«به نام خداوند بخشنده و مهربان»

دانشگاه صنعتی امیرکبیر

(پلی‌تکنیک تهران)

دانشکده کامپیوتر

گزارش پروژه درس ارزیابی کارایی

بررسی کنترل ترافیک در شبکه‌های شهری به کمک زنجیره‌های مارکف

استاد درس:

جناب آقای دکتر صبایی

با راهنمایی:

جناب آقای دکتر حجازی

نگارنده:

الهه امامی

تاریخ:

مرداد ۱۴۰۱

فهرست

فهرست شکل‌ها

با توجه به اهمیت کنترل ترافیک و ازدحام در شبکه‌های خیابانی شهرها، تحقیقات بسیاری در این خصوص در جریان است.

یکی از مهم‌ترین فناوری‌های فراهم‌ساز برای کنترل ترافک، مدل‌سازی درست ترافیک جهت کنترل و پیش‌بینی می‌باشد که امکان مدیریت پیش‌گیرانه، در مقابل واکنشی، را فراهم می‌آورد.

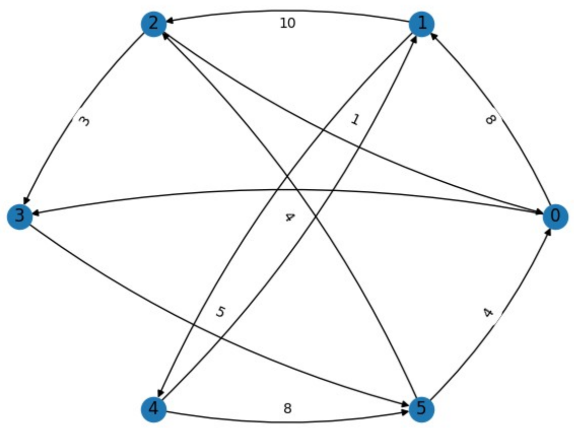
زنجیره‌های مارکف می‌توانند جهت این مدل‌سازی به ما کمک کنند و اطلاعاتی را در اختیار ما بگذارند که در مدل‌های دیگر به سختی قابل دسترس است. به عنوان مثال به دست آوردن خیابان‌ها یا تقاطع‌های حیاتی که هرگونه اشکال در آن‌ها منجر به ایجاد ترافیک شدید می‌شود، توسط احتمالات حالت پایدار به سادگی قابل دستیابی می‌باشد.

می‌دانیم زنجیره‌های مارکف توسط ماتریس احتمالات گذر یک‌گامه قابل توصیف می‌باشد.

۱ مدل‌سازی

در صورتی که از روی نقشه به شبکه‌ی شهری نگاه کنیم، تقاطع‌ها نمایانگر گره‌های گراف و خیابان‌ها یال‌ها هستند. در صورتی که در هر تقاطع، تعداد ماشین‌های خروجی به هر یال را شمارش کنیم، ماتریس احتمالات گذر به دست می‌آید.

در شبیه‌سازی انجام شده، گراف زیر را در نظر گرفتیم:



شکل ۱-۱: گراف شهری

که در آن، وزن یال‌ها، طول خیابان‌ها در نظر گرفته شده است.

در طول گزارش، نام گذاری یال‌ها به ترتیب از صفر تا ده در نظر گرفته شد: (از چپ به راست)

[(0, 1), (0, 3), (1, 2), (1, 4), (2, 0), (2, 3), (3, 5), (4, 1), (4, 5), (5, 0), (5, 2)]

سپس، تعدادی ماشین بر روی هر یک از تقاطع‌ها در نظر گرفتیم که هر یک مقصد مشخصی داشت و کوتاه‌ترین مسیر از نظر مسافت را برای هر یک از خودروها محاسبه کردیم که این امر به کمک کد زیر صورت گرفت:

```
def compute_shortest_paths(G, src_dst):
    paths = []
    for i in src_dst:
        paths.append(nx.dijkstra_path(G, source=i['src'], target=i['dst'],
weight='weight'))

    # print(paths)
    return paths
```

سپس، با داشتن مسیری که هر یک از خودروها می‌پیماید، با شمارش خودروهای خروجی از هر تقاطع، به کمک کد زیر ماتریس احتمالات گذر محاسبه گردید:

```
def compute_p(G, paths):
    nodes = list(G.nodes)
    edges = list(G.edges)

    # Count the number of cars exiting each intersection
    p = np.zeros(shape=(len(nodes), len(nodes)))
    for path in paths:
        for k in range(len(path) - 1):
            p[path[k]][path[k + 1]] += 1

    # print(p)
    # each row should sum up to 1
    i = 0
    for t in p:
        row_sum = np.sum(t)
        if row_sum != 0:
            for j in range(len(t)):
                p[i][j] /= row_sum

        i += 1

    return p
```

که نتیجه‌ی آن را می‌توان دید:

```
transition probability matrix:
[[0.         0.55555556 0.         0.44444444 0.         0.        ]
 [0.         0.         0.42857143 0.         0.57142857 0.        ]
 [0.46153846 0.         0.         0.53846154 0.         0.        ]
 [0.         0.         0.         0.         0.         1.        ]
 [0.         0.3        0.         0.         0.         0.7       ]
 [0.61111111 0.         0.38888889 0.         0.         0.        ]]
```

این ماتریس احتمال رفتن از هر تقاطع به تقاطع دیگری که با یال مستقیم به یکدیگر متصل هستند را نشان می‌دهد.

۲ محاسبه‌ی دوگان ماتریس احتمال گذر

ماتریس بالا که محاسبه شد، جهت بررسی تقاطع‌ها مناسب می‌باشد. به عنوان مثال اگر احتمال حالت پایدار آن را محاسبه کنیم، می‌توانیم تقاطع‌های اصلی و پرتراکم را شناسایی کنیم.

این اطلاعات جهت مدیریت شهری مانند زمان‌بندی مدت زمان سبز بودن چراغ‌های راهنمایی رانندگی بسیار مفید می‌باشد.

اما، آنچه ما به دنبال آن هستیم، یافتن خیابان‌های پرتراکم می‌باشد جهت مسیریابی بهینه. به این منظور، طبق مقاله‌ی [۱]، نیاز داریم ماتریس احتمالات دوگان شکل ۱-۱: گراف شهری را محاسبه نماییم. یعنی احتمال رفتن از هر یال به یال دیگر.

این محاسبه به کمک کد زیر صورت پذیرفت:

```
def compute_p_dual(G, paths):
    nodes = list(G.nodes)
    edges = list(G.edges)

    # convert paths from (node to node) to (edge to edge)
    paths_edge_based = convert_to_paths_edge_based(paths)


    num_edges = len(edges)
    p_dual = np.zeros(shape=(num_edges, num_edges))

    #print(edges)
    for i, src_edge in enumerate(edges):
        for j, dst_edge in enumerate(edges):
            for path in paths_edge_based:
                if src_edge in path:
                    if dst_edge in path:
                        if (path.index(src_edge) + 1 == path.index(dst_edge)):
                            p_dual[i][j] += 1


    # each row should sum up to 1
    i = 0
    for t in p_dual:
        row_sum = np.sum(t)
        if row_sum != 0:
            for j in range(len(t)):
                p_dual[i][j] /= row_sum

        i += 1

    #print(p_dual)
    return p_dual
```

به این ترتیب ماتریس احتمالات گذر از هر خیابان به خیابان دیگر محاسبه گردید که نتیجه‌ی آن را در ادامه می‌بینیم:

```
tpm of the dual matrix:
[[0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.5 0.5 0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0. ]
 [1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.6 0.4]
 [0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0. ]
 [1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0. ]]
```

نکته‌ای که در ماتریس فوق نادیده گرفته شده است، ویژگی‌های هر خیابان است مانند طول، حداکثر سرعت مجاز و غیره.

به منظور گنجاندن این پارامترها در ماتریس بالا، مانند مقاله‌ی [۲] عمل کرده و زمان سفر در هر خیابان را به صورت طول آن تقسیم بر سرعت محاسبه می‌کنیم و آن را طوری نرمال می‌کنیم که کوچکترین عدد به دست آمده مقدار یک و دیگر مقادیر بر مبنای این عدد مقدار بگیرند. سپس به کمک رابطه‌ی زیر، ماتریس را تغییر می‌دهیم:

$$p_{ij} = (1 - p_{ii})\, p'_{ij} \quad , \quad p_{ii} = \frac{tt_i - 1}{tt_i}$$

این امر به کمک دو تابع زیر انجام شد:

```python
def compute_normalized_tt(lengths, speed):
    tt = []
    for length in lengths:
        tt.append(length/speed)

    for t in range(len(tt)):
        tt[t] /= min(tt)

    return tt


def compute_modified_tpm(tpm, tt):
    n = len(tt)
    modified_tpm = np.zeros(shape=(n, n))

    for i in range(n):
        modified_tpm[i][i] = ((tt[i] - 1) / tt[i])

    for i in range(n):
        for j in range(n):
            if i != j:
                modified_tpm[i][j] = (1 - modified_tpm[i][i]) * tpm[i][j]

    #print(modified_tpm)
    return modified_tpm
```

و مقدار آن به صورت زیر محاسبه گردید:

```
modified_tpm:
[[0.875      0.         0.         0.125      0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.75       0.         0.         0.         0.
  0.25       0.         0.         0.         0.        ]
 [0.         0.         0.9        0.         0.05       0.05
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.5        0.         0.
  0.         0.         0.5        0.         0.        ]
 [1.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.33333333
  0.66666667 0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.6        0.         0.         0.24       0.16      ]
 [0.         0.         1.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.5        0.5        0.        ]
 [1.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         1.
  0.         0.         0.         0.         0.        ]]
```

۳ محاسبه‌ی احتمالات حالت پایدار

پس از این مرحله، می‌خواهیم احتمالات حالت پایدار را محاسبه کنیم:

```
def near(a, b, rtol = 1e-5, atol = 1e-8):
    return np.abs(a-b)<(atol+rtol*np.abs(b))


def steady_state_prob(p):
    # values, vectors = sp.sparse.linalg.eigs(p, k=1, sigma=1)
    values, vectors = sp.linalg.eig(p, left=True, right=False)
    #print(values)
    vectors = vectors.T
    vector = vectors[near(values, 1)]

    state = (vector/np.sum(vector))[0]
    steady_state = []

    for i,s in enumerate(state):
        steady_state.append(np.round(state[i].real, 6))

    return steady_state
```

در توضیح نحوه‌ی محاسبه‌ی احتمالات حالت پایدار باید گفت، رابطه‌ی احتمالات حالت پایدار می‌دانیم از رابطه‌ی زیر محاسبه می‌گردد:

$$tpm * \pi = \pi$$

از طرفی، از جبر خطی می‌دانیم:

$$A * v = \lambda v$$

که در آن، A ماتریس مختصاتی، v بردار ویژه (Eigen Vector) و لامبدا مقدار ویژه (Eigen Value) می‌باشد. اگر این دو رابطه را همزمان نگاه کنیم متوجه می‌شویم احتمالات حالت پایدار همان بردار ویژه‌ی معادل مقدار ویژه‌ی ۱ ماتریس احتمالات گذر می‌باشد.

به این ترتیب، حالت پایدار به صورت زیر محاسبه گردید:

```
steady_state:
[0.615385, -0.0, -0.0, 0.153846, -0.0, -0.0, -0.0, -0.0, 0.153846, 0.076923, -0.0]
```

نقد مقاله: باید توجه داشت در تعریف احتمال حالت پایدار، مقدار صفر عددی قابل پذیرش نیست (بازگشت‌پذیری و قابلیت دسترسی حالات را زیر سؤال می‌برد)، ولی در محاسبه‌ی کوتاهترین مسیر، این امر غیرطبیعی نیست که برخی از خیابان‌ها به هیچ عنوان در کوتاه‌ترین مسیر انتخاب نشوند. به عنوان مثال به این علت که بسیار طولانی هستند، راه‌های چندگامه‌ی کوتاه‌تر به مقصد یافت می‌شود.

این موضوع در مقالات نادیده گرفته شده بود که می‌تواند به عنوان گامی در بهبود آن‌ها در نظر گرفته شود و روشی برای مقابله با این مساله پیدا شود.

۴ محاسبه‌ی ازدحام در هر خیابان

با داشتن احتمالات حالت پایدار، به کمک رابطه‌ی زیر می‌توانیم ازدحام در هر خیابان را محاسبه کنیم:

$$D_i = \frac{V \pi_i}{L_i N_i}$$

که در آن، D تراکم، V تعداد کل خودروها، π بردار احتمالات حالت پایدار، L طول هر خیابان، و N تعداد لاین‌های هر خیابان می‌باشد.

در توضیح این رابطه می‌توانیم بگوییم که احتمال حالت پایدار، احتمال بودن در هر خیابان را به ما می‌دهد که اگر آن را در تعداد کل ماشین‌ها ضرب کنیم، تعداد ماشین‌های روی هر خیابان به صورت احتمالی به دست می‌آید.

حال باید توجه داشت به عنوان مثال ده ماشین بر روی یک خیابان ده متری، ازدحام بالایی حساب می‌شود ولی همان ده ماشین بر روی یک خیابان ۱۰۰ متری، تراکمی به مراتب کمتر دارد.

همین مثال، در خصوص تعداد لاین‌ها هر خیابان نیز صادق است.

پس، برای محاسبه‌ی ازدحام هر خیابان، تعداد ماشین‌های روی آن را به تعداد لاین و طول آن تقسیم می‌کنیم تا اثر آن‌ها در مدل‌سازی تراکم اعمال شود.

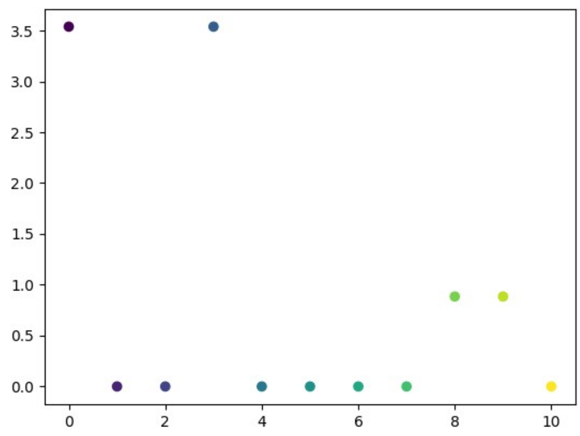
این محاسبات توسط قطعه کد زیر انجام گردید:

```python
def density(num_cars, stedy_state, road_len, num_lines):
    density = []
    n = len(stedy_state)
    for i in range(n):
        density.append((num_cars * stedy_state[i]) / (num_lines * road_len[i]))

    return density
```

و مقادیر آن به این صورت محاسبه شد:

```
density
[3.5384637499999996, -0.0, -0.0, 3.5384580000000003, -0.0, -0.0, -0.0, -0.0, 0.8846145000000001, 0.8846145000000001, -0.0]
```

که نمایش آن به صورت نمودار زیر است:



شکل ۱-۲: تراکم روی هر خیابان

۵ تعریف utility function

حال که تراکم روی هر خیابان را محاسبه کردیم، زمان آن رسیده است تا معیار انتخاب بهترین مسیر را بهبود بخشیم و به جای آنکه تنها طول مسیر را به عنوان معیار مقایسه‌ی مسیرها در نظر بگیریم و کوتاه‌ترین مسیر را پیدا کنیم، می‌توانیم ازدحام مسیر را نیز در آن دخیل کنیم.

انتخاب این تابع، می‌تواند با معیارهای متفاوتی تعریف شود. به عنوان مثال، می‌توانیم برای آن که ازدحام و طول مسیر به یک اندازه به عنوان بهتر بودن مسیر مؤثر باشد، مقادیر محاسبه شده‌ی ازدحام را ابتدا نرمال کنیم به این صورت که بیشترین ازدحام مقدار یک داشته باشد و سپس آن را در طول بزرگترین خیابان ضرب کنیم و مقادیر ازدحام بین صفر و طول بزرگترین خیابان قرار بگیرد و سپس جمع این مقدار و طول مسیر را به عنوان معیار هزینه‌ی هر یال تعریف کنیم.

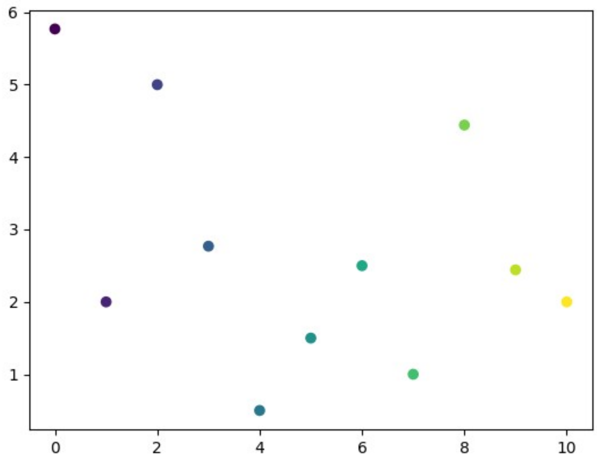
پیاده‌سازی این منطق، به کمک کد زیر انجام شد:

```python
def compute_cost_of_each_edge(length, density):
    cost = []
    normalized_density = []
    max_density = max(density)
    max_len = max(length)
    for i in range(len(density)):
        normalized_density.append((density[i] / max_density) * max_len)

    #print(normalized_density)

    for i in range(len(density)):
        cost.append((length[i] + density[i]) / 2)

    max_cost = max(cost)
    # normalize the cost
    # for i, c in enumerate(cost):
    #     cost[i] /= max_cost

    #print(f'cost: {cost}')
    return cost
```

به عنوان مثال، در شبیه‌سازی با مقادیر بالا، هزینه‌ی یال‌هایی که مبتنی بر کوتاه‌ترین طول، بدون توجه بـه ازدحـام محاسبه شده بود به این صورت به دست آمد:

```
cost of each edge:
[5.769231875, 2.0, 5.0, 2.769229, 0.5, 1.5, 2.5, 1.0, 4.44230725, 2.44230725, 2.0]
```

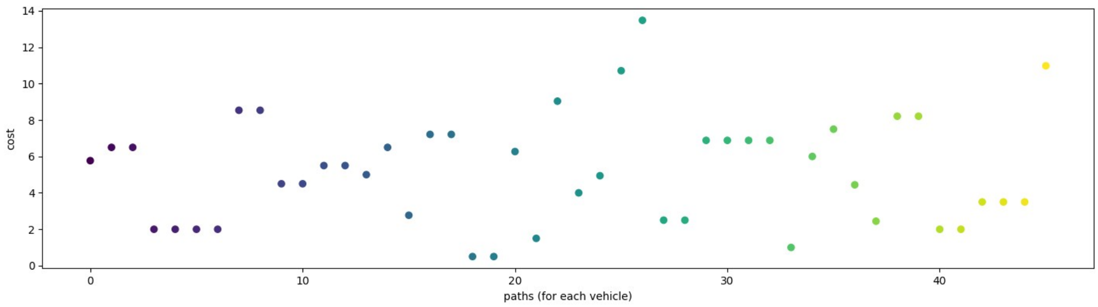
که این مقادیر را در نمودار زیر نیز می‌بینیم.



شکل ۱-۳: هزینه‌ی هر یال در حالت کوتاه‌ترین مسیر از نظر فقط مسافت

حال نگاهی بیندازیم به هزینه‌ی مسیرهایی که هر خودرو پرداخته است:

(با توجه به اینکه تعداد خودروها و به تبع آن تعداد مسیرها زیاد بود، از نوشتن مقادیر آن چشم پوشیده و تنها به آوردن نمودار آن بسنده می‌کنیم.)



شکل ۱-۴: هزینه هر مسیر به ازای هر خودرو با مبدا و مقصد مشخص

مجموع هزینه برای این ۴۶ خودرو که در شبیه‌سازی در نظر گرفته شد، مقدار ۲۳۹.۳۰۷۶ و به عبارتی هر خودرو به طور میانگین هزینه‌ی ۵.۲۰۲۳ پرداخته است.

۶ محاسبه‌ی کوتاه‌ترین مسیر با در نظر گرفتن ازدحام

حال برای آنکه بتوانیم کوتاه‌ترین مسیرها را مجدداً اما با در نظر گرفتن تابع هزینه‌ی جدیدی که تعریف کردیم و ازدحام را نیز در نظر می‌گیرد محاسبه کنیم، ابتدا وزن‌های گراف را تغییر داده به هزینه‌هایی که برای هر یال در شکل ۱-۳: هزینه‌ی هر یال در حالت کوتاه‌ترین مسیر از نظر فقط مسافت محاسبه کرده بودیم.

سپس، مراحل بالا را تکرار می‌کنیم تا این بار کوتاه‌ترین مسیر بر این مبنا محاسبه شود. اما نتیجه‌ی دریافتی به این صورت شد که ماتریس احتمال گذر محاسبه شده، حالت پایدار ندارد!

به نتیجه‌ی ماتریس احتمال گذر آن نگاه کنیم:

```
tpm of the dual matrix:
[[0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   1.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.75 0.25 0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   0.  ]
 [1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   1.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.6  0.4 ]
 [0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  ]
 [0.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.  ]]
```

نقد مقاله: همانطور که دیده می‌شود، با این هزینه‌ی جدید در نظر گرفته شده، هیچ خودرویی خیابان (یال) شماره هشت را در مسیر خود انتخاب نکرده است. این امر سبب شده تا احتمال رفتن از این یال به یال دیگر صفر شود در حالی که در ماتریس گذر باید جمع احتمالات در هر سطر یک شود.

این موضوع سبب می‌شود که امکان محاسبه‌ی احتمالات پایدار وجود نداشته باشد و برنامه قادر به ادامه نباشد.

برای جلوگیری از این موضوع می‌توان مسیر هر ماشین را مطلقاً کوتاه‌ترین مسیر در نظر نگرفت و گاهی احتمالی مسیر دیگری را انتخاب کرد. اما در هر صورت باید در نظر بگیریم این احتمال وجود دارد که احتمال عبور از یال در حالتی که احتمالی عمل نمی‌کنیم و صرفاً کوتاه‌ترین مسیر (با هر معیاری) را در نظر می‌گیریم، اتفاق بیفتد.

این موضوع، در مقالات مورد بررسی دیده نشده است.

این امر می‌تواند به عنوان گام بعدی در تحقیق در نظر گرفته شود و راهکاری برای حالت‌هایی که ماتریس احتمالات گذر حساب شده، حالت پایدار نداشته باشد، تعریف شود.

نقد مقاله: همچنین مورد دیگری که در مقالات نادیده گرفته شده است این است که گاهی اوقات برخی خودروها تنها از یک یال که مستقیماً مبدأ را به مقصد متصل می‌کند عبور می‌کنند. این امر سبب ایجاد ترافیک بر روی یال مورد نظر می‌شود اما در ماتریس احتمال گذر محاسبه نمی‌شود. دلیل آن این است که در ماتریس احتمال گذر، ماشین‌هایی را میشماریم که از یک یال به یال دیگر می‌روند اما ماشین‌هایی که روی یک یال است و از آن یال خارج نمی‌شود شمارش نمی‌شود. (مثل ماشینی که بین دو تقاطع حرکت کند ولی از خیابان به خیابان بعدی نرود. درواقع و مقصدش تنها به اندازه‌ی یک خیابان مستقیم فاصله داشته باشد.)

جهت رفع این مشکل، می‌توانیم هنگام محاسبه‌ی تراکم (density)، این خودروها را نیز به تراکم اضافه کنیم تا تأثیر آن‌ها در مراحل انتخاب مسیر بهینه اعمال شود.

۷ رفع مشکل به وجود آمده

می‌دانیم در دنیای واقعی، احتمال صفر و یک به ندرت اتفاق می‌افتد و لزوماً تمامی ماشین‌ها از مسیر بهینه‌ی محاسبه شده عبور نمی‌کنند و گاهی از مسیر دیگری عبور می‌کنند.

به این منظور که احتمالات به وجود آمده در ماتریس احتمال گذر را حالت احتمالی به آن دهیم و از قطعیت کوتاه‌ترین مسیر بکاهیم، سطر تمام صفر موجود در ماتریس را که هیچ ماشینی از آن عبور نکرده انتخاب کرده و یک ماشین روی آن قرار دادیم.

```
all_zero_row = np.where(~p_dual.any(axis=1))[0]
for i in range(len(all_zero_row)):
    p_dual[all_zero_row[i]][0] = 1
```

این کار می‌توانست بهتر و به صورت احتمالی صورت پذیرد اما در این‌جا صرفاً به منظور تست کد نوشته شده، جهت از بین بردن مشکل عدم امکان محاسبه‌ی احتمال حالت پایدار، این روش را پیش گرفتیم.

اما روش در نظر گرفته شده قطعاً نیاز به بهبود دارد و در اینجا فقط جهت دیدن نتیجه، این راهکار در نظر گرفته شد تا احتمال حالت پایدار قابل محاسبه شود. این‌جا فقط می‌توان از جهت منطق کد گفت، خودرویی به سیستم

افزودیم که از مسیر بهینه، به دلیل اشتباه یا هدف دیگری، عبور نکرده است و قطعیت اینکه تمامی خودروها حتماً از مسیر بهینه بروند را از بین بردیم.

سپس تابع هزینه را به این ترتیب تغییر دادیم که تراکم برای ما اهمیت بیشتری از طول مسیر دارد و وزن بیشتری (۱۰۰ برابر بیشتر) برای تراکم در نظر گرفتیم:

```
def compute_cost_of_each_edge(length, density):
    cost = []
    normalized_density = []
    max_density = max(density)
    max_len = max(length)
    for i in range(len(density)):
        normalized_density.append((density[i] / max_density) * max_len)

    # print(normalized_density)

    for i in range(len(density)):
        cost.append(length[i] + 100 * density[i])
```

و با این تغییرات، کد را مجدداً تا رسیدن به همگرایی هزینه‌ی مسیرهای محاسبه شده ادامه دادیم:

```
edge_costs = main_program(G, src_dst, road_len, speed, num_lines, num_cars)
# print(edge_costs)

while True:
    avg_costs = sum(edge_costs) / len(edge_costs)
    updated_G = G.copy()
    # update the graph with new costs
    i = 0
    for s, d, w in updated_G.edges(data=True):
        w['weight'] = edge_costs[i]
        i += 1

    edge_costs = main_program(updated_G, src_dst, road_len, speed, num_lines,
num_cars)
    new_avg_costs = sum(edge_costs) / len(edge_costs)
    # print(f'avg: {avg_costs}')
    # print(f'new: {new_avg_costs}')
    if np.isclose(avg_costs, new_avg_costs):
        break
```

نتیجه‌ی بهبود هزینه‌ی مسیرها در سه تکرار محاسبه شد که نتایج آن را به ترتیب تکرار می‌بینیم:

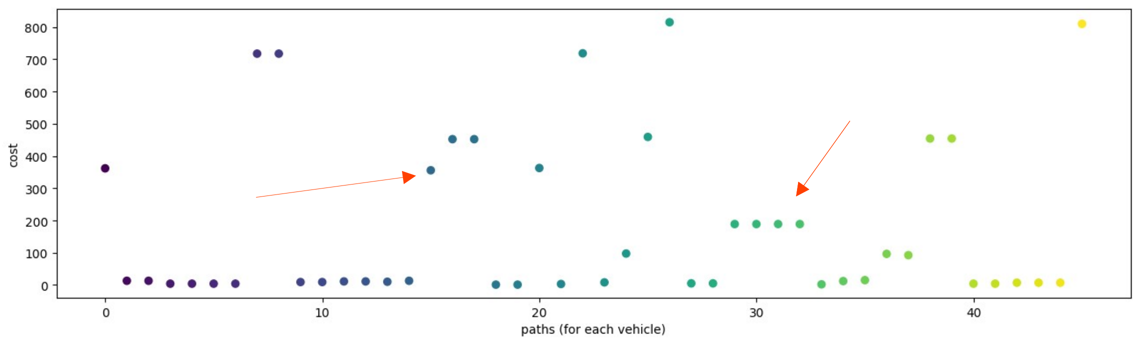
average cost of shortest path: 181.75078804347825

average cost of shortest path: 169.5217391304348

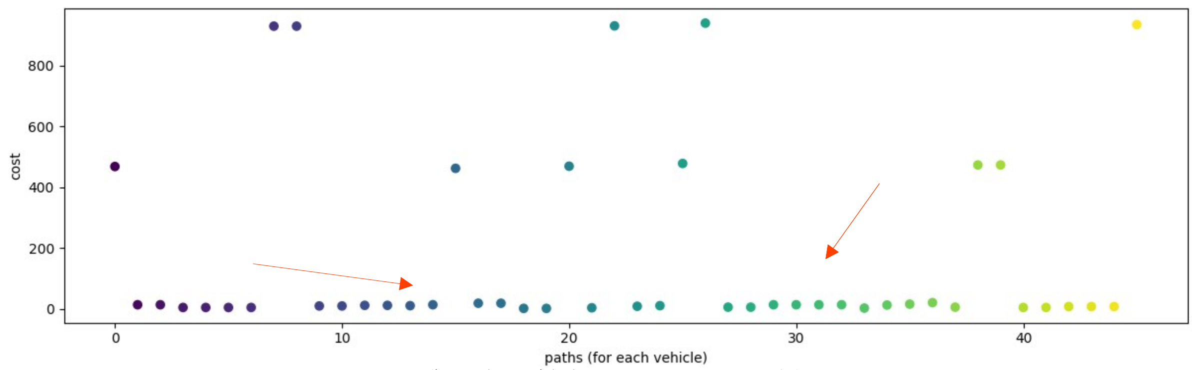
average cost of shortest path: 169.0217391304348

همانطور که دیده می‌شود، هزینه از حدود ۱۸۰ که مربوط به کوتاه‌ترین فاصله بود، به حدود ۱۷۰ با در نظر گرفتن تراکم در انتخاب مسیرها بهبود یافت.

نمودارهای مربوطه را در شکل زیر می‌توان مشاهده کرد:



شکل ۱-۵: هزینه مسیرها صرفا بر مبنای فاصله



شکل ۱-۶: هزینه مسیرها با احتساب تراکم

همانطور که در شکل با پیکان برخی از نمونه‌های بهبود هزینه‌ی مسیر بر مبنای تابع هزینه‌ی تعریف شده داده شد، با مقایسه‌ی دو نمودار بالا می‌بینیم برخی مسیرها به این ترتیب بهبود داشته‌اند و در نهایت میانگین آن‌ها از ۱۸۰ به ۱۷۰ بهبود یافته است و الگوریتم به درستی عمل کرده است.

کدهای مربوط به شبیه‌سازی ضمیمه گردیده است.

مراجع

(1) Crisostomi, E., Kirkland, S., & Shorten, R. (2011). A Google-like model of road network dynamics and its application to regulation and control. *International Journal of Control*, *84*(3), 633-651.

(2) Salman, S., & Alaswad, S. (2018). Alleviating road network congestion: Traffic pattern optimization using Markov chain traffic assignment. *Computers & Operations Research*, *99*, 191-205.