УДК 004.057.4

СТРУКТУРА СО СВОЙСТВАМИ ТЕСНОГО МИРА ДЛЯ РЕШЕНИЯ ЗАДАЧИ ПОИСКА БЛИЖАЙШЕГО СОСЕДА В МЕТРИЧЕСКОМ ПРОСТРАНСТВЕ

© 2012 г. ***А.А. Пономаренко, Ю.А. Мальков, А.А. Логвинов, В.В. Крылов***

ООО «МераЛабс», Нижний Новгород

aponom@meralabs.com

Поступила в редакцию 10.09.2012

Рассматривается новый подход к решению задачи поиска ближайшего соседа в метрическом пространстве. Предлагается в качестве структуры данных использовать граф, обладающий навигационными свойствами тесного мира, а в качестве алгоритма поиска – алгоритм градиентного спуска. Проблема существования локальных минимумов решается за счет произведения серии независимых поисков. Приведены экспериментальные данные, подтверждающие логарифмическую сложность алгоритма поиска.

Ключевые слова: поиск в метрическом пространстве, структура данных, тесный мир, вероятностный поиск, распределенные системы.

Введение

В этой работе рассматривается новый подход к решению задачи поиска ближайшего соседа в метрическом пространстве. Эта задача возникает, когда среди некоторого заданного множества объектов X необходимо отыскать некоторый объект $p \in X$, являющийся ближайшим к заданному объекту q (запросу)**.** Близость двух объектов o' и o'' оценивается исходя из заданной функции расстояния $d(o', o'')$, определенной на множестве всевозможных объектов D. Требуется так организовать конечное множество объектов $X \subset D$ в структуру $\boldsymbol{S}$, чтобы минимизировать количество вычислений функции d при поиске ближайшего объекта к q. Задача поиска ближайшего соседа или поиска наиболее схожего объекта к заданному образцу возникает в множестве приложений, таких как распознавание образов [1], компьютерное зрение, поиск изображений и видео по содержанию [2], машинное обучение [3], системы рекомендаций [4], кластерный анализ, поиск схожих последовательностей ДНК [5], семантический поиск документов [6].

В качестве самой тривиальной структуры $\boldsymbol{S}$ может служить обыкновенный линейный список. Сложность добавления в него новых элементов $O(1)$, но для того чтобы определить, какой элемент является ближайшим к q, требуется вычислить расстояния до каждого элемента из множества X, что эквивалентно сложности поиска $O(n)$, где n – количество элементов множества X.

Общим принципом уменьшения количества вычислений метрики является разбиение пространства на множество классов эквивалентности на этапе построении структуры. Тогда во время исполнения запроса часть классов эквивалентности отбрасывается, другая часть перебирается полным перебором. Описываемые методы различаются способом разбиения пространства на множество классов эквивалентности. Как было показано в [7], превалирующее большинство алгоритмов основываются на выборе k опорных точек и построении отображения исходного пространства в пространство R^k, используя L_∞-метрику.

Другой класс алгоритмов использует компактное разбиение.

Методы, основанные на компактном разбиении, более эффективны для пространств высокой размерности. Несмотря на это, почти все методы из обоих классов в качестве структуры данных используют деревья (GNAT, GHT, SAT, BST, VT, MT и другие), реже матрицы (ALAESA, LAESA и др.).

Более современным подходом для решения задачи поиска ближайшего соседа является построение структур данных в виде сетей. В качестве структуры $\boldsymbol{S}$ строится сеть, описываемая графом $G(V, E)$, где объектам из множества X однозначно сопоставлены вершины из множества V. Тогда поиск ближайшего элемента из множества X к запросу q будет сводиться к поиску вершины на графе $G(V, E)$. Использование данного подхода мотивировано следующим:

– существуют алгоритмы построения сети,

позволяющие производить поиск ближайшего соседа с логарифмической сложностью от количества элементов в структуре [8];

– локальность структуры позволяет производить добавление новых объектов с приемлемой сложностью и не требует перестроения всей структуры;

– структуры, имеющие топологию тесного мира, являются исходно распределенными, что позволяет естественным образом накладывать их на физические компьютерные сети.

Это дает возможность строить распределенные решения, которые обладают возможностью предоставления одновременного доступа к хранилищу большого числа пользователей (осуществления ими поиска и добавления новых данных), иметь хорошую отказоустойчивость, масштабируемость по производительности и вместимости.

Одним из самых простых алгоритмов поиска вершины в графе является «жадный поиск» (метод градиентного спуска). Этот алгоритм легко реализуем и в структуре с топологией сети и может быть инициирован от любой вершины.

Для того чтобы результатом работы алгоритма являлся точный ближайший элемент к запросу, необходимо, чтобы граф содержал в себе подграф Делоне (об этом подробнее в разделе «Алгоритм поиска»), который является двойственным к разбиению пространства Вороного [9].

Однако в приложениях, описанных выше, требование нахождения точного ближайшего соседа может быть избыточным. Это дает возможность для формулировки задачи приближенного поиска ближайшего соседа, что освобождает от необходимости иметь точный граф Делоне.

Для того чтобы алгоритм поиска масштабировался логарифмически, достаточно наличия в сети тесного мира со свойствами навигации, что уже хорошо изучено [8].

В данной работе мы представляем алгоритм построения структуры в виде сети с графом $G(V, E)$, эффективно решающей задачу приближенного поиска ближайшего соседа, основанный на алгоритме «жадного поиска». При этом граф $G(V, E)$ содержит в себе аппроксимированный граф Делоне и обладает навигационными свойствами тесного мира. В алгоритме поиска заложена возможность варьирования точности поиска без необходимости изменений в структуре. Алгоритм не использует координатные представления и не требует привлечения свойств линейных пространств, а основан только на вычислении метрики между объектами и потому применим для данных из абстрактных метрических пространств.

Существующие решения

Одними из первых структур для решения задачи точного поиска ближайшего соседа были Kd-tree [12] и quadra trees [13]. Они эффективны для метрических пространств малой размерности (сложность поиска приближается к $O(\log n)$). Однако анализ худшего случая, например для Kd-tree и quadra trees [14], для обеих структур говорит о сложности поиска $O(kn^{1-1/k})$, где k – размерность пространства.

Другие структуры древесного типа, такие как модифицированные варианты Kd-деревьев, R-деревья и структуры, основанные на кривых, заполняющих пространство, рассмотрены в книге [15]. Они также эффективны для поиска в метрическом пространстве малой размерности (сложность поиска приближается к $O(\log n)$ для $d < 4$), однако быстро теряют свою эффективность с ростом размерности пространства [16]. Более эффективная структура для точного поиска ближайшего соседа в R^k со сложностью $O(2^k \log n)$ впервые была описана в [17]. Но как видно, сложность поиска также имеет экспоненциальную зависимость от размерности пространства.

Используя эвристический метод «vantage point», были предложены такие структуры, как mvp-tree [18], vp^s-tree и vp^{sb}-tree [19], но при этом нет никакого анализа их эффективности для пространств высокой размерности.

В целом, на сегодняшний день не существует методов для эффективного точного поиска ближайшего соседа для пространств с большой размерностью, – причина этого лежит в «проклятье» размерности [7].

Чтобы избежать проклятья размерности, но при этом сохранить логарифмическую масштабируемость по количеству элементов, было предложено уменьшить требования к поиску ближайшего соседа, сделав его неточным. Так появилась масса работ, предлагающих отыскивать ближайшего соседа с точностью до ε (ε-NNS). Например, Arya и Mount предложили алгоритм со сложностью поиска $O(\log^3 n)$, но построение структуры требовало $O(n^2)$ времени и алгоритм применим только для данных из R^k [20].

Kleinberg предложил два метода [21]. Первый со сложностью $O(n \log d^{2k})$ для построения структуры и сложностью поиска с полиномиальной зависимостью от k, ε и логарифмической от n. Второй метод – со сложностью построения

структуры, полиномиально зависящей от d, ε и n, однако поиск требовал $O(n + k\log^3 n)$. Кроме того, оба метода применимы только для данных из R^k.

Первые алгоритмы со сложностью поиска, полиномиально зависящей от k, $\log n$, ε^{-1}, и полиномиальным временем построения структуры для фиксированного ε были предложены Indyk и Motwani [22] и Kushilevitz, Ostrovsky и Rabani [23]. Indyk и Motwani впервые используют упрощения задачи ε-ANN до «approximate point location in equal balls» (ε-PLEB). Для постановки задачи ε-PLEB точки в метрическом пространстве расширяются до шаров радиуса $(1 + \varepsilon)r$ с центром в данной точке; требуется определить, к какому шару принадлежит запрос q. Также в [22] был предложен второй метод, использующий концепцию locality-sensitive hashing относительно постановки задачи как ε-PLEB, осуществляющий поиск за $O(n^{1/(1+\varepsilon)})$, однако требующий околоквадратичной памяти (для малого ε). Кроме того, первый метод применим только для R^k, а второй – для пространства Хэмминга.

Вообще, концепция locality-sensitive hashing стала очень популярна в последнее десятилетие для решения задачи ANN. Другие работы, использующие концепцию locality-sensitive hashing, – [24–26]. Но все они обладают одним главным недостатком – каждый из алгоритмов хеширования ориентирован на узкий класс метрик, таких как расстояние Хэмминга, Джакарта, l_1, l_s нормы для евклидова пространства. Для того чтобы узнать, какой метод более эффективный, приходится составлять своего рода дайджесты [26].

Первая структура, имеющая топологию тесного мира (известная нам), решающая задачу ANN для k-мерного евклидова пространства, – Raynet [27]. Она является развитием более ранней работы тех же авторов Voronet [10], которая решала точную задачу NN для евклидовой плоскости. Изначально она позиционировалась как p-2-p-сеть, в которой в качестве идентификаторов узлов использовались координаты точек в двухмерной плоскости. В Raynet с узлами сети ассоциированы уже объекты из k-мерного евклидова пространства. В системах поддерживаются два уровня ссылок – короткие для обеспечения корректности работы «жадного алгоритма» и длинные для осуществления быстрого поиска. Коротким ссылкам соответствуют ребра графа Делоне, т.е. каждый объект имеет ссылки на объекты, являющиеся соседними к его области Вороного. Raynet отличается от Voronet тем, что каждый знает не всех своих соседей из области Вороного, т.е. его окрестность находится с некоторым приближением при помощи метода Монте-Карло.

По общей концепции Raynet – это наиболее близкая работа к нашей. Но в отличие от Raynet, мы предлагаем структуру для хранения объектов из произвольного метрического пространства.

Общее описание структуры

В этой работе рассматривается задача приближенного поиска ближайшего соседа. Она формулируется следующим образом: есть пространство предметной области D, на нем определена функция близости d: $D \times D \to \mathbf{R}$. Задано конечное множество $X = \{x_1, \ldots, x_n\}$, $X \subset D$. Необходим эффективный вероятностный способ нахождения элемента множества X, ближайшего к $q \in D$. Эффективный поиск значит, что сложность поиска логарифмически масштабируется с числом элементов в X. Поиск не гарантированный, т.е. результатом работы алгоритма может оказаться элемент, не являющийся истинным ближайшим, однако структура и алгоритм устроены так, чтобы эта вероятность была мала и её можно было корректировать, не изменяя структуру, варьируя только параметр алгоритма поиска.

Рассматриваемая структура $\boldsymbol{S}$ строится в виде сети, описываемой графом $G(V, E)$, где объектам из множества X однозначно сопоставлены вершины из множества V. Ребра E задаются алгоритмом построения структуры так, чтобы обеспечить корректную работу алгоритма «жадного поиска».

Поскольку в предлагаемой структуре с каждой вершиной ассоциирован элемент из множества X, то мы иногда будем употреблять слово «вершина», «элемент» или «объект», но смысл один. Вершины, имеющие общие ребра, мы иногда будем называть друзьями. Список вершин, имеющих общее ребро с вершиной v_i, будем называть списком друзей вершины v_i.

Алгоритм поиска

«*Жадный поиск*». Базовый алгоритм поиска в структуре осуществляется путем перехода по ребрам графа $G(V, E)$ от одной вершины к другой. Алгоритм принимает два параметра: запрос **query** и вершину $V_{enter\_point} \in V[G]$, с которой начинается поиск (такую вершину далее будем называть точкой входа). Начиная с точки входа, в каждой вершине алгоритм вычисляет значения метрики от запроса q и до всех вершин из френд-листа (соседних вершин, имеющих об-

щее ребро) текущей вершины, выбирает вершину с минимальным значением метрики. Если значение метрики между запросом и выбранной вершиной меньше, чем между запросом и текущим элементом, то алгоритм осуществляет переход к этой вершине, после чего повторяется. Алгоритм останавливается в вершине, среди множества друзей которой нет вершины, которая была бы ближе к запросу, чем она сама. Такую вершину мы называем локальным минимумом.

Сложность работы алгоритма Greedy_Search пропорциональна произведению длины пути, проделанного алгоритмом, l_{greedy} на среднее число соседей (друзей) у элементов

$$agv\_vertex\_degree = \frac{|E|}{|2V|}.$$

Причем существует такой класс графов, в которых $l_{greedy} \sim \log n$, что дает основания говорить, что сложность работы Greedy_Search в них пропорциональна $\log n$. Говорят, что если в графе $l_{greedy} \sim \log n$, то он обладает навигационными свойствами тесного мира. Далее мы приведем один из возможных алгоритмов построения такого рода графа.

```
Greedy_Search(q: object, venter_point: object)
1  vcurr ← venter_point;
2  dmin ← d(q, vcurr); vnext ← NIL;
3  foreach vfriend ϵ vcurr.getFriends() do
4    if d(query, vfriend) < dmin then
5      dmin ← d(q, vfriend);
6      vnext ← vfriend;
7  if vnext = Nil then return vcurr;
8  else return Greedy_Search(q, vnext);
```

Элемент, являющийся локальным минимумом относительно запроса q, может быть как истинно ближайшим элементом к запросу q из всего множества элементов X, так и ложно ближайшим.

Если бы каждый элемент в структуре имел среди своих друзей все элементы, области Вороного которых граничат с его собственной областью Вороного, то это исключало бы существование ложно локальных минимумов. Поддержание этого условия эквивалентно построению графа Делоне, который является двойственным к диаграмме Вороного.

Поскольку для абстрактных метрических пространств определение точного графа Делоне является невозможным [11] (исключая вариант полного графа), то избежать существования локальных минимумов не удается.

Но для задачи неточного поиска, определенной выше, это не играет существенной роли. Выходом является то, что для работы вероятностного поиска не обязательно строить весь граф Делоне [27]. Как будет показано далее, вероятность найти истинный ближайший элемент экспоненциально стремится к единице с ростом среднего числа ребер в аппроксимированном графе Делоне.

Серия поисков. Предположим, уже существует некоторая аппроксимация графа Делоне, но нас не устраивает точность, получаемая алгоритмом «жадного поиска».

Для того чтобы «обойти» ложно локальные минимумы, мы предлагаем следующую модификацию алгоритма поиска. Мы предлагаем использовать серию из m поисков, инициированных от случайных вершин, и выдавать в качестве результата элемент, ближайший к запросу из множества найденных. Алгоритм «жадного поиска» Greedy_Search(q, $v_{\text{enterPoint}} \in V$) детерминирован для каждой точки входа $v_{\text{enterPoint}} \in V$, он заканчивается либо успехом – нахождением истинного ближайшего, либо ошибкой – нахождением элемента, который не является ближайшим соседом для q.

То есть фактически поиск ближайшего к одному и тому же элементу может заканчиваться как нахождением истинного ближайшего, так и ошибочного ближайшего, в зависимости от того, с какой точки входа мы начали поиск.

Поскольку точку входа мы можем выбирать случайно, то появляется вероятность p нахождения истинного ближайшего к одному данному конкретному элементу q. Причем эта вероятность всегда отлична от нуля, т.к. всегда есть вариант, что точкой входа является истинный ближайший и алгоритм «жадного поиска» возвращает в качестве результата именно его.

Если вероятность найти истинный ближайший одним поиском равна p, то вероятность найти тот же самый элемент при m поисках $1 - (1 - p)^m$, т.е. вероятность ошибки падает экспоненциально с ростом количества поисков. Таким образом, мы можем улучшать точность поиска, увеличивая параметр m – количество производимых независимых поисков.

При значении $m = n$, где n – количество элементов в структуре, алгоритм сводится к полному перебору.

Если граф сети обладает свойствами тесного мира, то выбрать случайную вершину в нем можно за число случайных переходов, пропорциональное $\log n$.

Таким образом, сложность общего поиска будет $O(2m\log n)$, если сложность каждого элементарного поиска $\log n$.

Алгоритм поиска с многократным повторением выглядит следующим образом:

```
Multi_Search(object q, integer: m)
1   results: SET[objects];
2   for (i ← 0; i < m; i++) do
3     enter_point ← getRandomEnterPoint();
4     local_min ← Greedy_Search(query, enter_point)
5     if local_min ∉ results then
6       results.add(result);
7   return results;
```

```
Get_Best_Element(object: q, SET[object]: objects)
1   d_min ← ∞; o_best ← NIL;
2   foreach object o ∈ objects do
3     if d(q, o) < d_min then
4       o_best ← o;
5       d_min ← d(q, o);
6   return o_best;
```

Алгоритм построения структуры

Поскольку мы строим аппроксимацию графа Делоне, то существует большая свобода в выборе алгоритмов её построения. Например, в работе [27] для евклидова пространства строилась аппроксимация графа Делоне, которая минимизировала объем области Вороного при фиксированном числе ребер для каждой вершины соответствующего графа. В [28] в качестве алгоритма построения структуры использовался так называемый алгоритм соединения с ближайшими, основная идея которого заключается в том, что пересечение множества элементов, являющихся соседями по Вороному, и множества l ближайших элементов велико.

Граф, полученный предложенным алгоритмом, обладает свойствами тесного мира при условии, что данные поступают в случайном порядке.

```
Nearest_Neighbor_Add(object: new_object, integer: l, integer: init_attempts)
1   SET[object]: localMins ← MultiAttempts_Search(new_object, init_attempts);
2   SET[object]: u ← ∅; //neighborhood;
3     foreach object: local_min ∈ localMins do
4     u ← u ∪ local_min.getFriends();
6   отсортировать u так, чтобы выполнялось условие d(u[i], new_object) < d(u[i+1], new_object)
7   for (i ← 0; i < l; i++) do
8     u[i].connect(new_object);
9     new_object.connect(u[i]);
```

Алгоритм принимает в качестве аргументов три параметра: объект, который необходимо добавить в структуру, и два целых положительных числа l и init_attempts. Сначала алгоритм определяет множество локальных минимумов, используя процедуру Multi_Search, которая, в свою очередь, производит серию независимых поисков от init_attempts случайно выбранных элементов из множества объектов, уже добавленных в структуру. Затем формируется окрестность u, в которую входят все соседи каждого из найденных локальных минимумов. Множество u сортируется в порядке возрастания расстояния до добавляемого объекта new_object, после чего new_object соединяется с первыми K ближайшими элементами из множества u.

На рис. 1 изображена структура, построенная для точек из E^2.

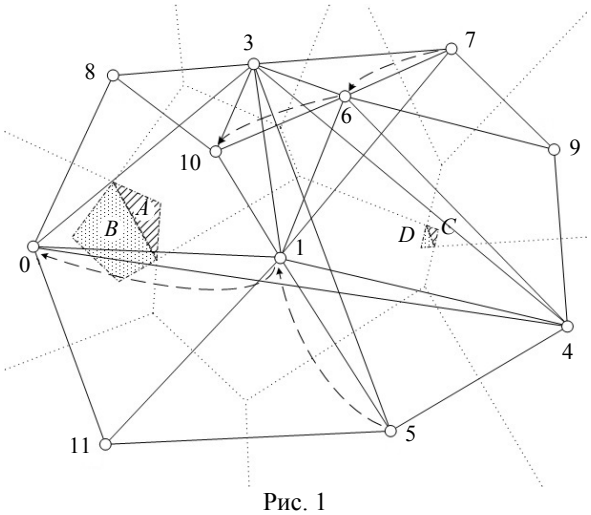

Рис. 1

Элементы обозначены кругами. Числа рядом с ними соответствуют порядку добавления. Сплошными линиями изображены связи между элементами (ребра графа). Пунктирные линии соответствуют границам областей разбиения Вороного. До полного графа Делоне не хватает ребра между элементами 0 и 10, 1 и 9. Структура получена алгоритмом при m = 3 и *attemptsNumber* = 5. Элемент с номером 0 будет локальным минимумом для запросов, попавших в заштрихованную область A, соответственно 10 – для B, 9 – для D и 1 – для области C. Штриховыми линиями изображены пути двух поисковых алгоритмов при поиске запроса q из области B; алгоритм, стартующий из вершины 7, останавливается на элементе 10, являющемся локальным минимумом; алгоритм, стартующий из вершины 5, – истинно ближайшим к запросу.

Результаты моделирования

Для того чтобы проверить правильность наших предположений о логарифмической зави-

симости поиска от количества элементов, мы реализовали приведенные выше алгоритмы.

В качестве тестового набора данных мы использовали случайно выбранные точки из E^k. В качестве функции близости была выбрана L_2 (евклидово расстояние).

В структуру добавлялось N элементов. Выбиралось такое число попыток поисков, чтобы вероятность нахождения истинного ближайшего элемента к запросу была не меньше 95%. Замерялось число перебранных элементов. На графике рис. 2 приведена зависимость процента перебранных элементов с ростом количества добавленных в структуру элементов.

Из графика видно, что с ростом числа элементов в структуре процент перебранных элементов падает, кривая на этом участке принимает вид прямой с углом наклона 45 градусов. Это дает основания говорить о логарифмической сложности поиска от числа перебранных элементов.

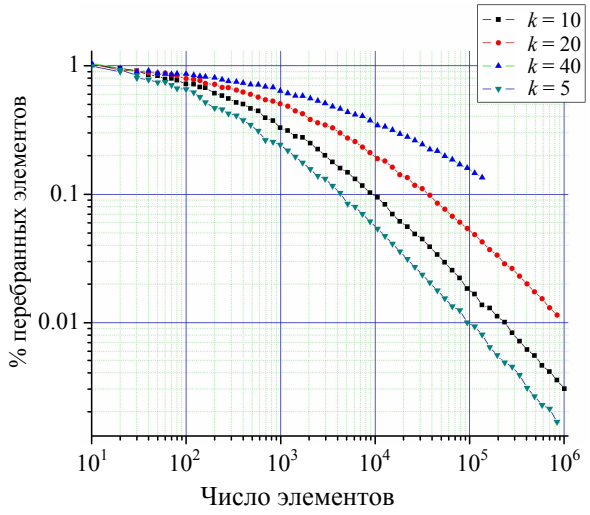

Рис. 2

Как видно из графика, кривая для более высоких размерностей ведет себя схожим образом.

Заключение

В настоящей статье предложен способ организации данных в структуру, имеющую свойства навигационного тесного мира, для произвольного метрического пространства при помощи модифицированного алгоритма соединения с k-ближайшими элементами. Описан алгоритм поиска ближайшего соседа на предложенной структуре.

Все предложенные алгоритмы используют только локальную информацию, что в сочетании с отсутствием центрального элемента у предложенной структуры открывает возможности для построения абсолютно децентрализованных систем поиска.

Представленные результаты моделирования подтверждают логарифмический характер сложности поиска от числа элементов.

Список литературы

1. Cover T.M., Hart P.E. Nearest neighbor pattern classification // Information Theory IEEE Transactions on. Jan. 1967. Vol. 13, No 1. P. 21–27.

2. Flickner M., et al. Query by image and video content: the QBIC system // Computer. Sep. 1995. Vol. 28, No 9. P. 23–32.

3. Salzberg S., Cost S. A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features // Machine Learning. 1993. Vol. 10, No 1. P. 57–78.

4. Sarwar B., Karypis G., Konstan J., Reidl J. Item-based collaborative filtering recommendation algorithms // Proceedings of the 10th International Conference on World Wide Web. New York, USA, 2001. P. 285–295.

5. Rhoads R.E., Rychli W., Unknown R. A computer program for choosing optimal oligonudeotides for filter hybridization, sequencing and in vitro amplification of DNA // Nucletic Acids Research. Oct. 1989. Vol. 17, No 21. P. 8543–8551.

6. Deerwester S., Dumais S., Furnas G. et al. Indexing by Latent Semantic Analysis // J. Amer. Soc. Inform. Sci. 1990. Vol. 41. P. 391–407.

7. Chávez E., Navarro G., Baeza-Yates R., Marroquín J.L. Searching in metric space // Journal ACM Computing Surveys (CSUR). Sep. 2001. Vol. 33, No 3. P. 273–321.

8. Kleinberg J. The Small-World Phenomenon: An Algorithmic Perspective // Annual ACM Symposium on Theory of Computing. 2000. Vol. 32. P. 163–170.

9. Aurenhammer F. Voronoi diagrams – a survey of a fundamental geometric data structure // ACM Computing Surveys (CSUR). Sep. 1991. Vol. 23, No 3. P. 345–405.

10. Beaumont O., Kermarrec A.-M., Marchal L., Riviere E. VoroNet: A scalable object network based on Voronoi tessellations // International Parallel and Distributed Processing Symposium. Long Beach, USA, 2007. P. 20.

11. Navarro G. Searching in metric spaces by spatial approximation // String Processing and Information Retrieval Symposium. Cancun, Mexico, 1999. P. 141–148.

12. Bentley J.L. Multidimensional binary search trees used for associative searching // Communications of the ACM. Sep. 1975. Vol. 18, No 9. P. 509–517.

13. Bentley J.L., Finkel R. Quad Trees: A Data Structure for Retrieval on Composite Keys // Acta Informatica. 1974. Vol. 4, No 1. P. 1–9.

14. Wong, Lee Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees // Acta Informatica. 1977. Vol. 9, No 1. P. 23–29.

15. Samet H. The design and analysis of spatial data structures. Reading, MA: Addison-Wesley, 1989.

16. Mount D.M., Arya S., Narayan O. Accounting for boundary effects in nearest-neighbor searching // Discrete & Computational Geometry. 1996. Vol. 16, No 2. P. 155–176.

17. Dobkin D., Lipton R. Multidimensional Searching Problems // SIAM J. Comput. 1976. Vol. 5, No 2. P. 181–186.

18. Bozkaya T., Ozsoyoglu M. Distance-based indexing for high-dimensional metric spaces // Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data. New York, USA, 1997. P. 357–368.

19. Yianilos P. Data structures and algorithms for nearest neighbor search in general metric spaces // SODA'93 Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms. Philadelphia, USA, 1993. P. 311–321.

20. Arya S., Mount D. Approximate nearest neighbor queries in fixed dimensions // SODA'93 Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms. Philadelphia, PA, USA, 1993. P. 271–280.

21. Kleinberg J. Two algorithms for nearest-neighbor search in high dimensions // STOC'97 Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing. New York, USA, 1997. P. 599–608.

22. Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality // STOC '98 Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing. New York, USA, 1998. P. 604–613.

23. Kushilevitz E., Ostrovsky R., Rabani Y. Efficient search for approximate nearest neighbor in high dimensional spaces // STOC'98 Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing. New York, NY, USA, 1998. P. 614–623.

24. Gionis A., Indyk P., Motwani R. Similarity Search in High Dimensions via Hashing // VLDB'99 Proceedings of the 25th International Conference on Very Large Data Bases. San Francisco, USA, 1999. P. 518–529.

25. Datar M., Immorlica N., Indyk P., Mirrokni V. Locality-sensitive hashing scheme based on p-stable distributions // SCG'04 Proceedings of the Twentieth Annual Symposium on Computational Geometry. New York, USA, 2004. P. 253–262.

26. Andoni A., Indyk P. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions // 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06). Berkeley, California, 2006. P. 459–468.

27. Beaumont O., Kermarrec A.-M., Rivière É. Peer to peer multidimensional overlays: approximating complex structures // Proceedings of the 11th International Conference on Principles of Distributed Systems. Berlin, Heidelberg, 2007. P. 315–328.

28. Krylov V.V., Logvinov A.A., Ponomarenko A.A., Ponomarev D.M. Metrized Small World Properties Data Structure // SEDE. Los Angeles, California, USA, 2008.

DATA STRUCTURE WITH SMALL WORLD PROPERTIES FOR SOLVING NEAREST NEIGHBOR SEARCH PROBLEM IN METRIC SPACE

A.A. Ponomarenko, Yu.A. Mal'kov, A.A. Logvinov, V.V. Krylov

A novel approach to solving the nearest neighbor search problem in metric space is considered. It is proposed as a data structure to use a graph with navigable small world properties and a gradient descent algorithm as a search algorithm. The problem of the existence of local minima is solved by a series of independent searches. Experimental data are presented to confirm logarithmic complexity of the search algorithm.

Keywords: similarity search in metric space, data structure, small world, probabilistic search, distributed systems.