Московский государственный университет имени М.В. Ломоносова
Факультет Вычислительной математики и кибернетики
Кафедра Алгоритмических Языков

Курсовая работа

Модель навигации в транспортных сетях

Выполнил:
студент 325 группы
Бондарь Артём Александрович

Научный руководитель:
к.ф.-м.н. доцент, учёный секретарь кафедры
Абрамов Владимир Геннадьевич

Москва — 2018

Содержание

1 Введение

1.1 Актуальность задачи

В связи с растущим интересом к динамическому управлению транспортных систем, возникает необходимость находить кратчайшие пути в большом графе (например, дорожной сети), где вес или задержки, связанные с ребрами, динамически изменяются со временем. Появляются транспортные системы, которые могут обеспечивать в режиме реального времени данные о ситуации на дорогах. (например, VICS [1] или TMC [2]). Совместно с доступностью топологий дорожных сетей и информации об движении на них, собранной за значительный промежуток времени, становится возможным использовать эти данные для предоставления лучших услуг по навигации в транспортных системах, учитывая закономерности движения внутри них.

1.2 Предметная область

Сферой исследования являются алгоритмы поиска крайтчайшего пути на графах, имитирующих траспортные системы. Из чего естественным образом можно вывести следующие свойства, присущие этим графам:

- Ориентированность
- Неотрицательный вес рёбер
- Изменение топологии графа в течении времени
- Изменение весов рёбер графа в течении времени
- Изменения могут быть как случайными, так и запланированными

Таким образом, пользуясь классификацией, предложенной в обзоре существующих алгоритмов поиска кратчайшего пути [3], выделим те типы алгоритмов, область примения которых ближе всего подходит к графам с вышеперечисленными свойствами. (см. рис. 1)

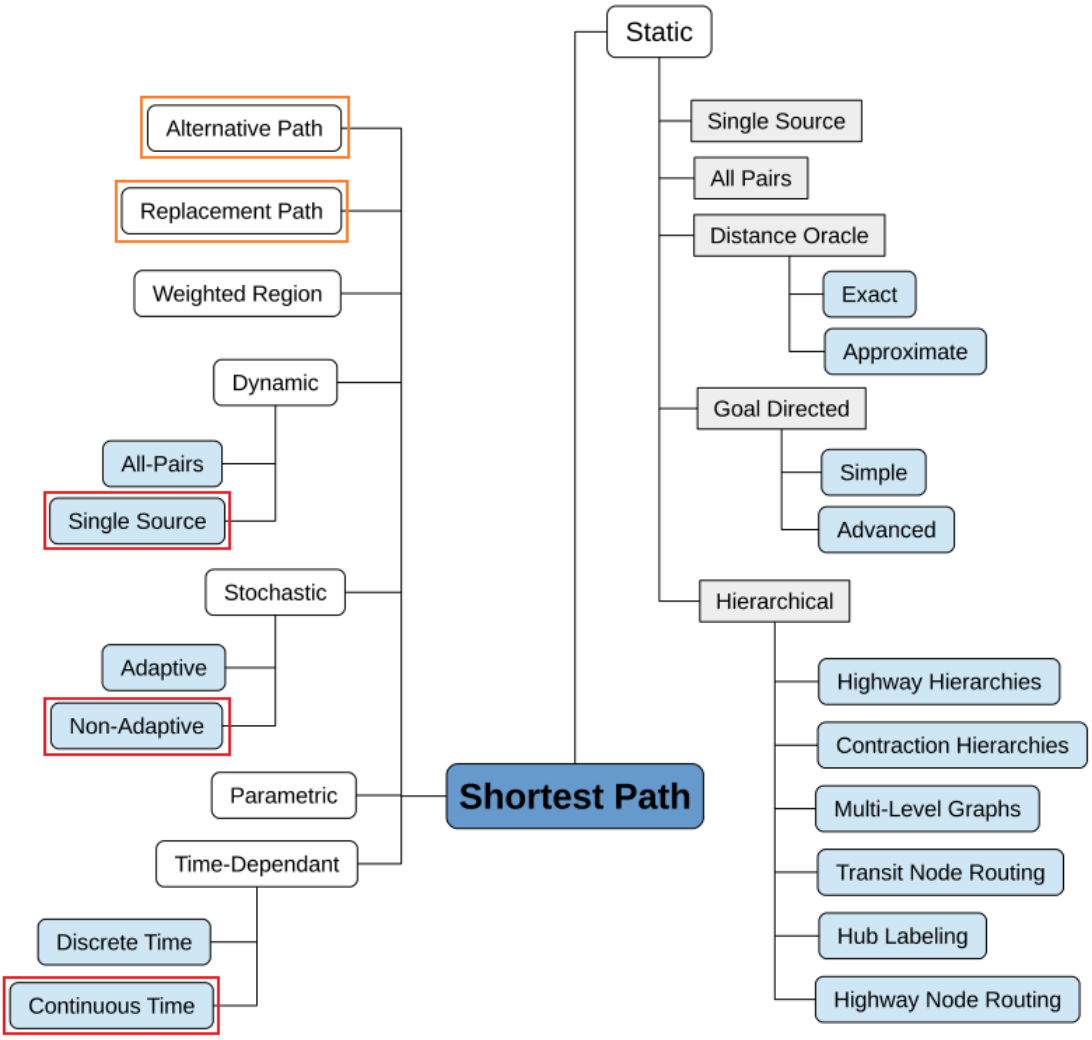

Рис. 1

То есть, при поиске кратчайшего пути в дорожной сети мы гарантированно столкнёмся со следующими задачами:

1. Поиск кратчайшего пути из одного источника в динамических графах.

   Пусть задан граф $G = (V, E)$ с весами ребёр $f(e)$ и выделенной вершиной (называемой *источником*) u. При этом множество вершин графа фиксировано, а множество ребёр может изменятся. Т.е. ребра могут произвольно быть удалены и/или добавлены. Последовательность $P(u, v)$ рёбер $e_1 = (u, w_1), e_2 = (w_1, w_2), \ldots, e_k = (w_{k-1}, v)$ называется *путём*, идущим от вершины u к вершине v. Суммарный вес этого пути равен

   $$f(P(u, v)) = \sum_{i=1}^{k} f(e_i)$$

   Требуется для каждой вершины v, достижимой из вершины-источника u, указать путь $P^*(u, v)$, имеющий наименьший суммарный вес:

   $$f^*(v) = f(P^*(u, v)) = \min f(P(u, v))$$

2. Поиск кратчайшего пути в стохастических графах в условиях неопределённости.

Граф называется *стохастическим*, если веса его рёбр представлены случайными величинами. *Условиями неопределённости* следует считать состояние, при котором распределение случайных величин неизвестно.

3. Поиск кратчайшего пути в графах с весами, задаваемыми функциями от непрерывного времени

   Считая временной отрезок множеством T, пусть $T \in \mathbb{R}$ и $f : E \times T \to \mathbb{R}$. Тогда:

$$\lim_{t \to t_0} f(e, t) = f(e, t_0)$$

Также, с большой вероятностью потребуется решать ещё две проблемы:

4. Поиск кратчайшего альтернативного пути.

   Для пути $P(u, v)$, состоящем из рёбер $e_1, e_2, \ldots, e_k$ *альтернативным* называется такой путь $P'(u, v)$, который не включает в себя ни одного ребра из множества рёбер $E' = \{e_1, ..., e_k\}$.

5. Поиск кратчайшего заменяющего пути.

   Путь $P'(u, v)$ называется *заменяющим* для пути $P(u, v)$, если не содержит ни одного ребра из множества E'', где $E'' \subseteq E', |E''| \neq 0$

Однако, это не означает, что задача строго укладывается в указанные рамки. Вполне возможно успешное применение наработок и с других областей, например, использование иерархических стуктур и/или каких-нибудь более эффективных форматов хранения и обработки информации о графах (или промежуточных вычислений).

1.3 Сфера применения

Наиболее востребованными алгоритмы поиска кратчайшего пути являются при использование в средствах навигации. Перечислим возможности, при наличии которых значительно возрастает эффективность практического использования:

- Получение информации об изменениях в графе с различных источников в реальном времени
- Использование распределённых ресурсов (в том числе удалённых)
- Проведение предварительных вычислений там, где это возможно
- Поскольку все ещё не всегда и/или везде возможно иметь подключения к сети, требуется возможность работы автономно на устройствах с относительно малой вычислительной мощностью
- Быстрое просчитывание хотя бы начала пути, чтобы не тратить лишнее потенциальное время в дороге на ожидание результатов
- Просчитывание дальнейшего маршрута, поиск оптимальных альтернативных путей во время самого движения, а не только до него

Из перечисленных специфик следует почти полное отсутствие ограничений по памяти и сложности для предварительных вычислений и относительно не высокие требования к средней продолжительности вычислений, что позволяет частично уйти от вопросов оптимизации и сосредоточиться на точности полученных результатов.

2 Обзор существующих решений

2.1 Базовые алгоритмы

Классической тройкой алгоритмов, на базе которых строятся множество других различных алгоритмов поиска кратчайшего пути, являются:

2.1.1 Алгоритм Дейкстры

Дан взвешенный ориентированный граф $G(V, E)$ без дуг отрицательного веса. Алгоритм находит кратчайший путь от некоторой вершины s графа G до всех остальных вершин этого графа.

Algorithm 1 Dijkstra's algorithm

```
 1: for each vertex v ∈ V_G
 2:     dist[v] ← ∞
 3:     parent[v] ← NIL
 4: dist[s] ← 0
 5:
 6: Q ← V_G
 7: while Q ≠ ∅
 8:     u ← Extract-Min(Q)
 9:     for each edge e = (u, v)
10:         if dist[v] > dist[u] + weight[e]
11:             dist[v] ← dist[u] + weight[e]
12:             parent[v] ← u
13:
14: H ← (V_G, ∅)
15: for each vertex v ∈ V_G, v ≠ s
16:     E_H ← E_H ∪ {(parent[v], v)}
17: return H, dist
```

На стадии инициализации каждой вершине, кроме начальной s, присваивается метка бесконечности и нулевой путь, начальной же вершине полагается метка 0. (строки 1-4)

Пока все вершины не были пройдены, из ещё не посещённых вершин выбирается вершина u, имеющая минимальную метку. Для каждого соседа вершины u, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки u и длины ребра, соединяющего u с этим соседом. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину u как посещённую и повторим шаг алгоритма. (строки 6-12)

Далее остаётся только собрать полученные пути и вернуть результат. (строки 14-18)

2.1.2 Алгоритм Беллмана-Форда

Дан ориентированный или неориентированный граф G со взвешенными рёбрам без отрицательных циклов. Требуется найти кратчайшие пути от выделенной вершины s до всех вершин графа. Цикл, сумма весов рёбер которого отрицательна, называется *отрицательным циклом.*

Используется те же метки, что и в алгоритме Дейкстры, инициализация аналогична. Разница в том, что вместо обхода вершин рассматриваются все возможные пути из s, длина которых не более i рёбер.

Algorithm 2 Bellman–Ford algorithm

```
1: for v ∈ V
2:     dist[v] ← +∞
3:     parent[v] ← NIL
4: dist[s] ← 0
5:
6: for i ← 1 to |V| − 1
7:     for (u, v) ∈ E
8:         if dist[v] > dist[u] + weight(u, v)
9:             dist[v] ← dist[u] + weight(u, v)
10:            parent[v] ← u
11: return dist
```

2.1.3 A*

Дан взвешенный граф G. Алгоритм ищет маршрут наименьшей стоимости от выбранной начальной вершины s до выбранной конечной d. В процессе работы для вершин рассчитывается функция $f(v) = g(v) + h(v)$, где

- $g(v)$ — наименьшая стоимость пути в v из стартовой вершины
- $h(v)$ — эвристическое приближение стоимости пути от v до конечной цели

Algorithm 3 A* search algorithm

```
1: open ← s
2: close = ⊘
3: g[s] = 0
4: f[s] = g[s] + h[s]
5:
6: while close ≠ 0
7:     x = min_{f(x)} ∈ open
8:     if x = d
9:         return true
10:    open → x
11:    close ← x
12:    for v ∈ neighbours(x)
13:        cost = g[x] + weight(x, v)
14:        if v ∈ close and cost g[v]
15:            continue
16:        if v ∉ close or cost < g[v]
17:            parent[v] = x
18:            g[v] = cost
19:            f[v] = g[v] + h[v]
20:            if v ∉ open
21:                open ← v
22: return false
```

Обозначения:

- open — множество вершин, которые требуется рассмотреть

- close — множество рассмотренных вершин
- $f[x]$ — значение эвристической функции "расстояние + стоимость"для вершины x
- $g[x]$ — стоимость пути от начальной вершины до x
- $h(x)$ — эвристическая оценка расстояния от вершины x до конечной вершины.

На каждом этапе работы алгоритма из множества open выбирается вершина с наименьшим значением эвристической функции и просматриваются её соседи. Для каждого из соседей обновлятся расстояние, значение эвристическо функции и он добавляется в множество open. Поведение алгоритма сильно зависит от того, какая эвристика используется. В свою очередь, выбор эвристики зависит от постановки задачи.

2.1.4 Сложность алгоритмов

Худшую вычислительную сложность имеет алгоритм Беллмана-Форда - $O(|V||E|)$, зато у него самое широкое применение. Немного лучше сложность у алгоритма Дейкстры - $O(|V|^2)$.
Временная сложность алгоритма A* зависит от эвристики. В худшем случае, число вершин, исследуемых алгоритмом, растёт экспоненциально по сравнению с длиной оптимального пути, но сложность становится полиномиальной, когда эвристика удовлетворяет следующему условию:

$$|h(x) - h^*(x)| \leq O(\log h^*(x))$$

, где h^* — оптимальная эвристика, то есть точная оценка расстояния из вершины x к цели.

Таким образом, в своем изначальном варианте эти алгоритмы плохо применимы для больших динамических транспортных графов с переменными значениями рёбер. Первые два - в силу большой вычислительной сложности, а последний - в виду сложности выбора эвристики приближенной к оптимальной.

2.2 Модификации базовых алгоритмов

2.2.1 2S-LTT

Двухшаговый алгоритм [4], основанный на алгоритме Дейкстры, для графов большой размерности с весами рёбер, задаваемыми функцией от времени. При этом также начальным параметром является временной интервал - промежуток времени, в который можно выехать из пункта отправления.

Algorithm 4 Two-Step-LTT$(G_T(V, E, W), v_s, v_e, T)$

Input: a time-dependent graph G_T, a query LTT(v_s, v_e, T) -
source v_s, destination v_e, and starting-time interval $T = [t_s, t_e]$;
Output: optimal $v_s - v_e$ path p^*, and optimal starting time t^*.

1: $\{g_i(t)\} \leftarrow timeRefinement(G_t, v_s, v_e, T)$;
2: **if** $\neg(g_e(t) = \infty$ for the entire $[t_s, t_e])$ **then**
3: $t^* \leftarrow \text{argmin}_{t \in T}\{g_e(t) - t\}$;
4: $p^* \leftarrow pathSelection(G_T, \{g_i(t)\}, v_s, v_e, t^*)$;
5: **return** (t^*, p^*);
6: **else return** $\emptyset$

Вычисления проходят в две фазы, на первом шаге вычисляется самое быстрое время прибытия для каждого ребра, на втором происходит выбор пути. Фактически, от алгоритма Дейкстры 2S-LTT отличается только первым шагом, поскольку выбор пути происходит аналогично:

Algorithm 5 *pathSelection*$(G_T(V, E, W), \{g_i(t)\}, v_s, v_e, t^*)$;

Input: a time-dependent graph G_T, the set of earliest arrival-time functions $g_i(t)$ for all nodes $v_i \in V$, source node v_s, destination node v_e, and the optimal starting time t^*;
Output: an optimal $v_s - v_e$ path p^* for starting time t^*.

```
1: v_j ← v_e;
2: p* ← ∅;
3: while v_j ≠ v_s do
4:     for each (v_i, v_j) ∈ E do
5:         if g_i(t*) + w_{i,j}(g_i(t*)) = g_j(t*) then
6:             v_j ← v_i; break;
7:     p* ← (v_i, v_j) · p*
8: return p*
```

Алгоритм ищет наименьшее время прибытия с условием возможности ожидания на вершинах. Полученная вычислительная сложность: $O((|V| \log |V| + |E|)\alpha(T))$, где $\alpha(T)$ - стоимость применимых функций.

2.2.2 OR

Относительно простым алгоритмом является обобщение алгоритма Беллмана-Форда - OR [5] (назван по первым буквам инициалов авторов).

Algorithm 6 Bellman-Ford Based Algorithm

```
1: for all v_t ∈ V do g_t(t) ← inf for t ∈ T;
2: for all (v_k, v_l) ∈ E do h_{k,l}(t) ← inf for t ∈ T;
3: g_s(t) ← t for t ∈ T;
4: repeat
5:     for all (v_k, v_l) ∈ E do h_{k,l}(t) ← g_k(t) + w_{k,l}(g_k(t));
6:     for all v_l ← V do g_l(t) ← min_{v_k ∈ N(v_l)}{h_{k,l}(t)};
7: until all functions g_l(t) are unchanged
8: return (t* ← argmin_{t∈T}{g_e(t) − t}, p*);
```

Пусть $g_l(t)$ - время кратчайшего прибытия в вершину v_l из вершины v_s за начальное время t и пусть $h_{k,l}(t)$ - кратчайшее время прибытия в v_l из v_s за время t через ребро (v_k, v_l).
Сначала инициализируются функции $g_l(t)$ и $h_{k,l}(t)$ (строки 1-3), а потом происходит повторяющиеся обновления $g_l(t)$ и $h_{k,l}(t)$ до тех пор, пока они не сойдутся до правильных значений (строки 4-7).
Наконец (строка 8), происходит возврат лучшего начального времени t^* и оптимального пути p^*, который высчитывается на основе $g_l(t)$ и $h_{k,l}(t)$.

Временная сложность алгоритма - $O(|V||E|\alpha(T))$, что делает его не применимым для больших временных графов.

2.2.3 KDXZ

Данный алгоритм [6], тоже названный по первым буквам инициалов авторов, является расширением A* алгоритма на случай графов с весами, зависящими от времени. Главная идея схожа -

происходит обработка очереди Q всех возможных путей. Пускай p_k - это путь от начальной вершины v_s к вершине v_k в графе G_T. Возможно существование нескольких путей из v_s в v_k и все из них будут содержаться в Q одновременно. Для каждого пути v_k высчитывается функция $f_{p_k}(t) = g_{p_k}(t) + d_{k,e} - t$, где:

- $g_{p_k}(t)$ - время прибытия из v_s в v_k начиная с времени t
- $d_{k,e}$ - эвристическая функция, представляющая собой нижнюю оценку предполагаемой продолжительности пути из v_k в конечную вершину v_e
- $f_{p_k}(t$ - оценочное время пути из v_s в v_e по пути p_k с начального момента времени t

На каждой итерации, выбирается путь $p_i \in Q$, такой, что $\min_t\{f_{p_i}(t)\} = \min_{p_k \in Q} {}_t\{f_{p_k}(t)\}\ \forall k$
Каждый путь p_j, получаемый увеличением путя p_i на одно ребро (v_i, v_j) добавляется в очередь Q, а путь p_i удаляется. Процесс остановится, когда будет получен первый путь $v_s - v_e$.

Алгоритм KDXZ эффективен только тогда, когда выбранная эвристика эффективно сокращает ширину поиска, а v_s и v_e относительно близко расположены друг к другу. К тому же выбор эвристики $d_{k,e}$ для обобщённых графов достаточно сложен. В худших случаях, при удалённых на значительное расстояние начальной и конечной вершинах и/или выборе не оптимальной эвристики получаемая сложность алгоритма экспоненциальна.

2.2.4 Сравнение алгоритмов

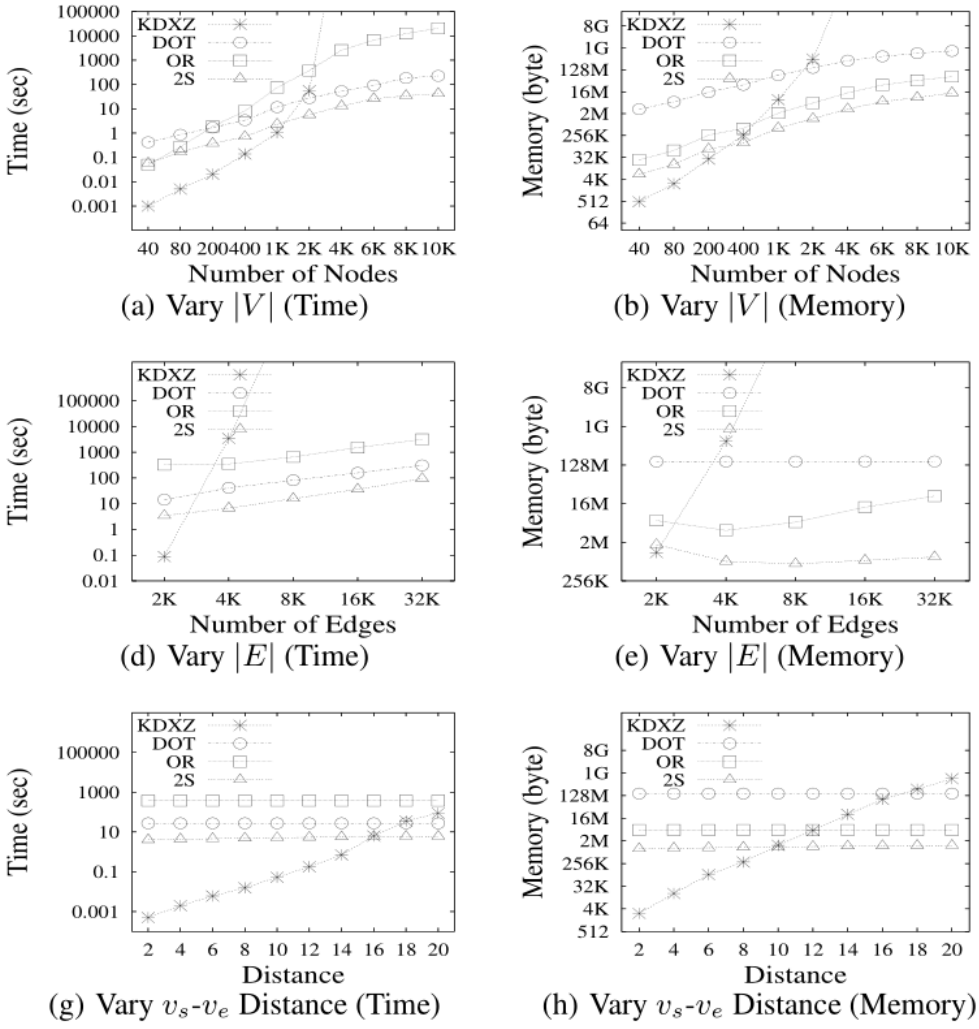

Рис. 2

На Рис. 2 представлены результаты работы трёх предыдущих алгоритмов и ещё одного четвёртого - DOT [7], который не был рассмотрен в виду того, что не попадает в исследуемую сферу алгоритмов (поскольку используется для графов с дискретным временем, а не непрерывным).

Тестирование проводилось на 2.8GHz CPU с 1G RAM. В качестве основы для данных использовалась дорожная сеть штата Мэрилэнд в США, полученная системой TIGER/Line [8]. Количество вершин - 16,326, рёбер - 26,528. Весы вершин были сгенерированы в случайном диапазоне.

2.3 Выводы

Используя за основу классические алгоритмы, можно добиться довольно неплохих результатов при их расширении на временные графы, по крайней мере небольшого размера. При должной степени оптимизации вычислений, а также используя более продвинутые методики хранения и работы с данными, применяя различные подходы из других областей алгоритмов кратчайших путей (например, используя иерархические структуры) можно получить более эффективные реализации алгоритмов.

Однако, глобальная проблема состоит в том, что существует зазор между теоретическим моделированием и реальной ситуацией. В транспортных системах происходит большое количество случайных событий (аварии, перекрытия дорог, снижение/увеличение плотности движения), которые нельзя заранее спрогнозировать (начиная от самого факта появления события и заканчивая конкретной величиной изменений). Из-за этого сильно падает точность вычислений, появляется необходимость учитывать динамику и распределение переменных величин.

Помимо этого, заранее вычисленные маршруты движения могут перестать быть минимальными по выбранному параметру вследствии поведения пользователя. Почти все теоретические алгоритмы принимают за аксиому, что будет произведено движение с определённой скоростью по определённому маршруту. Но в реальной жизни могут произойти отклонения, связанные с человеческим фактором, например, водитель может намеренно свернуть с оптимального пути по какой-то причине, может немного задержаться или просто снизить скорость движения.

Проанализировав особенности транспортных систем и сферы использования можно прийти к выводу, что ни один из существующих алгоритмов в полной мере не удовлетворяет накладываемым ограничениям и требованиям к результатам в связи с тем, что эти ограничения и требования могут произвольно менятся. Следовательно, в связи с отсутствием теоретических решений, придётся применить экспериментальные прикладные методы.

3 Постановка задачи

3.1 Параметры графа

Задан ориентированный граф $G_T = (V, E)$, $V = \{v_i\}$ - множество вершин графа, $E \subseteq V \times V$ - множество рёбер, T - область времени.

- t_0 - начальное время
- t_{cur} - текущее время
- s - вершина, с которой началось движение
- d - вершина, по направлению к которой происходит движение
- $l(e)$ - длина ребра, не меняется
- $A(e,t)$ - доступность ребра

$$A(e,t) = \begin{cases} 1, & \text{если ребро } e(u,v) \text{ существует и доступно для прохождения} \\ 0, & \text{если ребро } e(u,v) \text{ не существует или запрещено для прохождения} \end{cases}$$

- T_{act} - интервал времени, на протяжении которого полученные данные считаются актуальными
- $v_{avg}(e,t)$ - средняя скорость движения на ребре $e(u,v)$ в момент времени $t = t_0$
- $v_{rec}(e)$ - средняя скорость движения на ребре $e(u,v)$ для последнего обновления
- $t_{rec}(e)$ - время последнего обновления средней скорости на ребре $e(u,v)$, инициализируется так:

$$\begin{cases} v_{rec}(e) = v_{avg}(e, t_0) \\ t_{rec}(e) = t_0 \end{cases}$$

- $v_{act}(e,t_0)$ - скорость движения на ребре $e(u,v)$, применяемая для вычислений в момент времени $t_0 > t_{cur}$

$$v_{act} = \begin{cases} v_{rec}, & t_{cur} - t_0 \leq T_{act} \\ v_{avg}, & t_{cur} - t_0 > T_{act} \end{cases}$$

- $\frac{l(e)}{v_{act}(e,t_0)}$ - вычисляемый вес ребра $e(u,v)$ в момент времени t_0

Последовательность $P(s,d)$ рёбер $e_1 = (s, v_1), e_2 = (v_1, v_2), \ldots, e_k = (v_{k-1}, d)$ называется *путём*, идущим от вершины s к вершине d, причём $A(e_i, t_i) = 1 \ \forall i$, где t_i - момент попадания в начало ребра e_i.

Изменение времени задается рекурсивно:

$$t_i = t_{i-1} + \frac{l(e_i)}{v_{act}(e_i, t_i)}$$

Суммарная длительность пути равна:

$$L_t(P(u,v)) = \sum_{i=1}^{k} \frac{l(e_i)}{v_{act}(e_i, t_i)}$$

Требуется для двух указанных вершин s и d привести такой путь $P(s,d)$, чтобы суммарная длительность были минимальными.

3.2 Данные для приёма и передачи

В любой момент времени t_{cur} могут поступить следующие данные:
$\{u, v, v_{rec}\}$ - обновление средней скорости на ребре $e(u,v)$, при получении $t_{rec}(e) = t_{cur}$
$\{u, v, A(e(u,v), t), t_0, t_1\}$ - доступность ребра $e(u,v)$ в интервале времени $[t_0, t_1]$

Для вычислений передаются: $\{G_T, t_0, A(e_i, t), T_{act}, s, d, v_{avg}(e_i, t), v_{rec}(e_i), t_{rec}(e_i)\}$ - слепок всех данных, используемых для вычислений кратчайшего пути, в момент времени t_0
Получаемые результаты вычислений: $\{t_0, P, L_t\}$

4 Модель навигации

4.1 Предпосылки

В силу быстро растущей сложности просчитывания кратчайшего пути для динамического графа со случайными изменениями - это не является осуществимой в удобоваримый срок задачей даже для графов средней величины. Из-за случайного изменения параметров графа и непредсказуемого

поведения *выполняющего* алгоритм (например, водителя) становится бессмысленным просчитывание оптимального маршрута только единожды до выезда из отправной точки. Более того, просчёт второй половины пути может не пригодится, если что-то случится раньше того, как *выполняющий* доберётся туда.

Таким образом, необходимо отслеживать в реальном времени все те события, которые могут поменять уже вычисленный результат и при их возникновении проводить перерасчёт кратчайшего пути. Для решения поставленной задачи предлагается реализация системы по управлению вычислениями, позволяющей производить:

- Версионное хранение информации о состоянии графа - требуется для анализа полученных результатов, вычисленных на основе данных о уже возможно устаревшем графе
- Обработку поступающих изменений графа - правильно разбирать и сохранять данные об изменении топологии и средних скоростей на рёбрах, определять которые из них могут повлиять на корректность уже полученного результата
- Асинхронное оперирование различным количеством вычислительных узлов - организация приёма и передачи данных, эффективного использования ресурсов
- Анализ поступающих результатов вычислений - требуется определять по полученным результатам вносимые в текущий маршрут изменения, если таковые необходимы для уменьшения времени маршрута
- Корректирование работы в связи с изменениями параметров движения, зависящих от выбора выполняющего алгоритм - отслеживать в реальном времени, насколько точно происходит следование маршрута и в случае критических изменений срочно предпринимать меры по просчёту нового пути

В виду высокой сложности данной системы, она выходит за рамки курсовой работы. Поэтому в данной работе будет подробно рассмотрен только её отдельный компонент - сам алгоритм вычисления кратчайшего пути. Вообще говоря, сама система вычислений полностью независима от используемых в ней алгоритмов поиска кратчайшего пути, она оперирует только с передачей им параметров и возвращаемыми результатами, однако поскольку не существует подходящего алгоритма для работы с графами, предполагаемыми в задаче, все равно придётся реализовывать свою версию такого алгоритма.

4.2 Идея алгоритма

Рассматривая проблему глобально, причина роста сложности напрямую зависит от количества вершин и ребёр в графе. С маленькими графами успешно может справится любой алгоритм за приемлимое время работы. Поскольку для нашей задачи подсчёт сразу всего пути бессмысленный, логично разбить весь путь на небольшие участки, на которых можно быстро подсчитывать кратчайший путь. Также, следует как можно сильнее ограничить уже разбитые участки, так чтобы не проводить лишних вычислений маршрутов через те зоны графа, которые очевидно лежат в стороне или вообще в противоположном направлении от вершины конца. Таким образом, рассматривая только небольшой подграф на нём можно добиться быстрого вычисления кратчайшего пути до его границы. Следует заметить, что этот же самый подход используется в алгоритмах, проводящих вычисления для графов с иерархией.

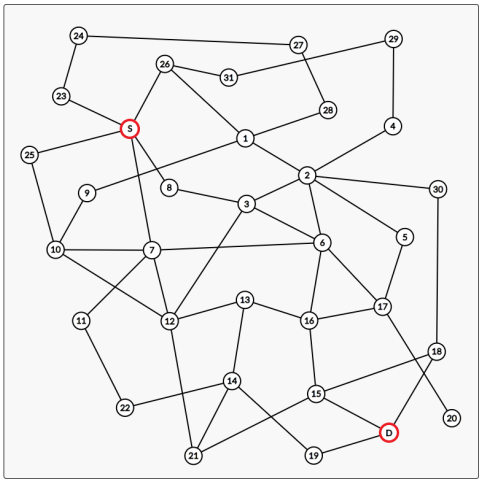

Рис. 3а

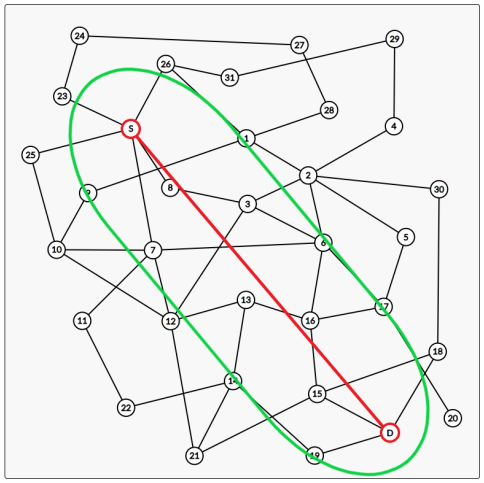

Рис. 3б

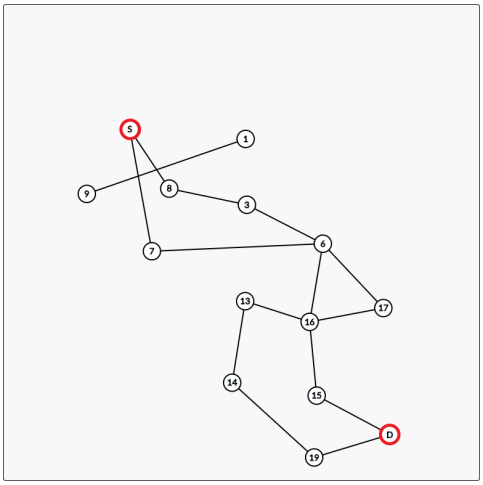

Рис. 3в

Очевидной идеей для достижения пункта назначения является предпринятие движения в сторону этого самого пункта. Попробуем применить её для редуцирования графа. Допустим, у нас имеется некая дорожная сеть, граф которой изображён на Рис. 3а и нам необходимо попасть из вершины s в вершину d. Проведём прямую между этими вершинами и выделим вокруг неё некоторую окрестность фиксированого радиуса r от каждой точки прямой (в результате получится овальная область, как на Рис. 3б). Далее убираем все вершины, не попавшие в эту область и все рёбра, выходящие из них (результат показан на Рис. 3в). В итоге получим подграф с меньшим количеством вершин и рёбер. Помимо этого также можно убрать и все вершины, не состоящие в одном компоненте связности с вершинами s и d.

При достаточно большой длине r большая часть кратчайших путей может попадать в область редуцирования. Рассмотрим возникающие при этом проблемы.

1. Существует некий окружной путь, не попавший в область редуцирования, однако при этом являющийся кратчайшим путём. Например, это может быть некая магистраль с повышенной скоростью движения (см. Рис. 4а)

2. В области редуцирования отсутсвует путь из s в d и для его получения возможно придётся слишком сильно расширять эту область. (см. Рис. 4б)

3. В области редуцирования находится подграф, не имеющий связи с вершиной концом, вследствии чего попадание в этот подграф будет напрасной тратой времени. (см. Рис. 4в)

При этом отсуствие самого лучшего пути не является сильно критическим фактором, а вот остальные две проблемы делают данный подход не дееспособным. Поэтому необходимо как-нибудь иначе выбирать исходное направление к концу маршрута. Тут на помощь приходят обычные алгоритмы поиска кратчайшего пути. Рассмотрим следующую эвристическую гипотезу:

Гипотеза. Кратчайший путь *по времени* находится недалеко от кратчайшего пути *по расстоянию*.

Действительно, примем во внимание сильно разветлённые топологии городов, как самых часто используемых для навигации. Какие факты из этого следуют?

- Наличие большого количества объездных дорог
- Очень маловероятен случай ухудшения движения в целом районе, чтобы его нельзя было объехать по соседним улицам

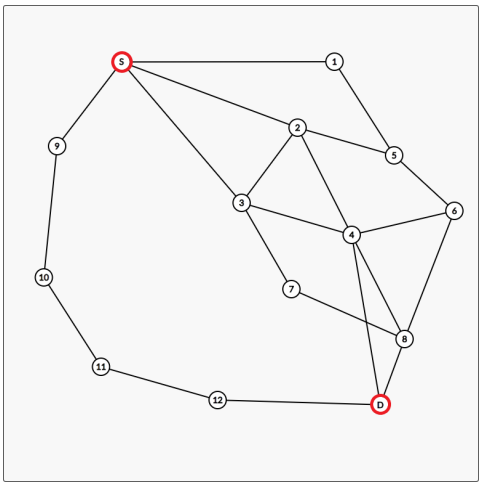

Рис. 4а

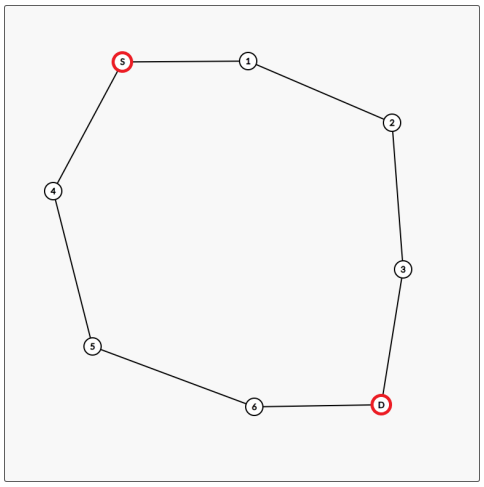

Рис. 4б

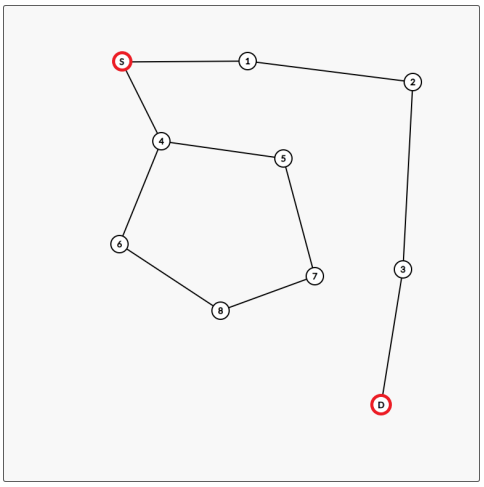

Рис. 4в

- Скоростные дороги имеют периодические заезды и выезды

Приведённые факты позволяют опустить проблему 1), поскольку если более короткий путь и будет пропущен, то в дальнейшем он сможет быть найден при вычислениях на больших областях. Проблемы 2) и 3) решаются автоматически. Решим последнюю существенную проблему, которая может возникнут при использовании такого способа выбора направления, а именно - проблема *узких мест*. Рассмотрим топологию, изображённую на Рис. 5.

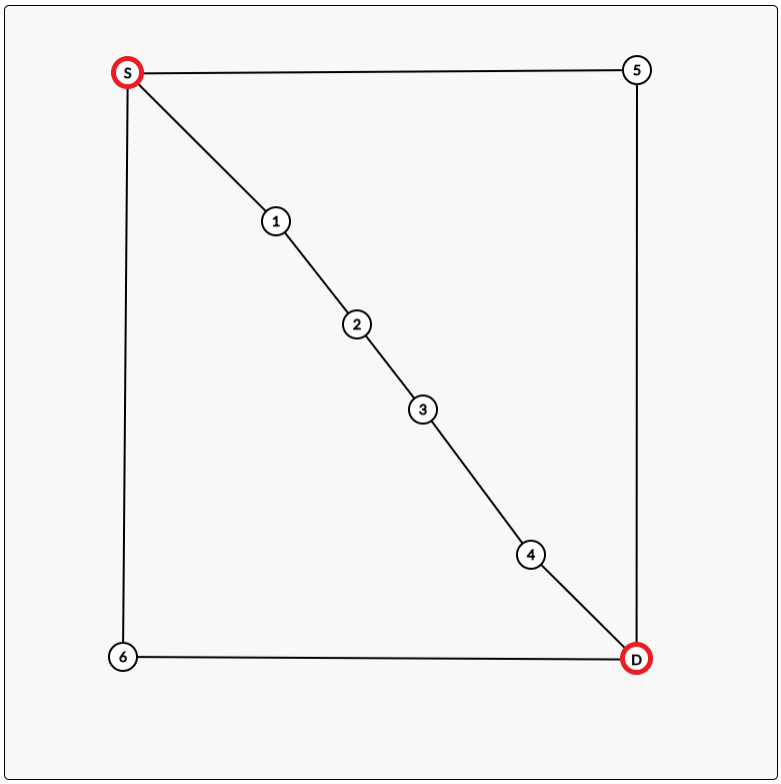

Рис. 5

Очевидно, кратчайший путь - это $S-1-2-3-4-D$. Припустим, что исполняющий алгоритм достиг вершины 2 и в этот момент на ребре $2-3$ что-то случается, его перекрывают или там возникает беспросветная „пробка“, а короткого объезда нет, только возврат и движение по путям $S-5-D$ или $S-6-D$. Помимо таких сценариев, это ещё и частный случай проблемы 1), когда пути $S-5-D$ или $S-6-D$ могут быть заранее более выгодными, чем самый короткий по расстоянию.

Однако, можно предпринять определённые шаги, которые помогут избежать проблемы таких „узких“ мест. Достаточно убрать такое ребро из графа и высчитать альтернативный путь. Не составит труда подсчитать его *временную* сложность и сравнить с исходной сложность самого короткого пути *по расстоянию*, и выбрать среди них лучший.

Последний не решённый вопрос - как именно определять такие проблемные рёбра? Ничего сложного тут тоже нет, поскольку можно заранее найти все „узкие“ места во всём графе G_T. Перебираем каждое ребро $e(u, v) \in E$ по очереди. Для каждой пары вершин u и v удаляем из графа ребро $e(u, v)$ и считаем наименьшее расстояние от вершины u до вершины v. Если оно больше наперёд заданной константы L_{narrow}, тогда объезд этого ребра слишком длинный и это ребро является „узким“местом, помечаем его и переходим к оставшимся ещё не рассмотреным рёбрам. Таким образом, получим множество всех потенциально проблемных рёбер V_{narrow}.

4.3 Описание

Алгоритм поиска кратчайшего пути *по времени* в окрестности кратчайшего пути *по расстоянию*:
Шаг 1. Высчитываем кратчайший путь $P'(s, d)$ *по расстоянию*. Получаем для него $L_t(P'(s, d))$
Шаг 2. Для каждого ребра $v_i \in V_{narrow} : v_i \in P$ высчитываем *альтернативный* кратчайший путь *по расстоянию* $P''_i(s, d)$ и его временную стоимость $L_t(P''_i(s, d))$. Выбираем такой путь $P(s, d)$, что $L_t(P(s, d)) = \min_t\{L_t(P'(s, d)), L_t(P''_i(s, d))\}\ \forall i$
Шаг 3. Для наперёд заданной константы T_{max} разбиваем путь $P(s, d)$ на n подпутей $P(u_i, v_i)$, так что $P(s, d) = \bigcup_{i=1}^{n} P(u_i, v_i)$, $L_t(P(u_i, v_i)) \leq T_{max}$ и n минимально.
Шаг 4. Для каждого подпути $P(u_i, v_i)$ высчитываем самый короткий путь $P^*(u_i, v_i)$ *по времени* в *редуцированном* подграфе $G_{T,i} \subseteq G_T$, содержащем $P(u_i, v_i)$. $G_{T,i}$ получается из графа G_T взятием всех тех его вершин, находящихся на расстоянии меньше R_{max} от любой точки на подпути $P(u_i, v_i)$.
Шаг 5. Объединяем полученные решения и получаем результат: $P(s, d) = \bigcup_{i=1}^{n} P^*(u_i, v_i)$,

5 Реализация

В качестве алгоритма поиска кратчайшего пути *по расстоянию* используется алгоритм Дейкстры в виду простоты реализации и небольшого размера графа. Для больших же графов предполагается к использованию намного более эффективный алгоритм NBA* [9] с 4-5 кратным приростом скорости.

В качестве алгоритма поиска кратчайшего пути *по времени* используется модифицированный алгоритм Дейкстры, который от оригинала отличается тем, что в качестве меток для вершин берётся время, затрачиваемое на путь к ним от начальной вершины. Также, функция подсчёта веса ребра принимает ещё и временной параметр, на основании которого определяется скорость движения на ребре:

$$weight(e, t) = l(e)/v_{act}(e, t)$$

Algorithm 7 Modified Dijkstra's algorithm

for each vertex $v \in V_G$
    $cost[v] \leftarrow \infty$
    $parent[v] \leftarrow NIL$
$cost[s] \leftarrow 0$

$Q \leftarrow V_G$
while $Q \neq \emptyset$
    $u \leftarrow$ Extract-Min(Q)
    for each edge $e = (u, v)$
        if $cost[v] > cost[u] + weight(e, t_0 + cost[u])$
            $cost[v] \leftarrow cost[u] + weight(e, t_0 + cost[u])$
            $parent[v] \leftarrow u$

$H \leftarrow (V_G, \emptyset)$
for each vertex $v \in V_G,\ v \neq s$
    $E_H \leftarrow E_H \cup \{(parent[v], v)\}$
return $H, cost$

6 Результаты

Была сделана программная реализация предложенного алгоритма (и сопутствующих ему компонентов), позволяющая:

- визуализировать заданный граф, весы рёбер и пути на нём
- находить множество рёбер V_{narrow}
- редуцировать граф до окрестности заданного пути
- вычислять кратчайшие пути *по расстоянию* и *по времени*

В качестве графа для тестирования использовался сгенерированный граф из 100 вершин и 167 рёбер. Весы рёбер, представляющие среднюю скорость движения на ребре, в основном задавались случайно сгенерированными величинами (некоторым рёбрам скорость движения была выставлена вручную для имитации „пробок“ в „узких“ местах графа).

Несколько характерных примеров работы алгоритма показаны на Рис. 6а и Рис. 6б. Жёлтым изображены кратчайшие пути *по расстоянию*, зелёным - *по времени*. В первом случае алгоритм находит более быстрые альтернативы для участков пути. Во втором же предлагает полностью другой обходной путь из-за низкой скорости движения на „мосту“ (ребре в центре пути, который соединяет два компонента связности, и является „узким“ местом для графа).

Тестируемый граф в виду своих небольших размеров не позволил найти ответа на вопрос выбора оптимального параметра R_{max}, поскольку даже при выборе $R_{max} = \infty$ заметной разницы в скорости работы алгоритма не было замечено. Следовательно, единственное, что можно уверенно заявить - следует выбирать такой R_{max}, который в среднем при редукции графа в области пути продолжительностью меньше T_{max} давал подграф $G = (V, E)$ размерности $N(G) = |V| + |E|$, где N не меньше, чем у заданного графа ($N(G_{test}) = 267$).

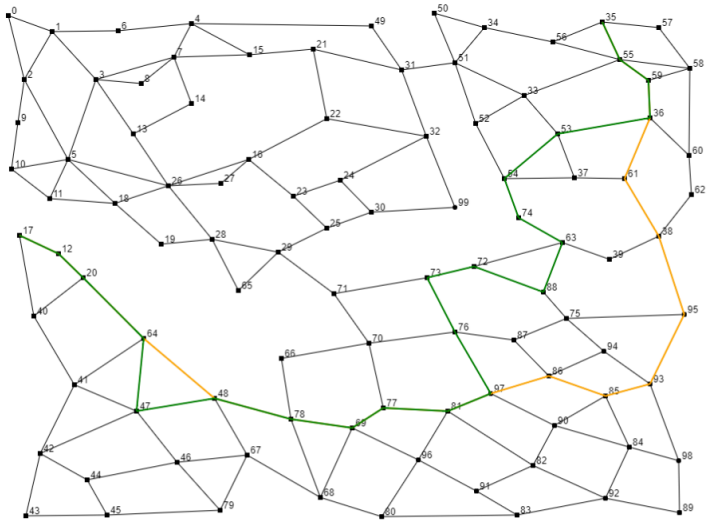

Рис. 6а

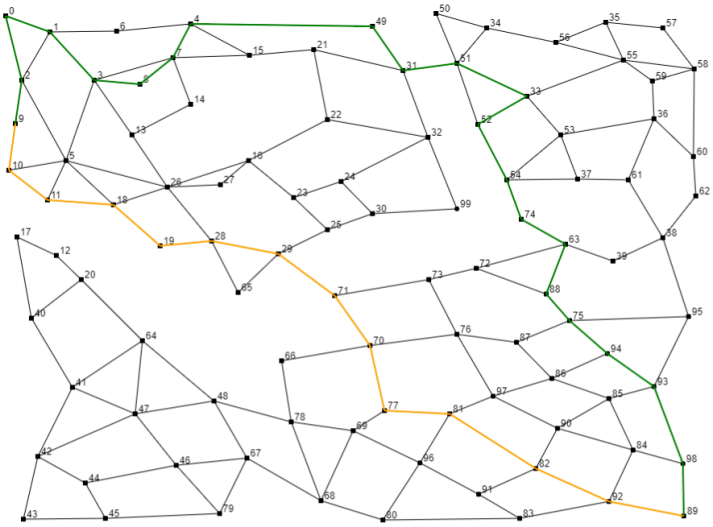

Рис. 6б

Список литературы

[1] Vehicle Information and Communication System. http://www.vics.or.jp/en/

[2] Traffic Message Channel. https://en.wikipedia.org/wiki/Traffic_message_channel

[3] Amgad Madkour, Walid G. Aref, Faizan Ur Rehman, Mohamed Abdur Rahman, Saleh Basalamah. A Survey of Shortest-Path Algorithms. Purdue University, West Lafayette, USA. Umm Al-Qura University, Makkah, KSA. May 8, 2017

[4] B. Ding, J. X. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. Proceedings of the 11th international conference on Extending database technology Advances in database technology EDBT 08, page 205, 2008.

[5] A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. J. ACM, 37(3):607–625, 1990.

[6] E. Kanoulas, Y. Du, T. Xia, and D. Zhang. Finding fastest paths on a road network with speed patterns. In ICDE, pages 10–19, 2006.

[7] I. Chabini. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. Transportation Research Record, 1645:170–175, 1998.

[8] Topologically Integrated Geographic Encoding and Referencing system: http://www.census.gov/geo/www/tiger/

[9] Wim Pijls, Henk Post. Yet another bidirectional algorithm for shortest paths. Econometric Institute Report EI 2009-10. 15 June, 2009