УДК 004.023:629.3.072.1-022.233

МНОГОКРАТНЫЙ ПОИСК КРАТЧАЙШИХ ПУТЕЙ НА БОЛЬШИХ ГРАФАХ МЕТОДОМ БУТСТРЭППИНГА

М.П. Ревотюк
Доцент кафедры информационных технологий автоматизированных систем БГУИР, кандидат технических наук, доцент

Н.В. Хаджинова
Старший преподаватель кафедры информационных технологий автоматизированных систем БГУИР

Белорусский государственный университет информатики и радиоэлектроники, Республика Беларусь
E-mail: rmp@bsuir.by

М.П. Ревотюк
Окончил Минский радиотехнический институт. Доцент кафедры информационных технологий автоматизированных систем БГУИР. Направления научных исследований: моделирование и оптимизация управления дискретными процессами в реальном времени, защита информации, системное и объектно-ориентированное программирование и проектирование.

Н.В. Хаджинова
Окончила Белорусский государственный университет информатики и радиоэлектроники. Заместитель декана по учебно-методической работе факультета инфокоммуникаций, старший преподаватель кафедры информационных технологий автоматизированных систем БГУИР. Направления научных исследований: моделирование и оптимизация управления взаимодействующими процессами, объектно-ориентированное программирование и проектирование.

Аннотация. Предлагаются приемы ускорения многократного поиска кратчайших путей на больших графах, когда порядок порождаемых деревьев путей существенно меньше порядка графа. Однократная инициализация расширяемой области переменных состояния, выделение предопределенных решений снижает сложность поиска путей до линейной зависимости от объема сканируемого пространства.

Ключевые слова: транспортные сети, кратчайшие пути, вычислительная сложность.

Постановка задачи. Известно, что при поиске кратчайших путей на нагруженном ориентированном графе $G(N, A)$, где N – множество вершин, A – множество дуг с весовой функцией $W : A \to R^+$, время построения дерева путей одним из лучших для подобной задачи алгоритмом Дейкстры растет в первом приближении по закону $x \cdot \log_2 x$ с увеличением расстояния x от корня дерева [1]. Особенность алгоритма Дейкстры – однократный просмотр дуг формируемого дерева, что отражается появлением количества дуг в асимптотиках вычислительной сложности процедур поиска. Например, реализация таких процедур с отображением очередей анализируемых вершин на кучи Фибоначчи характеризуется сложностью $O(m + n \cdot \log_2 n)$, где $m = |A|$, $n = |N|$. Отображение очереди вершин на вектор

размером L позволяет снизить сложность до величины $O(m+n\cdot L)$, где L – максимальная длина дуги графа [1,2].

Существует ряд приемов ускорения поиска кратчайших путей, использующих идею сокращения количества анализируемых дуг графа: целенаправленный поиск; встречный поиск; многоуровневый подход; ограничение локальных областей поиска [1-3]. Реализация перечисленных приемов предполагает предварительное формирование вспомогательного графа, отображающего вершины исходного графа и деревья кратчайших путей. Затем построение дерева путей идет по волновой схеме однократного просмотра дуг, реализуемой алгоритмом Дейкстры. Результат отображается на исходный граф за время $O(m)$.

Наличие в асимптотиках вычислительной сложности параметра n отражает необходимость выполнения перед построением дерева действий, сложность которых $O(n)$. Это снижает эффективность алгоритма Дейкстры в задачах определения множества кратчайших путей. Пример подобной задачи – расчет подматриц кратчайших расстояний: необходимо построить кратчайшие пути от заданного множества исходных вершин $S\in N$ до всех вершин из множества $F\in N$. Практически всегда в таких задачах выполняется условие $|F|<n$.

Расчет подматриц можно проводить прямолинейным использованием агоритма Флойда, но при этом для любых значений $|S|$ и $|F|$ вычислительные затраты имеют оценку $O(n^3)$, а потребность в памяти – $O(n^2)$ [1]. Для разреженных графов реальных транспортных сетей с переменной структурой в таких случаях лучшим оказывается алгоритм Дейкстры, где при потребности в памяти $O(n+m)$ оценка вычислительных затрат – не хуже $|S|O(n^2)$. Далее предлагаются приемы улучшения таких оценок, основанные на исключении излишних операций открытыми для расширения процедурами построения множества деревьев кратчайших путей для всех элементов $S\in N$.

Ревизия алгоритма построения деревьев кратчайших путей. Известно, что вполне достаточным для реализации алгоритма Дейкстры представлением разреженного нагруженного графа $G(N,A)$ является структура смежности *FSF* (*Forward Star Form*) [1]. В этом случае для каждой вершины x эффективно представлено множество непосредственно достижимых смежных вершин x', $x'=\{k \mid w(x,k)\geq 0\}$, где $w(x,k)$ – вес дуги $x\to k$, $x,k\in N$. Объем памяти для хранения структуры смежности – $O(n+m)$.

Пространство состояний поиска решения алгоритмом Дейкстры включает:

D – массив расстояний от корня дерева, $D=\{D(i),\ i\in N\}$;

P – массив номеров предшествующих вершин, $P=\{P(i),\ i\in N\}$;

Q – очередь вершин, $Q=\{Q_i, i\in N\}$, элементы которой упорядочены по текущему значению расстояния от корня дерева [1,2].

Процесс построения дерева кратчайших путей имеет волновой характер до исчерпания возможности его развития из исходной вершины (рисунок 1).

На итерациях построения дерева последовательно выполняются операции:

−выборка вершины из очереди вершин с минимальным расстоянием от корня дерева;

−развитие дерева из выбранной вершины, когда для всех ее выходных дуг выполняется процедура релаксации с включением новых вершин в очередь.

Вершина графа может находиться в следующих последовательно фиксируемых состояниях: не рассматривалась, рассматривается и рассмотрена (постоянно помечена). Признак

состояния явно обычно не используется, а его отображение проводится неявно на элементах $D = \{D(i),\ i \in N\}$. Последовательный переход состояний окажется принципиальным для предлагаемого метода построения алгоритма поиска кратчайших путей.

Все вершины графа перед построением дерева помечаются парами начальных значений $D(i) \leftarrow \infty,\ P(i) \leftarrow i,\ i \in N$. Принадлежность произвольной вершины $x \in N$ дереву путей на любом этапе его построения определяется оценкой выражения $(D(x) = \infty) \vee (P(x) = x)$. На практике именно вершины дерева путей значимы для задач координации транспортных операций.

```
bool SPT(r,t){
 foreach i ∈ N do D(i) ← ∞, P(i) ← i;
 D(r) ← 0;  Q ← {r};
 do {
   i ← Q1, Q ← Q \ {i};
   if   (i ≡ t)  return true;
   foreach j ∈ i' {
     dj = D(i) + w(i, j);
     if (D(j) > dj) {
       if  (D(j) < ∞)  Q ← Q \ {j};
       D(j) ← dj, P(j) ← i;  Q ← Q ∪ {j};
     }
   }
 } while  (Q ≠ ∅);
 return  false;
}
```

Рисунок 1. Традиционная версия реализации алгоритма Дейкстры

Отображение состояния вершин на множествах D и P вынуждает каждый раз перед построением дерева устанавливать начальные значения для всех элементов. Вычислительная сложность такой операции – $O(n)$. В случае многократного построения кратчайших путей на больших графах количество вершин дерева чаще всего оказывается существенно меньше значения n.

Метод бутстрэппинга. Как известно, бутстрэппинг (bootstrapping) – название некоторых методов и процессов, реализующих принцип повторения и саморазвития без воздействия извне. Очевидно, что процессы построения дерева кратчайших путей вполне соответствуют такому принципу, но в публикациях он лишь неявно используется для доказательства корректности алгоритма Дейкстры методом математической индукции. Однако конструирование вычислительных схем на основе реализации принципа повторения и саморазвития может получить формально обоснованные эффективные решения по критерию “память - быстродействие”. Это представляет практический интерес в задачах большой размерности или работе в реальном времени.

Метод бутстрэппинга может быть реализован разными способами, отличающимися определениями структуры пространства состояния. Рассмотрим далее возможные приемы улучшения реализации схемы Дейкстры для поиска кратчайших путей на динамически меняющихся графах большой размерности.

Учитывая последовательный характер изменения состояния вершин дерева в соответствии со значением расстояния от его корня, очевидна идея включения такой последовательности в качестве этапа построения множества деревьев. Для ее реализации необходимо обеспечить распознавание исходного состояния вершин новых деревьев. Возможная реализация обсуждаемой идеи (рис.2) базируется на дополнении пространства состояния глобальной переменной e с целью идентификации вершин разных деревьев. Это позволяет

выполнить инициализацию массивов D и P один раз.

Переменная e содержит начало интервала номеров очередного дерева (функция $SPM(r,t)$). Построение дерева приводит к изменению элементов массива P, соответствующих подвергшимся анализу вершинам графа. Предикат $(P(j)<e)$ определяет множество не рассмотренных вершин и соответствует состоянию $(D(j)=\infty)$, $j \in N$. Использование такого предиката позволяет исключить повторную установку начального состояния множеств D и P. Как следствие, вычислительная сложность построения очередного дерева путей функцией $SPM(r,t)$ зависит лишь от количества фактически рассматриваемых дуг.

```
void SPL() {
  e = 0;
 foreach i ∈ N  P(i) = e;
}

bool SPM (r,t) { // Поиск кратчайшего пути  r → t
  D(r) ← 0; Q ← {r}; e ← e + n; P(r) ← r + e;
  do {
    i ← Q1, Q ← Q \ {i};
    if   (i ≡ t)  return true;
    foreach  j ∈ i' {
      dj = D(i) + w(i, j);
      if  (P( j) < e) ∨ ((D( j) > dj ) ∧ (Q ← Q \ { j})) {
        D( j) = dj , P( j) = i + e;  Q ← Q ∪ { j};
      }
  } while  (Q ≠ ∅);
  return   false;
 }
```

Рисунок 2. Пример реализации алгоритма Дейкстры для регулярного поиска кратчайших путей

В случае решения задач поиска кратчайших путей на множестве вершин некоторой компактной части графа (рисунок 3) возможна несложная модификация алгоритма Дейкстры (рисунок 2), когда используется динамически формируемый расширяемый справочник номеров вершин создаваемого дерева кратчайших путей (рисунок 4).

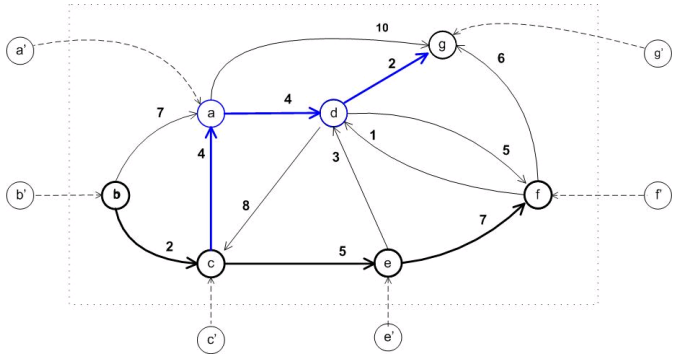

Рисунок 3. Пример задачи поиска кратчайших путей на подграфе

Здесь построение любого дерева с корнем $r \in \{a,b,c,d,e,f\}$ может проводиться на изображенных сплошными линиями дугах подграфа, но переменные пространства состояний алгоритма Дейкстры должны представлять весь граф. Жирными линиями выделены дуги дерева кратчайших путей с корнем $r=b$, среди которых черным цветом выделены дуги кратчайшего пути $r \to f$.

Пусть по ходу построения дерева путей функцией $SPH(r,t)$ (рисунок 4) операция $i = h^{+}(x)$ получает номер помещаемой в справочник ссылки на вершину x, а обратное отображение выполняет операция $x = h^{-}(i),\ x \in N$. Номера зарегистрированных в справочнике вершин, представляя фактически требующийся подграф, используются в качестве индексов массивов расстояний $D_k = \{D_k(i),\ i \in \overline{1,n(k)}\}$ и предшествующих вершин $P_k = \{P_k(i),\ i \in \overline{1,n(k)}\}$, где $n(k)$ – количество узлов дерева после окончательной фиксации k вершин.

```
bool SPH(r,t) { // Поиск кратчайшего пути r → t
 I = h+(r), K = h+(t); D(I) ← 0; Q ← {I}; e ← e + n; P(I) ← I + e;
 do {
  I ← Q1, Q ← Q \ {I};
  if (I ≡ K) return true;
  i = h−(I); di = D(I);
  foreach j ∈ i' {
   J = h+(j); dj = di + w(i, j);
   if (P(J) < e) ∨ ((D(J) > dj) ∧ (Q ← Q \ {J})) {
    D(J) = dj, P(J) = I + e; Q ← Q ∪ {J};
   }
 } while (Q ≠ ∅);
 return false;
}
```

Рисунок 4. Пример реализации метода бутстрэппинга на основе справочника

Далее предлагается реализация метода бутстрэппинга на основе аналогии складывания ветвей дерева от корня до последнего постоянно включаемого в дерево узла.

Построенное дерево кратчайших путей – связный граф по определению. Если s – исходная вершина, а x – произвольный узел или лист дерева путей, $s, x \in N$, то после завершения поиска кратчайший путь $p(s,x)$ можно восстановить обратным движением из листа x. Последовательность посещаемых вершин обратного пути

$$p(x,s) = \{x,\ P(x),\ P(P(x)),\ldots\ P(\ldots P(P(x)))\ldots s\} \qquad (1)$$

после тривиального изменения направления перечисления элементов представляет искомый кратчайший путь

$$p(s,x) = \{s,\ \ldots\ P(\ldots P(P(x)))\ldots P(P(x)), P(x),\ x\}. \qquad (2)$$

Можно заметить, что согласно выражениям (1) и (2), $|\ p(x,s)\ | = |\ p(x,s)\ |$, а их элементы упорядочены по значениям расстояния от корня дерева путей.

Альтернативы формирования дерева кратчайших путей отражаются листьями, путь от корня до которых не обязательно кратчайший, но восстанавливается по правилу построения $p(x,s)$. Обозначим L – множество листьев текущего дерева, L^{*} –подмножество листьев без постоянной пометки. Очевидно, что в любой момент построения дерева кратчайших путей его узлы можно отобразить на элементы множества

$$T(s) = \bigcup_{x \subset L} p(s,x)\cdot \qquad (3)$$

Расширение дерева кратчайших путей происходит только из некоторого листа без пометки, а листья из множества

$$T^H(s) = \bigcup_{x \in H} p(s,x),\ H = L \setminus L^*, \tag{4}$$

представляют лишь исторический интерес и становятся пассивными. Так как каждому элементу $x,\ x \in T(s),$ соответствует $d(x)$ – длина кратчайшего пути до корня s, то парами $(x, d(x))$ активные элементы множества $T^Q(s) = T(s) \setminus T^H(s)$ представляют приоритетную очередь [1], а пассивные элементы заменяют массив расстояний D.

Отсюда можно записать уравнение динамики процесса построения дерева кратчайших расстояний

$$\begin{cases} x_{k+1} = \arg\min\{ d(x),\ x \in L_k^* \}, \\ L_{k+1}^* = L_k^* \setminus x_{k+1}, \\ L_{k+1+0}^* = L_{k+1}^* \cup \{x \subseteq x'_{k+1}\}, \\ H_{k+1} = H_k \cup \{x_{k+1}\}. \end{cases} \tag{5}$$

Здесь этап расширения приоритетной очереди представлен менее подробно, чем в ранее рассмотренных вариантах реализации алгоритма построения дерева, а также для краткости опущена проверка достижимости вершины t. Однако очевидно, что выражение (5) – пригодное для реализации бутстрэппинга рекуррентное уравнение первого порядка, где состояние процесса на любом этапе k представлено тройкой $(x_k, p(x_k), d(x_k))$. Начальное состояние процесса построения дерева с корнем s соответствует тройке $(s, s, 0)$. Условие завершения процесса – $(L_k^* = \varnothing)$ или $(x_k = t)$, но в любом случае $k \le n$.

Эффективная в вычислительном отношении реализация операций для перехода в следующее состояние возможна на основе учета инвариантов любого состояния:

упорядочение элементов L^* по условию $d(x_{k-1}) \le d(x_k) \le d(x_{k+1})$ позволяет ускорить поиск элемента x_k, а также обеспечивает сохранение такого упорядочения после обновления листа без постоянной пометки;

упорядочение элементов $T(s)$ по значениям x позволяет проводить быструю проверку условия $x \in L_k^*$, а также получать доступ к парам $(x, d(x))$ или $(x, p(x))$.

Поддержка нескольких законов упорядочения для элементов динамически изменяющегося множества – типовая задача современных информационных технологий. В рамках объектно-ориентированных технологий говорят об итераторах, а в технологиях баз данных – о представлениях (view) или индексации таблиц данных. Переключение между законами упорядочения выполняется по потребности конкретной операции.

В терминах реляционной модели данных представим переменные состояния процесса построения дерева элементами тернарного отношения со схемой $Trace(x, d, p)$, где *Trace* – имя типа отношения, x – номер узла дерева, d – расстояние от корня до узла дерева, p – номер предшествующего узла кратчайшего пути в узел x. Кортежи отношения типа *Trace* в этом случае адресуются значениями первичного ключа с использованием некоторого метода доступа.

Отсюда следует, что для реализации схемы алгоритма Дейкстры достаточно:

назначить атрибут x в качестве первичного ключа отношения T со схемой *Trace*;

определить для представления $p(s,x)$ класс с операциями доступа к кортежам расширяемого отношения по ключу;

кортежи отношения должны оставаться упорядоченными по значению атрибута d ;

элементы приоритетной очереди будут представлены подмножеством кортежей отношения T , для которых справедливо условие $d \geq \overline{d}$, где $\overline{d}$ – текущее минимальное удаление листьев растущего дерева от его корня.

Процесс построения методом бутстрэппинга дерева кратчайших путей с корнем в вершине b для ранее изображенного подграфа (рисунок 3) соответствует (5) и завершается формированием трассы порождения кортежей отношения T (рисунок 5 и 8). Цветом выделены анализируемые по ходу построения дерева ассоциативные связи между состояниями.

Можно заметить, что из каждого листа без пометки порождается не более одного экземпляра нового кортежа, который до выборки из приоритетной очереди может быть неоднократно модифицирован (например, T1 и T4). Таким образом, операция захвата памяти для нового кортежа из пула свободных элементов реализуется тривиальным инкрементом счетчика листьев текущего дерева.

Далее для изложения идеи реализации метода бутстрэппинга будем использовать понятия объектно-ориентированного программирования на языке C++.

Класс объектного представления отношения T содержит следующие операции:

T(type_x *x*, type_d *d*, type_x *p*) – создание кортежа с преставлением корня дерева;

bool operator+=(type_x *x*) – возвращает истинное значение после успешного создания кортежа с новым ключом *x*;

T &operator[](type_x *x*) – возвращает ссылку на существующий кортеж типа T с заданным ключом *x*;

T ***operator new** (*T* &target, type_x *x*, type_d *d*, type_x *p*) – создание нового экземпляра кортежа на месте существующего с сохранением упорядочения кортежей отношения T по возрастанию значений атрибута d.

bool operator(type_x &*x*, type_d &*d*) – чтение атрибутов *x* и d очередного кортежа из представления приоритетной очереди.

Псевдокод реализации функции поиска кратчайших путей между заданными вершинами s и t (рисунок 6) предназначен для построения методом бутстрэппинга множества $p(s,t)$, представленного элементами отношения *T*.

Псевдокод реализации функции поиска кратчайших путей между заданными вершинами s и t (рис. 6) предназначен для построения методом бутстрэппинга множества $p(s,t)$, представленного элементами отношения *T*.

Здесь ранее определенные понятия D, P и Q (рис. 1) неявно представлены кортежами отношения T , для которых внутренним механизмом хранения кортежей в любой момент поддерживается два варианта упорядочения:строгий порядок по значениям атрибута x (по определению понятия первичного ключа);нестрогий порядок по возрастанию значений атрибута d, что необходимо для реализаций операций над приоритетной очередью.

Возможности объектно-ориентированных технологий позволяют представить понятия (3), (4) и (5) полиморфным классом, объекты которого представляет клеточный конечный автомат, порождающий описание дерева кратчайших путей с заданным корнем. Такой объект должен взаимодействовать с моделью развиваемой и изменяемой транспортной сети, представляемой отношением *FSF,* и формировать описание дерева кратчайших путей в виде отношения типа *Trace*.

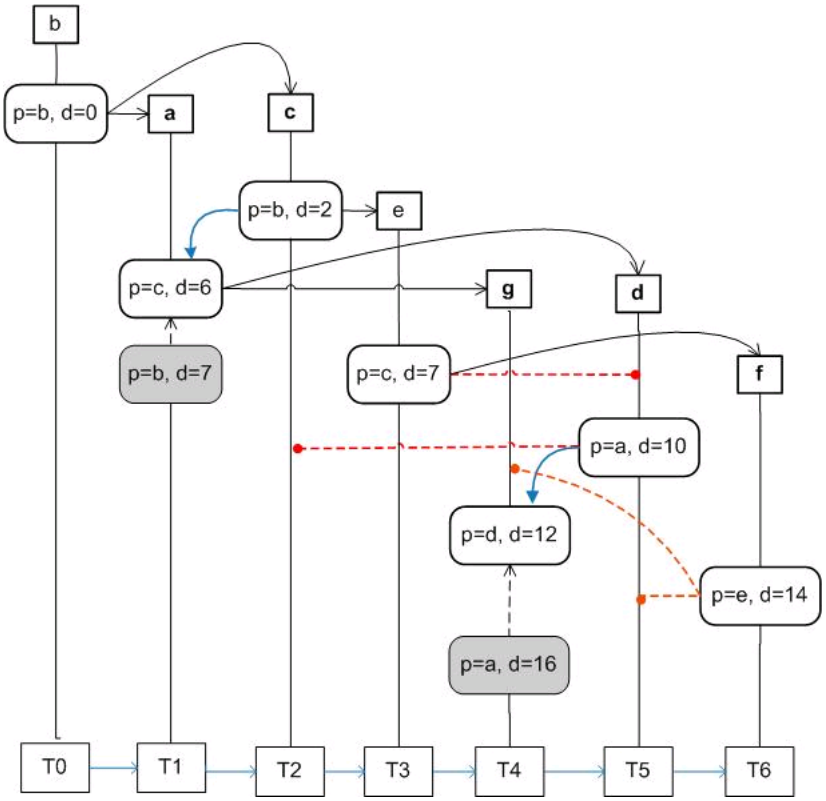

Рисунок 5. Трасса построения дерева кратчайших путей методом бутстрэпинга

Практически интересные задачи поиска кратчайших путей на динамически изменяющихся неполных графах большой размерности удобно решать с использованием баз данных. В таких случаях процедуры поиска могут строиться на основе идиом и механизмов манипулирования данными (рисунок 7). В частности, поддержка двух видов упорядочения кортежей отношения T может обеспечиваться индексацией баз данных.

Здесь приведен исходный текст работающей версии тестовой программы в среде системы программирования FoxPro [6], демонстрирующий пример реализации метода бутстрэппинга на основе идиом языка xBase и выражений (3), (4) и (5).

```
bool SPT(r,t) { // Поиск кратчайшего пути r → t
 type_w d = 0;
 type_x i = r;
 Trace T(i,d,i);
 do {
  if (i ≡ t) return true;
  foreach (j ∈ i') {
   d_j = d + w(i, j);
   if (T+= j) ∨ (T[j].d > d_j) new T(T, j,.d_j, i);
 } while T(i,d);
 return false;
}
```

Рисунок 6. Пример реализации алгоритма Дейкстры методом бутстрэппинга

```
proc spt && Поиск кратчайшего пути s->t
 parameter s,t
 private x,d,r,e
 insert into Trace (x,p,d) values (s,s,0)
 select Trace
 set order to d
 scan nooptimize all for (x!=t) and seek(x,"Graph")
  scatter memvar fields x,d
  m->r=recno()
  set order to x
  select Graph
  scan rest while x=m.x
   e=m.d+w
   select Trace
   if seek(Graph.y)
    if d>e
     replace p with m.x, d with e
    endi
   else
    insert into Trace (x,p,d) values (Graph.y,m.x,e)
   endi
  ends
  select Trace
  set order to d
  goto record m->r
 ends
return
```

Рисунок 7. Пример реализации метода бутстрэппинга на языке xBase

Очевидно, что отношение *T* содержит кортежи, соответствующие ветвям дерева вне кратчайшего пути (рисунок 8).

```
create table Graph (x C(3), y C(3), w N(3))
append from Graph.txt delimited
index on x to Graph && Представление графа в виде FSF
create table Trace (x C(3), p C(3), d N(5))
index on x tag x && Справочник вершин дерева
index on d tag d && Приоритетная очередь

do spt with "b","*"

select Graph
brow title "FSF:Graph"
set order to
brow title "DBF:Graph"
use
select Trace
brow title "PQA:Trace"
set order to
brow title "DBF:Trace"
use
return
```

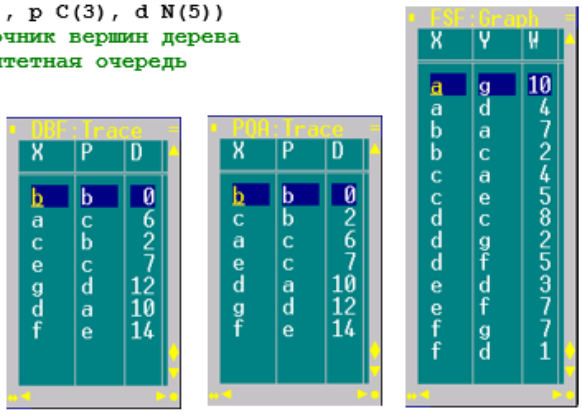

Рисунок 8. Пример применения метода бутстрэппинга для построения дерева всех путей

Характерная особенность результатов бутстрэппинга – отсутствие в $T(s)$ значений расстояния и номера предшествующей вершины для вершин вне дерева. Выявление таких вершин посредством прямолинейной проверки вхождения в $T(s)$ на практике обычно не требуется. Рассмотренный метод построения кратчайших путей не запрещает известные приемы совершенствования схемы алгоритма Дейкстры, например, использование предопределенных решений, встречного поиска, проецирование кратчайших путей на дуги графа [3-5]. Однако реализация таких приемов требует некоторой модификации процессов ветвления. Например, ассоциация дуг с оптимальными решениями может быть эффективно представлена характеристическими множествами признаков вхождения вершин в заранее выделенные любым способом подмножества вершин [3]. Очевидно, что метод бутстрэппинга остается работоспособным для решения задач поиска путей, но для этого требуется уточнить итератор выходных дуг листьев дерева.

Заключение. Таким образом, процесс построения кратчайших путей на графах старым, хорошо изученным алгоритмом Дейкстры, в форме, согласованной с возможностями современных технологий. Структуризация процедуры поиска непосредственно проецируется на понятия объектно-ориентированного моделирования и программирования, идиомы систем управления баз данных для создания открытых для расширения и детализации систем. Динамическое выделение области поиска решения линейно снижает вычислительную сложность локальных задач поиска деревьев кратчайших путей на больших графах.

Литература

[1]. Deo, N. Shortest-Path Algorithm: Taxonomy and Annotation/Deo N., Chi-yin Pang//Networks, 1984. – Vol. 14. – P. 275–323.

[2]. Fredman, M.L. Fibonacci heaps and their uses in improved network optimization algorithms/ M.L. Fredman, R.E. Tarjan// Journal of the Association for Computing Machinery, vol. 34, 1987. – P. 596–615.

[3]. Ревотюк, М.П. Быстрый поиск кратчайших путей на графах с предопределенными решениями/ М.П. Ревотюк, М.К. Кароли, Н.В. Хаджинова//Доклады Белорусского Государственного университета Информатики и Радиоэлектроники, № 4(82), 2014. – С. 73-79.

[4]. Ревотюк, М.П. Поиск кратчайших путей на графах полиморфных сетей с ограничениями/ М.П. Ревотюк, Ю.И. Дарадкех, В.А. Кирейчук//Известия Белорусской инженерной академии, № 1(17)/1, 2004. – С. 126–129.

[5]. Клепинин, В. Visual FoxPro в подлиннике/ В. Клепинин, Т. Агафонова //СПб.: БХВ-Петербург, 2007. – 1216 с.

MULTIPLE SEARCH OF THE SHORTEST PATHS ON THE BIG GRAPHS BY THE BOOTSTRAPPING METHOD

M.P. REVOTJUK, PhD
Department of Information Technologies of Automated Systems of BSUIR, Associate Professor

N.V. KHAJYNOVA
Department of Information Technologies of Automated Systems of BSUIR, Senior Lecturer

Belarusian State University of Informatics and Radioelectronics, Republic of Belarus
E-mail: rmp@bsuir.by

Abstract. On the classical problem of searching the shortest paths in massive graphs considered the possibility of accelerating the search procedure by incorporating a priori information about the search space. Global initialization of state variables predefined search and selection solutions can improve performance of multiple procedures to find paths to a linear dependence on the volume of the scanned area.

Keywords: transport networks, shortest paths in graphs, computational complexity.