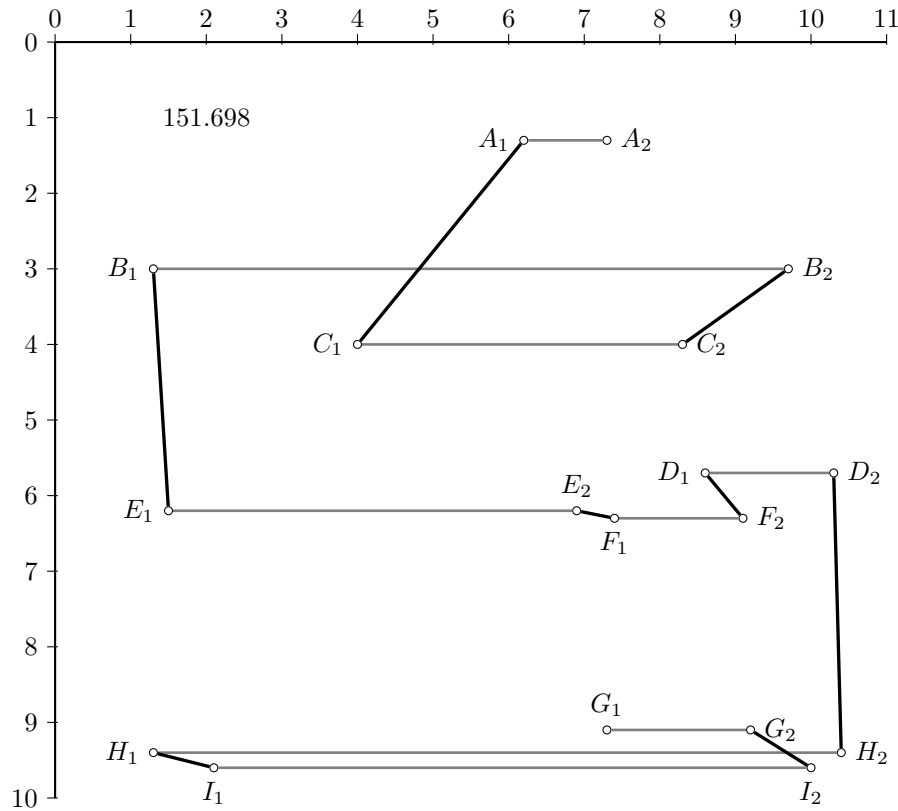


Рис. 1. Точное решение. Жирными линиями отмечены холостые переходы, в левом верхнем углу подписана их суммарная длина

5. Планы дальнейшего изучения задачи

- Исследование применительно к данной задаче предложенных Кристофидесом алгоритмов приближённого решения задачи коммивояжёра, дающих 2- и 1.5-приближение [2].
- Применение к данной задаче методов кластеризации.
- Изучение оптимальности выдаваемых предложенными алгоритмами ответов в случае специфических входных данных.

Список литературы

1. Алгоритмы: построение и анализ / Т. Кормен [и др.]. — 2-е изд. — М. : Вильямс, 2006. — 1296 с.
2. Кристофидес, Н. Теория графов: алгоритмический подход / Н. Кристофидес. — М. : Мир, 1978. — 432 с.

Поступила в редакцию 18.02.2015

После переработки 02.02.2016

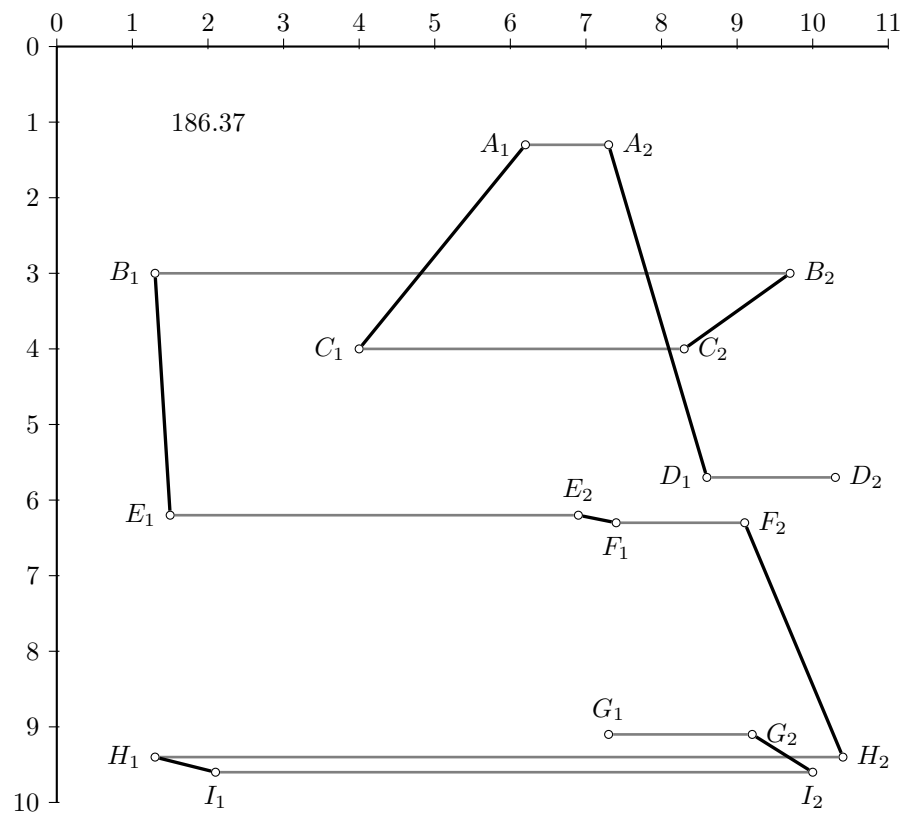


Рис. 2. Решение, основанное на модификации алгоритма Прима

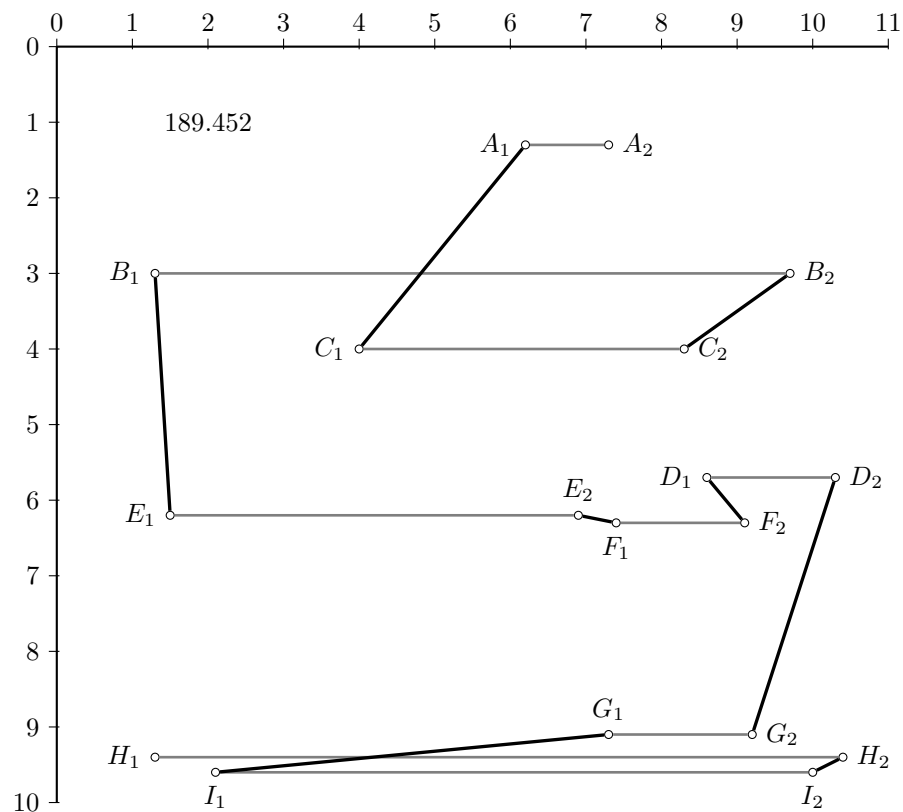


Рис. 3. Решение, основанное на модификации алгоритма Краскала

Сведения об авторах

Алексеев Михаил Николаевич, кандидат педагогических наук, доцент кафедры вычислительной механики и информационных технологий, Челябинский государственный университет, Челябинск, Россия; e-mail: alexeev@csu.ru.

Алексеев Фёдор Михайлович, студент факультета инноваций и высоких технологий, Московский физико-технический институт (государственный университет), Долгопрудный, Россия; e-mail: alekseev@phystech.edu.

Chelyabinsk Physical and Mathematical Journal. 2016. Vol. 1, iss. 1. P. 16–23.

ON OPTIMAL MARKING ALGORITHMS

M. N. Alekseev^{1,a}, F. M. Alekseev^{2,b}

¹*Chelyabinsk State University, Chelyabinsk, Russia*

²*Moscow Institute of Physics and Technology, Dolgoprudnyj, Russia*

^a*alexeev@csu.ru*; ^b*alekseev@phystech.edu*

The problem of choosing of the marking optimal order is considered as a special case of the traveling salesman problem. The algorithms for exact and approximate solutions search for the problem are proposed. The estimates of the algorithms complexity and possible approaches to their improvement are discussed.

Keywords: *the traveling salesman problem, undirected weighted graph, minimum spanning tree, Hamiltonian path, dynamic programming, greedy algorithm, Prim's algorithm, Kruskal's algorithm.*

References

1. **Cormen T.H. et al.** *Introduction to Algorithms*. 2nd Edition. MIT Press and McGraw — Hill, 2001.
2. **Kristofides, N.** *Teoriya grafov: algoritmicheskiy podkhod* [Graph theory: algorithmic approach]. Moscow, Mir Publ., 1978. 432 p. (In Russ.).

Article received 18.02.2015

Corrections received 02.02.2016