***Н.И. Листопад**, д.т.н., профессор, заведующий кафедрой информационных радиотехнологий Белорусского государственного университета информатики и радиоэлектроники,*
***И.А. Карук**, магистрант Белорусского государственного университета информатики и радиоэлектроники,*
***А.А. Хайдер**, аспирант Белорусского государственного университета информатики и радиоэлектроники*

Алгоритмы поиска кратчайшего пути и их модификация

В статье представлен обзор основных тенденций и методов поиска кратчайшего пути передачи информации в сетях телекоммуникаций. Описаны основные алгоритмы и их модификация. Особое внимание уделено модифицированному алгоритму Дейкстры, учитывающему при поиске кратчайшего пути требования QoS. Описан сам алгоритм и представлена его диаграмма классов.

Введение

Мультисервисная сеть должна обладать надежностью и обеспечивать высокую скорость и низкую стоимость передачи данных в расчете на единицу объема информации.

Основная задача мультисервисных сетей нового поколения заключается в обеспечении работы разнородных информацион-

ных и телекоммуникационных систем и приложений в единой транспортной среде, когда для передачи данных и мультимедийного трафика используется единая инфраструктура.

Сеть нового поколения отличается новыми возможностями:

а) универсальным характером обслуживания разных приложений;

б) независимостью от технологий услуг связи и гибкостью получения набора, объема и качества услуг;

в) полной прозрачностью взаимоотношений между поставщиком услуг и пользователями.

Интеграция трафика разнородных данных и речи позволяет добиться качественного повышения эффективности информационной поддержки управления предприятием или организацией, при этом использование интегрированной транспортной среды позволяет снизить издержки на создание и эксплуатацию сети. Оптимально функционирующая мультисервисная сеть использует единый канал для передачи данных разных типов, позволяет уменьшить разнообразие типов оборудования, применять единые стандарты, технологии и централизованно управлять коммуникационной средой.

Базовыми понятиями мультисервисных сетей являются обеспечение требований *QoS* (*Quality of Service*) и *SLA* (*Service Level Agreement*), то есть качество обслуживания и соглашение об уровне (качестве) предоставления услуг сети. Переход к новым мультисервисным технологиям изменяет саму концепцию предоставления услуг, когда качество гарантируется не только на уровне договорных соглашений с поставщиком услуг и требований соблюдения стандартов, но и на уровне технологий и операторских сетей.

Средства поддержки качества обслуживания в современных протоколах маршрутизации в последнее время достаточно сильно изменились, прежде всего, за счет пересмотра метрик, используемых при выборе маршрута. Во-первых, происходит отказ от топологических метрик – числа переприемов (hops), характерного для протокола RIP, и осуществляется переход на QoS-метрики, основанные на учете основных показателей качества обслуживания: скорости передачи, средних задержек, вариации задержек, уровня потерь в трактах передачи сети. В результате маршруты передачи пакетов того или иного трафика прокладываются с учетом QoS-показателей вдоль них. Во-вторых, все больше протоколов поддержи-

вают так называемые композитные (комбинированные) метрики, в рамках которых учитываются одновременно несколько основных QoS-показателей.

1. Алгоритмы поиска кратчайших путей

Благодаря своему широкому применению, теория о нахождении кратчайших путей в последнее время интенсивно развивается и используется в различных сферах деятельности, например, для нахождения оптимального маршрута между двумя объектами на местности (кратчайший путь от дома до университета), для нахождения оптимального маршрута при перевозках, в системах автопилота, в системах коммутации информационных пакетов в сети Internet и т.п.

Кратчайший путь можно определить, используя графокомбинаторные и потоковые математические модели. Приведем пример некоторых алгоритмов нахождения кратчайшего пути, базирующихся на теории графов:

- алгоритм Дейкстры (используется для нахождения оптимального маршрута между двумя вершинами);
- алгоритм Флойда (используется для нахождения оптимального маршрута между всеми парами вершин);
- алгоритм Беллмана-Форда (для нахождения кратчайшего пути от одной вершины графа до всех остальных);
- алгоритм A* (используется для нахождения маршрута с наименьшей стоимостью от одной вершины (начальной) к другой (конечной), используя алгоритм поиска по первому наилучшему совпадению на графе);
- алгоритм Джонсона (используется для нахождения кратчайшего пути между всеми парами вершин взвешенного ориентированного графа);
- волновой алгоритм (переборный алгоритм, основанный на методе поиска в ширину) находит путь между вершинами графа, содержащего минимальное количество промежуточных ребер);
- алгоритм Габова (используется для нахождения кратчайшего пути с помощью масштабирования);
- алгоритм Карпа (используется для отыскания цикла с наименьшим общим весом).

Указанные алгоритмы легко выполняются при малом количестве вершин в графе. При увеличении их количества задача поиска кратчайшего пути усложняется. В этой связи ак-

туальной является задача разработки собственных алгоритмов поиска оптимального пути или модификация существующих подходов на базе комбинированных метрик.

Рассмотрим более подробно вышеперечисленные существующие алгоритмы поиска кратчайшего пути.

Алгоритм Дейкстры

Алгоритм Дейкстры (*Dijkstra's algorithm*) [1, 2] – алгоритм на графах, изобретенный нидерландским ученым Э. Дейкстрой в 1959 году. Алгоритм позволяет находить кратчайшее расстояние от одной из вершин графа до всех остальных. Алгоритм работает только для графов без ребер отрицательного веса и широко применяется в программировании и технологиях, например, его использует протокол OSPF для устранения кольцевых маршрутов [1, 2]. Алгоритм Дейкстры решает задачу о кратчайших путях из одной вершины для взвешенного ориентированного графа $G = (V, E)$ с исходной вершиной s, в котором веса всех ребер неотрицательны ($\omega(u, v) \geq 0$ для всех $(u, v) \in E$), где V – это непустое множество вершин или узлов; E – это множество пар (в случае неориентированного графа неупорядоченных) вершин, называемых ребрами.

В процессе работы алгоритм Дейкстры поддерживает множество $S \subseteq V$, состоящее из вершин υ, для которых $\delta(s, \upsilon)$ уже найдено (т.е. $d[\upsilon] = \delta(s, \upsilon)$). Алгоритм выбирает вершину $u \in V \setminus S$ с наименьшим $d[u]$, добавляет u к множеству S и производит релаксацию всех ребер, выходящих из u, после чего цикл повторяется. Вершины, не лежащие в S, хранятся в очереди Q с приоритетами, определяемыми значениями функции d. Предполагается, что граф задан с помощью списков смежных вершин. Каждой вершине из V сопоставляется метка – минимальное известное расстояние от этой вершины до α. Алгоритм работает пошагово – на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

Алгоритм Флойда-Уоршелла

Алгоритм Флойда-Уоршелла – динамический алгоритм для нахождения кратчайших расстояний между всеми верши-

нами взвешенного ориентированного графа. Разработан в 1962 году Робертом Флойдом и Стивеном Уоршеллом [2, 3].

Краткое описание алгоритма. Пусть вершины графа $G = (V, E)$, $V = n$ пронумерованы от 1 до n и введено обозначение d_{ij}^k для длины кратчайшего пути от i до j, который кроме самих вершин i, j, проходит только через вершины 1...k. Очевидно, что d_{ij}^0 – длина (вес) ребра (i, j), если таковое существует (в противном случае его длина может быть обозначена как ∞).

Существует два варианта значений d_{ij}^k, $k \in (1,..., n)$:

1. Кратчайший путь между i, j не проходит через вершину k, тогда $d_{ij}^k = d_{ij}^{k-1}$.

2. Существует более короткий путь между i, j, проходящий через k, тогда он вначале идет от i до k, а потом от k до j. В этом случае, очевидно, $d_{ij}^k = d_{ij}^{k-1} + d_{kj}^{k-1}$.

Таким образом, для нахождения значения функции достаточно выбрать минимум из двух обозначенных значений.

Тогда рекуррентная формула для d_{ij}^k имеет вид:

d_{ij}^0 – длина ребра (i, j);

$d_{ij}^k = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$.

Алгоритм Флойда-Уоршелла последовательно вычисляет все значения d_{ij}^k, $\forall i, j$ для k от 1 до n. Полученные значения d_{ij}^n являются длинами кратчайших путей между вершинами i, j.

Алгоритм Беллмана-Форда

Предложен независимо Ричардом Беллманом и Лестером Фордом [4]. Алгоритм Беллмана-Форда – алгоритм поиска кратчайшего пути во взвешенном графе. Алгоритм находит кратчайшие пути от одной вершины графа до всех остальных. В отличие от алгоритма Дейкстры, алгоритм Беллмана-Форда допускает ребра с отрицательным весом.

Опишем алгоритм по шагам.

Шаг 0. Пронумеруем все вершины из $G = (A, B)$ так, что $A = \{1, 2, \ldots, p\}$ и при этом номер «1» имеет именно та вершина, из которой будут найдены кратчайшие пути во все остальные вершины. Построим далее матрицу $M = (m_{ij})$, $i, j = 1, 2, \ldots, p$, положив

$$m_{ij} = \begin{cases} f((i, j)), \text{ если } (i, j) \in B \\ \textit{клетка остается незаполненной, если } (i, j) \notin B. \end{cases}$$

Шаг 1. Около первой строки матрицы M (слева от матрицы), поставим числовую пометку «0» и такую же пометку проставим над первым столбцом матрицы. Затем просмотрим помеченную строку слева направо и всякий раз, встречая клетку с числом, прибавим это число к метке строки и сумму проставим над столбцом, в котором эта клетка находится. Затем отразим метки столбцов относительно главной диагонали. Возникнут помеченные строки. С каждой из помеченных строк проделаем то же самое: просмотрим помеченную строку слева направо и всякий раз, встречая клетку с числом, прибавим это число к метке строки и сумму поставим в качестве метки над столбцом, в котором эта клетка стоит. При этом будем соблюдать принцип: «имеющуюся метку не менять». Затем метки столбцов отразим относительно главной диагонали и с помеченными строками вновь проделаем то же самое. И так далее, пока не окажутся помеченными все строки и все столбцы.

Шаг 2. Просмотрим строки таблицы в порядке возрастания их номеров. В каждой строке просматриваются клетки слева направо и всякий раз, когда встречается число, оно складывается с пометкой строки и сумма сравнивается с пометкой столбца, в котором найденное число расположено. Если сумма оказалась меньше, чем пометка столбца, то эта пометка столбца заменяется на упомянутую сумму. Если же сумма оказалась больше или равной пометке, то ничего не меняется. После такого просмотра всех строк новые пометки столбцов отражаются относительно главной диагонали и вся процедура повторяется. И так до тех пор, пока не будут прекращены какие бы то ни было изменения в пометках.

Шаг 3. Теперь по пометкам можно построить кратчайшие пути из первой вершины во все остальные. Фиксируем про-

извольную вершину k ($k = 2, 3, \ldots, p$) и опишем кратчайший путь из первой вершины в вершину k. Во-первых, длина этого кратчайшего пути равна пометке λ_k, стоящей над столбцом номер k. Во-вторых, предпоследняя вершина в кратчайшем пути из первой вершины в вершину k находится так: в столбце номер k отыскиваем число, сумма которого с пометкой строки, в которой оно расположено, равна λ_k. Пусть номер строки, в которой оказалось найденное число, равен l. Тогда предпоследней вершиной в кратчайшем пути из 1 в k будет вершина l. Вершину, которая предшествует вершине l, надо отыскивать как предпоследнюю в кратчайшем пути из 1 в l и так далее.

*Алгоритм A**

Алгоритм A* используется для нахождения маршрута с наименьшей стоимостью от одной вершины (начальной) к другой (конечной), используя алгоритм поиска по первому наилучшему совпадению на графе. В нем применяется оценка узлов, объединяющая в себе $g(n)$ – стоимость достижения данного узла, и $h(n)$ – стоимость прохождения от данного узла до цели:

$$f(n) = g(n) + h(n)$$

Поскольку функция $g(n)$ позволяет определить стоимость пути от начального узла до узла n, а функция $h(n)$ определяет оценку стоимости наименее дорогостоящего пути от узла n до цели, то справедлива следующая формула:

$f(n)$ = оценка стоимости наименее дорогостоящего пути решения, проходящего через узел n.

Таким образом, при осуществлении попытки найти наименее дорогостоящее решение вначале необходимо попытаться проверить узел с наименьшим значением: если эвристическая функция $h(n)$ удовлетворяет некоторым условиям, то поиск A* становится и полным, и оптимальным.

Анализ оптимальности поиска A* является несложным, если этот метод используется в сочетании с алгоритмом Tree-Search. В таком случае поиск A* является оптимальным, при условии, что $h(n)$ представляет собой допустимую эвристическую функцию, т.е. при условии, что никогда не переоценивает стоимость достижения цели. А поскольку $g(n)$ – точная стоимость достижения узла n, из этого непосредственно

следует, что функция $f(n)$ никогда не переоценивает истинную стоимость достижения решения через узел n.

Одно заключительное наблюдение состоит в том, что среди оптимальных алгоритмов такого типа (алгоритмов, которые развертывают пути поиска от корня) поиск A* является оптимально эффективным для любой конкретной эвристической функции. Это означает, что не гарантируется развертывание меньшего количества узлов, чем в поиске A*, с помощью какого-либо иного оптимального алгоритма.

Те соображения, что поиск A*, как один из всех подобных алгоритмов, является действительно полным оптимальным и оптимально эффективным, не означают, что поиск A* может служить ответом на все потребности в поиске. Сложность заключается в том, что при решении большинства задач количество узлов в пределах целевого контура пространства состояний все еще зависит экспоненциально от длины решения. Экспоненциальный рост происходит, если ошибка эвристической функции растет не быстрее, по сравнению с логарифмом фактической стоимости пути. В математических обозначениях условие субэкспоненциального роста состоит в следующем:

$$\left|h(n) - h^*(n)\right| \le o\left(\log h^*(n)\right),$$

где $h(n)$ – истинная стоимость достижения цели из узла n. Почти для всех практически применяемых эвристических функций эта ошибка, по меньшей мере, пропорциональна стоимости пути, и происходящий в связи с этим экспоненциальный рост в конечном итоге превосходит возможности любого компьютера. По этой причине на практике стремление находить оптимальное решение часто не оправдано. Иногда вместо этого целесообразно использовать варианты поиска A*, позволяющие быстро находить неоптимальные решения, а в других случаях – разрабатывать эвристические функции, которые являются более точными, но не строго допустимыми.

Алгоритм Джонсона

Алгоритм Джонсона находит кратчайшие пути для всех пар вершин за время $O(V^2 \log V + VE)$ и основан на идее изменения весов (*reweighting*). Если веса ребер графа неотрицательны, то можно найти кратчайшие пути между всеми парами вершин, применив алгоритм Дейкстры к каждой

вершине. При этом данный алгоритм либо возвращает матрицу весов кратчайших путей, либо сообщает, что в графе имеется цикл отрицательного веса. Если же в графе имеются ребра с отрицательным весом, то можно попытаться свести задачу к случаю неотрицательных весов, заменив весовую функцию ω на новую функцию $\hat{\omega}$. При этом должны выполняться следующие свойства.

а) Кратчайшие пути не изменились: для любой пары вершин $u, v \in V$ кратчайший путь из u в v с точки зрения весовой функции ω является также кратчайшим путем с точки зрения $\hat{\omega}$ и наоборот.

б) Все новые веса $\hat{\omega}(u, v)$ неотрицательны.

Отсюда следует, что новая весовая функция $\hat{\omega}$ обладает такими же свойствами и может быть построена за время $O(VE)$ [6].

Алгоритм Габова

Рассмотрим взвешенный ориентированный граф $G = (V, E)$, в котором веса ребер являются целыми неотрицательными числами, не превосходящими W. Определим, как можно найти кратчайшие пути из одной вершины за время $O(E \log W)$.

Пусть $k = [\log(W + 1)]$ – количество битов в двоичном представлении числа W. Для $i = 1, 2, \ldots, k$ положим $\omega_i(u, v) = [\omega_i(u, v) / 2^{k-i}]$ (иными словами, $\omega_i(u, v)$ получается из $\omega(u, v)$ отбрасыванием $k - i$ младших битов в двоичном представлении числа $\omega(u, v)$). Например, если $k = 5$ и $\omega(u, v) = 25 = \langle 11001 \rangle$, то из $\omega_3(u, v) = \langle 110 \rangle = 6$. В частности, ω_1 принимает только значения 0 и 1, определяемые старшим разрядом, а $\omega_k = \omega$.

Пусть $\delta(u, v)$ – вес кратчайшего пути из u в v относительно весовой функции ω_i (в частности, $\delta_k(u, v) = \delta(u, v)$). Алгоритм находит сначала все $\delta_1(s, v)$ (s – исходная вершина), а затем все $\delta_2(u, v)$, и так далее, пока не дойдет до $\delta_k(s, v) = \delta(s, v)$. Далее, мы полагаем, что $|E| \geq |V| - 1$; стоимость нахождения δ_i при известном δ_{i-1} есть $O(E)$, так что алгоритм будет работать за время, равное $O(kE) = O(E \log W)$.

Такой план решения задачи – замена исходных данных их двоичными приближениями с последовательными уточнением – называется масштабированием (*scaling*) [6].

Алгоритм Карпа

Пусть $G = (V, E)$ – ориентированный граф с весовой функцией $\omega: E \to R$, и пусть $n = |V|$. Средним весом (*mean weight*) цикла $c = \langle e_1, e_2, \ldots, e_k \rangle$, где e_j – ребра графа, назовем число

$$\mu(c) = \frac{1}{k} \sum_{i=1}^{k} \omega(e_i).$$

Пусть $\mu^* = \min \mu(c)$, где c пробегает все (ориентированные) циклы. Не ограничивая общности (поскольку можно добавить дополнительную начальную вершину), будем считать, что каждая вершина $\upsilon \in V$ достижима из некоторой вершины s. Через $\delta(s, \upsilon)$ обозначим вес кратчайшего пути из s в υ; пусть $\delta_k(s, \upsilon)$ – вес кратчайшего пути из s в υ, состоящего в точности из k ребер (если такого пути нет, полагаем $\delta_k(s, \upsilon) = \infty$) [6].

2. Модифицированный алгоритм Дейкстры

Рассмотрим более подробно модифицированный алгоритм Дейкстры как один возможных алгоритмов поиска оптимального пути при комбинированной метрике, объединяющей в себя такие метрики, как пропускная способность канала связи, вероятность потерь пакетов, задержка в передаче пакетов и вариации задержки при минимальной стоимости передачи единицы информации. Модификация алгоритма подробно описана в [7] и заключается в отбрасывании в процессе поиска тех путей, на которых не выполняются ограничения требований (ограничений) по качеству обслуживания и новом способе описания и вычисления меток узлов.

Пусть путь p – множество ребер, соединяющих узел источника и узел получателя от источника s к получателю t [7]. В качестве *QoS* параметров каждого канала связи (ребра графа) будем рассматривать полосу пропускания Y_e, задержку D_e, вариации задержки J_e и вероятность потери пакетов Z_e. Для каждого из возможных путей из s в t будут справедливы следующие соотношения:

$$Y_{s,t} = \max_{e \in p} \{Y_e\};\ D_{s,t} = \sum_{e \in p} D_e;\ J_{s,t} = \sum_{e \in p} J_e;\ Z_{s,t} = \prod_{e \in p} Z_e. \qquad (1)$$

Ограничения на метрики, обеспечивающие требования заданного качества обслуживания, можно представить в следующем виде [7]:

$$Y_{s,t} - Y^{\min} \geq 0;\ D^{\max} - D_{s,t} \geq 0;\ J^{\max} - J_{s,t} \geq 0;\ X_{s,t} - X^{\min} \geq 0. \quad (2)$$

$$X_e = \ln(1 - Z_e) \quad (3)$$

В исходном алгоритме Дейкстры метки j-го узла, которого можно достичь из узла s через соседний узел i, имеют вид $[R_{s,j}, i]$, где величина стоимости $R_{s,j}$, соответствующая данному пути, аддитивна и вычисляется по формуле $R_{s,j} = R_{s,i} + R_{i,j}$, где $R_{i,j}$ – стоимость ребра e_{ij}, а величина $R_{s,i}$ берется из метки i-го узла. Вместо этого, введем метку, имеющую 6 компонентов: $[R_{s,j}, Y_{s,j}, D_{s,j}, J_{s,j}, X_{s,j}, i]$. Новую метку при переходе из узла i в j будем вычислять следующим образом:

$$D_{s,j} = D_{s,i} + D_{i,j} \quad (4)$$

$$J_{s,j} = J_{s,i} + J_{i,j} \quad (5)$$

$$X_{s,j} = X_{s,i} + X_{i,j} \quad (6)$$

$$Y_{s,j} = min\ \{ Y_{s,i},\ Y_{i,j}\} \quad (7)$$

Комбинированная метрика представлена в виде следующей свертки:

$$r = -w_Y \frac{Y_{s,j}}{Y^{\min}} + w_D \frac{D_{s,j}}{D^{\max}} + w_J \frac{J_{s,j}}{J^{\max}} + w_X \frac{X_{s,j}}{X^{\min}}, \quad (8)$$

$$R_{s,j} = \begin{cases} r, \text{ если для } D_{s,j}, J_{s,j}, X_{s,j}, Y_{s,j} \text{ выполняются условия (2)} \\ \infty, \text{ если для } D_{s,j}, J_{s,j}, X_{s,j}, Y_{s,j} \text{ не выполняется хотя бы одно из условий (2)} \end{cases} . \quad (9)$$

Блок-схема алгоритма представлена на рис. 1. Более подробное описание алгоритма и назначение его блоков представлены в работе [7].

Программа для нахождения кратчайшего пути написана на языке С++. Для ее реализации использовались принципы объектно-ориентированного программирования: абстракция, инкапсуляция, наследование, полиморфизм, использование объектов и классов.

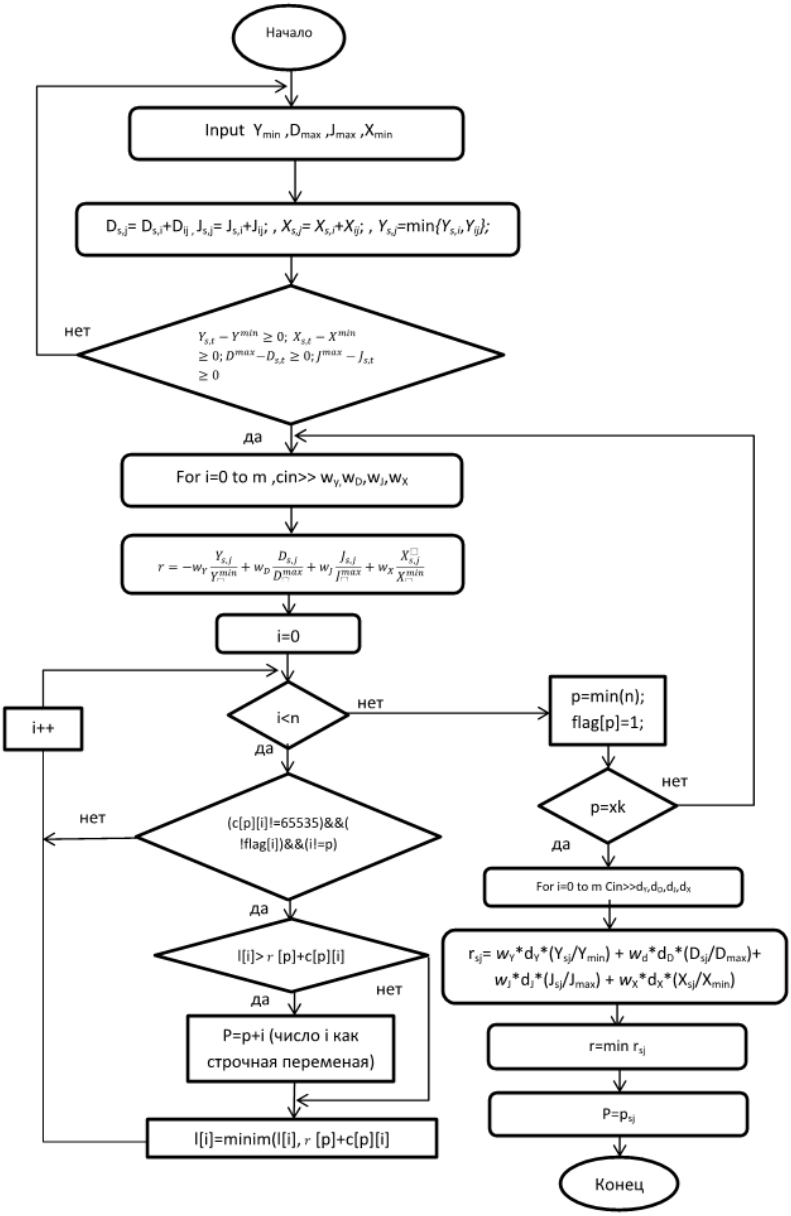

Рис. 1. Блок-схема алгоритма поиска оптимального пути

В качестве абстракции был использован класс весовых и стоимостных коэффициентов (weightsFactors и costFactors).

Пример инкапсуляции – это использование данных для нахождения пути: узел начала и конца, R-параметр, путь и др. Данные были сокрыты от ошибочного или специального изменения, так как это может повлиять на работу программы.

Для нахождения кратчайшего пути были созданы следующие специальные классы.

boundaryConditions (граничные условия) – класс, содержащий граничные условия, которые необходимы для отсеивания путей, не удовлетворяющих граничным условиям.

CostFactors (стоимостной коэффициент) – класс, содержащий стоимостные коэффициенты, необходимые для расчета пути с учетом стоимостных коэффициентов. Основные методы – это ввод и хранение данных.

edge (ребра) – класс, содержащий данные, входящие в ребро: задержка, пропускная способность, вероятность потери пакетов, вариация задержки, и r-параметр, в который рассчитываются все используемые параметры. Важным параметром является булевская переменная, которая сигнализирует о прохождении пути через узел. Основные методы – это ввод и хранение данных и установка флага прохождения пути.

host (узел) – класс, содержащий данные, входящие в ребро, которые суммируются по пути прохождения: задержка, пропускная способность, вероятность потери пакетов, вариация задержки, и r-параметр, в который рассчитываются все используемые параметры. Важным параметром является булевская переменная, которая сигнализирует о прохождении пути через узел. Номер узла, необходимый для нумерации пути, хотя как таковыми номера узлов используются лишь для нумерации найденного пути. Основные методы – это хранение данных и установка флага прохождения пути.

way (путь) – класс, содержащий номер начального и конечного узла, начальное значение r-параметра и его промежуточные значения при нахождения пути, номер узла в процессе нахождения пути. Основные методы – это хранение данных, сохранение пути, его очистка при следующем нахождении.

weightsFactors (весовой коэффициент) – класс, содержащий весовые коэффициенты, которые необходимы для расчета пути с учетом стоимостных коэффициентов. Основные методы – ввод и хранение данных.

Основные функции программы – это нахождение кратчайшего пути. Расчет происходит по двум модифицированным алгоритмам. Для хранения данных используются массивы, так как это наиболее удобный и быстрый способ хранения и использования большого массива данных. Максимальный размер графа не ограничен, но в качестве бесконечности используется 2^{16}, это соответствует типу double в языке C++. Сложность расчета увеличивается с количеством узлов, для каждого алгоритма расчет производится по своей формуле.

Диаграмма введенных классов представлена на рисунке 2. Проведенные расчеты показали, что модификации алгоритма не дают серьезного ухудшения в скорости и сложности расчета кратчайшего пути.

Заключение

Разработка программного обеспечения на основе модифицированного алгоритма Дейкстры основана на замене весового параметра сложной весовой функцией и учете граничных QoS-параметров; предполагает выбор языка программирования и подходящей для его реализации среды и позволяет использовать преимущества программной платформы Microsoft.NET, линейки Visual Studio Express и других ресурсов.

Разработанный с учетом отмеченных условий модифицированный алгоритм поиска на графе (на основе алгоритма Дейкстры) обладает преимуществом автоматического поиска оптимального пути посредством координированного учета всего массива различных параметров других мультимедийных сервисов.

Будучи интегрированным в систему мультисервысных сетей нового поколения, разработанное программное обеспечение позволяет увеличить скорости сетевого пользования и оптимизировать многокритериальный поиск оптимального пути с учетом требований заданного качества обслуживания.

Область применения разработанного программного обеспечения включает в себя такие сферы как создание экспертных систем, разработка баз данных, систем навигации и слежения, картографических сервисов, оптимальных протоколов маршрутизации информационных потоков.

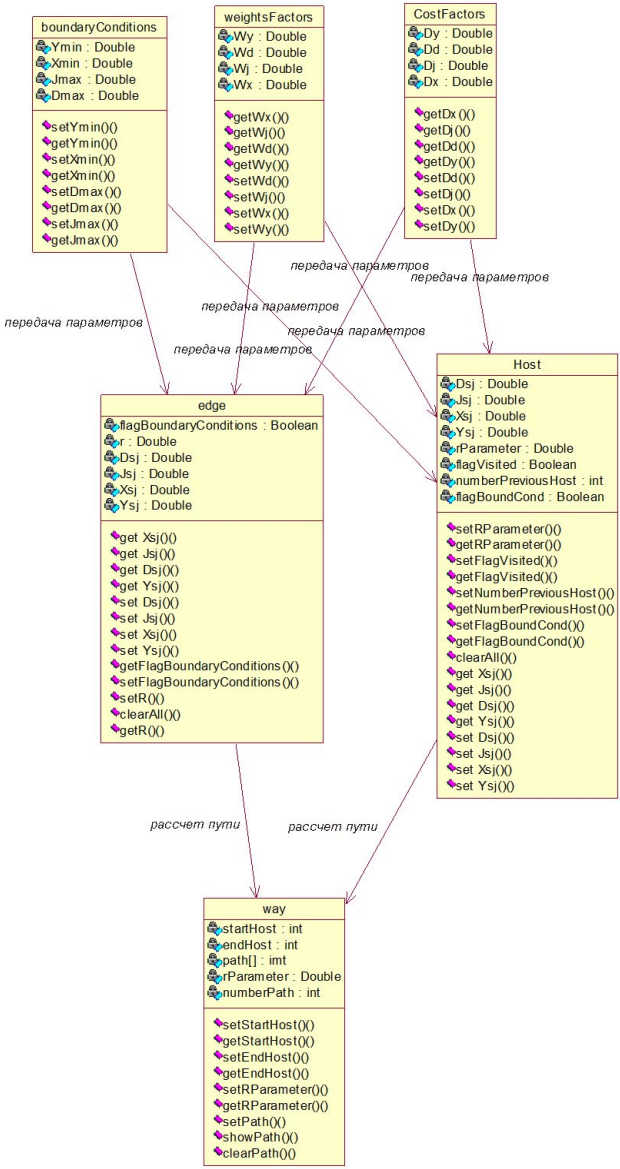

Рис. 2. Диаграмма классов модифицированного алгоритма Дейкстры

Литература

1. E.W. Dijkstra. A note on two problems in connexion with graphs. // Numerische Mathematik. V. 1 (1959), P.269-271.

2. Левитин А.В. // Алгоритмы: введение в разработку и анализ = Introduction to The Design and Analysis of Aigorithms. – М.: Вильямс, 2006. – С.189-195, С.349-353.

3. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн Алгоритмы: построение и анализ = Introduction to Algorithms. – 2-е изд. – М.: Вильямс, 2006. – С.1296.

4. [Электронный ресурс]. – Режим доступа: http://comp-science.narod.ru/KPG/BelmanFord.htm.

5. Рассел С., Норвиг П. Искусственный интеллект: современный подход (AIMA) = Artificial Intelligence: A Modern Approach (AIMA). – 2-е изд. – М.: Вильямс, 2007. – 1424 с.

6. Томас Х. Кормен и др. Алгоритмы: построение и анализ. – 1-е изд. – М.: МЦНМО, 2004. – 523 с.

7. Н.И. Листопад, Ю.И. Воротницкий, А.А. Хайдер // Оптимальная маршрутизация в мультисервисных сетях телекоммуникаций на основе модифицированного алгоритма Дейкстры. // Вестник БГУ, серия 1. – 2015. – № 1. – С.70-76.

N.I. Listopad, I.A. Karuk, A.A. Hayder

Algorithms for Searching the Shortest Path and Its Modification

The article presents an overview of the main trends and methods of searching the shortest path information transmission in the telecommunications networks. The basic algorithms and their modifications are described. Special attention is given to the modified Dijkstra's algorithm that takes into account QoS requirements. The modified algorithm and its UML diagram are presented.

Статья поступила 02.05.2016