نشریه علمی- ترویجی مهندسی نقشه‌برداری و اطلاعات مکانی، دوره هفتم، شماره ۴، آذر ماه ۱۳۹۵

مروری بر روش‌های مسیریابی قطعی و ابتکاری مورد استفاده در GIS

سینا ابوالحسینی*[۱]، ابوالقاسم صادقی نیارکی[۲]

[۱] کارشناس ارشد سیستم‌های اطلاعات مکانی - دانشکده مهندسی نقشه برداری - دانشگاه صنعتی خواجه نصیرالدین طوسی
abolhoseini.sina@gmail.com

[۲] استادیار دانشکده مهندسی نقشه برداری - دانشگاه صنعتی خواجه نصیرالدین طوسی
a.sadeghi@kntu.ac.ir

(تاریخ دریافت اسفند ۱۳۹۴، تاریخ تصویب شهریور ۱۳۹۵)

چکیده

مسئله‌ی ترافیک در شهرهای بزرگ کشور یکی از مهم‌ترین مسائلی است که امروزه در حمل‌ونقل شهری مطرح است؛ زیرا بر بسیاری دیگر از حوزه‌های مهم تأثیر مستقیم و غیرمستقیم دارد. به‌طور مثال در حوزه‌ی محیط زیست و آلودگی آب و هوا، بهداشت جسمی افراد جامعه و همین‌طور بهداشت روانی آن‌ها، تأثیرات ترافیک بر هیچ کسی پوشیده نیست. یکی از راه‌هایی که می‌توان به‌وسیله‌ی آن به‌طور ضمنی از ترافیک شهری کاست، پیشنهاد بهترین مسیر بسته به پارامترهای مختلف از جمله ترافیک به کاربران است. در سیستم اطلاعات مکانی، می‌توان با تکیه بر الگوریتم‌های مختلفی که تاکنون توسعه داده شده‌اند، برای حرکت در شبکه راه شهری، مسیری را به کاربران پیشنهاد داد که از لحاظ هزینه، زمان، ترافیک، مسافت و دیگر فاکتورها بهینه باشد. هدف این مقاله بررسی انواع روش‌های مسیریابی به‌منظور استفاده در سیستم اطلاعات مکانی است. بدین منظور الگوریتم‌ها در دو بخش الگوریتم‌های قطعی و الگوریتم‌های ابتکاری بررسی می‌شوند. همچنین مسائل پیشرفته‌ی مسیریابی که به دنیای واقعی نزدیک‌تر هستند نیز شرح داده شده‌اند. در آخر مقایسه‌ای بین این الگوریتم‌ها در پارامترهای متفاوت انجام می‌شود تا محققان با مراجعه به آن بتوانند بهترین الگوریتم را به‌منظور به‌کارگیری در شرایط مسئله‌ی پیش رویشان شناسایی کنند.

واژگان کلیدی: مسیریابی، الگوریتم، قطعی، ابتکاری، سیستم اطلاعات مکانی

---

* نویسنده رابط

۱- مقدمه

هر سال ترافیک راه‌های شهری افزایش می‌یابد و تقریباً همه‌ی افراد ساعاتی از زمان خود را در ترافیک از دست داده‌اند. بدیهی است که این مشکل هم به لحاظ اقتصادی و هم به لحاظ سلامت جسمی و روانی به جامعه ضربه میزند. یکی از ابزارهایی که در سیستم اطلاعات مکانی[۱]، می‌تواند هم از تشکیل ترافیک‌های طولانی و هم از گیرافتادن در ترافیک جلوگیری کند، استفاده از الگوریتم‌های بهینه در بخش مسیریابی است.

الگوریتم‌های مسیریابی، الگوریتم‌هایی هستند که بین دو یا چند نقطه، بهترین یا کم هزینه‌ترین مسیر را مشخص می‌نمایند. این هزینه می‌تواند مسافت، زمان سفر، علاقه‌مندی، ترافیک و بسیاری پارامترهای دیگر باشد. معمولاً در این مسائل، ابتدا شبکه‌ی راه به‌صورت گرافی وزن‌دار مدل می‌شود. گراف در حقیقت مدلی ریاضی است که ترکیبی از گره و لینک است و می‌تواند روابط را مدل کند. در مرحله بعدی باید وزن لینک‌ها تعیین شود. همان‌طور که گفته شد، وزنی که به گراف نسبت داده می‌شود می‌تواند بسته به معیارها، متفاوت باشد. پس از تعیین وزن لینک‌ها الگوریتم مسیریابی بر روی گراف اجرا می‌شود.

گاهی به دلیل استفاده از فناوری‌های جدید، کاربر با جریان داده‌ها مواجه است. در این مواقع، وزنی که به لینک‌های گراف نسبت داده‌شده قابلیت تغییر در زمان را دارد و کاربر با گرافی پویا مواجه خواهد بود. برای مسیریابی روی این گراف‌ها نمی‌توان از الگوریتم‌های ایستا که در گذشته رایج بودند، استفاده کرد، زیرا هزینه‌های گراف دائماً در حال تغییر است. به همین دلیل از الگوریتم‌های پویا استفاده می‌شود. نمونه‌ای از چند الگوریتم پویا نیز در قسمت‌های مختلف ذکر شده‌اند.

الگوریتم‌های مسیریابی بررسی شده در این مقاله در ۲ دسته‌ی الگوریتم‌های قطعی و الگوریتم‌های ابتکاری دسته‌بندی می‌شوند. ابتدا الگوریتم‌های مسیریابی قطعی و سپس الگوریتم‌های مسیریابی ابتکاری بررسی می‌شوند. در نهایت در قسمت نتیجه گیری، الگوریتم‌ها با یکدیگر مقایسه می‌شوند.

۱ Geographical Information System (GIS)

۲- الگوریتم‌های قطعی

روند ثابتی برای اکثر الگوریتم‌های کوتاه‌ترین مسیر وجود دارد. قبل از پرداختن به این منظور باید با چند اصطلاح آشنا شد. منظور از برچسب در این مقاله عددی است که به هر گره i نسبت داده می‌شود و هزینه‌ی حرکت از گره مبدأ به گره مورد نظر i است و با Li نمایش داده می‌شود. Pi لیستی است که در آن ترتیب لینک‌های کوتاه‌ترین مسیر تا گره i مشخص شده است. Q هم مجموعه‌ای از گره‌های واجد شرایطی است که باید مورد بررسی قرار بگیرند. حال روند الگوریتم‌های کوتاه‌ترین مسیر به این شکل است (شکل۱):

قدم اول: $(i = 0)$ و $\forall i \neq j$ $(L_j = \infty)$ و $(L_i = 0)$ و $(P_i = UNLL)$
و تعریف $Q = \{i\}$
قدم دوم: انتخاب و حذف یک گره (i) از Q
قدم سوم: اسکن کردن هر لینکی که از گره i خارج می‌شود.
برای هر لینک داریم $a = (i , j)$:
اگر $(L_i + c_a < L_j)$ **آنگاه**
$P_j = a$ **و** $L_j = L_i + c_a$
اضافه کردن گره j به Q
قدم چهارم: اگر $(Q = \emptyset)$ **آنگاه STOP.**
در غیراین‌صورت به قدم دوم برو.

شکل۱- شبه کد عمومی الگوریتم‌های مسیریابی

چیزی که مابین این الگوریتم‌ها تفاوت ساز می‌شود، نحوه ساختار داده برای اسکن گره‌های واجد شرایط و نحوه‌ی شناسایی، انتخاب و بررسی آن‌هاست [۳]. بر همین اساس الگوریتم‌های کوتاه‌ترین مسیر بهینه به دو دسته‌ی الگوریتم‌های تعیین برچسب[۲] و الگوریتم‌های تصحیح برچسب[۳] تقسیم بندی می‌شوند.

۲-۱- الگوریتم‌های تعیین برچسب LS

در این الگوریتم‌ها مجموعه‌ی گره‌های واجد شرایط بر حسب هزینه‌ی مسیر از گره آغازین تا گره مدنظر مرتب شده‌اند. این الگوریتم‌ها تنها مسیر بین نقطه شروع و پایان را می‌یابند و زمانی الگوریتم می‌ایستد که برچسب گره پایانی تعیین شود. به همین دلیل این الگوریتم‌ها برای تعیین مسیر بین دو نقطه‌ی مشخص مناسب هستند [۴].

۲ Label Setting
۳ Label Correction

در حقیقت در هر تکرار، الگوریتم LS به‌صورت دائمی برچسب یک گره را تعیین می‌کند که کوتاه‌ترین فاصله‌ی آن تا گره مبدأ است. به همین دلیل، نمی‌توانند در گراف با لینک‌های منفی به کار بروند.

۲-۱-۱- Dijkstra

برای هر گره در گراف یک برچسب تعیین می‌شود که برابر است با کوتاه‌ترین فاصله‌ی آن گره تا گره مبدأ. الگوریتم به‌صورت ترتیبی کار می‌کند و روند آن تا زمانی ادامه پیدا می‌کند که تمامی گره‌ها بررسی شوند در غیر این صورت از میان گره‌های بررسی نشده، گره‌ای که کمترین مقدار را دارد برای بررسی بعدی انتخاب می‌شود. قسمتی از کد الگوریتم دایجسترا را می‌توان در شکل ۲ مشاهده کرد [۱].

```
for (int i = 0; i < n; i++) {
    // vertex
    int v = -1;
    // finding minimum label among unvisited vertices
    for (int j = 0; j < n; j++) {
        if (u[j] == false && (v == -1 || d[j] < d[v])) {
            v = j;
        }
    }
    // set as visited.
    u[v] = true;

    for (int j = 0; j < n; j++) {
        // if there is exists path between v and j vertices
        if (a[v][j] > 0) {
            if (d[v] + a[v][j] < d[j]) {
                d[j] = d[v] + a[v][j];
            }
        }
    }
}
```

شکل۲- قسمتی از کد الگوریتم دایجسترا [۱]

این الگوریتم از دسته‌ی الگوریتم‌های تک منبع[1] است. تک منبع بودن یک الگوریتم بدین معناست که این الگوریتم در برای انجام مسیریابی، تنها یک نقطه را به‌عنوان نقطه‌ی شروع در نظر می‌گیرد.

۲-۲- الگوریتم‌های تصحیح برچسب LC

این الگوریتم‌ها از یک ساختار لیست مانند برای گره‌های واجد شرایطی که باید بررسی شوند، استفاده می‌کنند. این الگوریتم به‌صورت دائمی هیچ برچسبی را تعیین نمی‌کند بلکه برچسب را برای هر گره، پیش بینی می‌کند و در هر تکرار تعدادی از آن‌ها را تصحیح می‌کند [۴].

۲-۲-۱- الگوریتم Bellman-Ford

الگوریتم Bellman-Ford از نمونه‌ی الگوریتم‌های تصحیح برچسب است که اولین بار توسط Lester Ford و Richard Bellman در سال ۱۹۵۸ مطرح شده است. در این الگوریتم ابتدا برچسب تمامی گره‌ها برابر با ∞ در نظر گرفته می‌شود به‌جز گره مبدأ که صفر است. سپس الگوریتم با گره‌هایی که به‌طور مستقیم به گره مبدأ راه دارند شروع می‌کند و به آن‌ها برچسبی می‌دهد که برابر با فاصله‌ی آن‌ها از نقطه‌ی مبدأ است. سپس به نقاط همسایه منتقل شده و همین کار را برای آن‌ها نیز انجام می‌دهد. مشاهده می‌شود که در برخی مواقع برچسب‌های انتصاب شده به گره‌ها کاهش می‌یابند. این کار برای تمامی گره‌ها انجام می‌شود و کل فرآیند V بار تکرار می‌شود (V تعداد گره‌های گراف است). در نهایت اسکنی نهایی بر روی گره‌ها انجام می‌شود؛ اگر گرهی نیاز به اصلاح داشت، بدین معنا خواهد بود که در گراف حداقل یک دور منفی وجود دارد.

۲-۲-۲- الگوریتم Floyd-Warshll

این الگوریتم یک الگوریتم ساده برای محاسبه‌ی کوتاه‌ترین مسیر است (شکل ۳). این الگوریتم در صورتی که در گراف دور منفی وجود داشته باشد قادر به محاسبه‌ی بهترین مسیر نخواهد بود و تبدیل به یک مسئله‌ی NP-hard خواهد شد؛ اما با این حال قادر به شناسایی دور منفی است [۲].

```
M[i, j] := ∞ ∀i ≠ j
M[i, i] := 0 ∀i
M[i, j] := c((i, j)) ∀(i, j) ∈ E(G)
for i := 1 to n do
    for j := 1 to n do
        for k := 1 to n do
            if M[j, k] > M[j, i] + M[i, k] then M[j, k] := M[j, i] + M[i, k]
for i := 1 to n do
    if M[i, i] < 0 then return ('graph contains a negative cycle')
```

شکل۳- شبه کد الگوریتم فلوید-وارشال [۲]

این الگوریتم از دسته‌ی الگوریتم‌های تمام-زوج[2] است بدین معنا با یک‌بار اجرا بر روی شبکه، کوتاه‌ترین مسیر را بین تمامی جفت نقاط محاسبه می‌کنند.

۲-۲-۳- الگوریتم Johnson

این الگوریتم از دسته الگوریتم‌های تمام-زوج است که امکان کار در گراف با وزن منفی را دارد اما باوجود دور

---

۱ Single source

۲ All-pair

منفی پاسخ صحیحی نمی‌دهد. این الگوریتم از دو الگوریتم Bellman-Ford و الگوریتم Dijkstra نیز استفاده می‌کند.

روند این الگوریتم به این صورت است. یک گره جدید به گراف افزوده می‌شود که با یکی از گره‌ها در ارتباط است و وزن لینک آن صفر است (گره q). سپس الگوریتم Bellman-Ford برای محاسبه‌ی کمترین فاصله‌ی بین تمامی گره‌ها (v) و گره q استفاده می‌شود (h(v)). در صورت شناسایی دور منفی در این قسمت توسط الگوریتم Bellman-Ford، اجرای الگوریتم متوقف می‌شود. در مرحله‌ی بعدی لینک‌های گراف اصلی دوباره بر اساس معادله ۱ وزن دهی می‌شوند. در این فرمول W(x,y) وزن قبلی لینک است و مقادیر h(x) و h(y) مقادیر محاسبه شده در مرحله‌ی قبلی هستند. نتیجه‌ی این مرحله الگوریتم، گرافی با وزن‌های مثبت است.

$$\boldsymbol{W(x,y) = W(x,y) + h(x) - h(y)} \qquad (۱)$$

در نهایت برای هر گره، الگوریتم Dijkstra اجرا می‌شود و کوتاه‌ترین مسیرها با توجه به مقادیر جدید محاسبه می‌شوند [۳].

۳- الگوریتم‌های ابتکاری

الگوریتم‌هایی که در بخش دوم به آن‌ها پرداخته شد، الگوریتم‌هایی هستند که از لحاظ محاسباتی بسیار سنگین هستند و از این دست الگوریتم‌ها نمی‌توان در شرایط واقعی استفاده کرد. محاسبات سنگین در این الگوریتم‌ها به این دلیل اتفاق می‌افتند که الگوریتم از دانش قبلی و دانشی که رفته‌رفته با جستجوی شبکه به دست می‌آورد، استفاده‌ای نمی‌کند. استراتژی‌های متفاوتی برای استفاده از دانش در این زمینه وجود دارد که در ادامه به آن‌ها پرداخته خواهد شد [۴].

۳-۱- محدود کردن منطقه جستجو

در این ایده از استراتژی‌هایی استفاده می‌شود که از خواص مسیر بین گره مبدأ و پایانی برای محدود کردن منطقه تحت جستجو استفاده می‌کند. دو نمونه از مشهورترین این الگوریتم‌ها، الگوریتم‌های A* و روش هرس کردن شاخه[۱] هستند [۴].

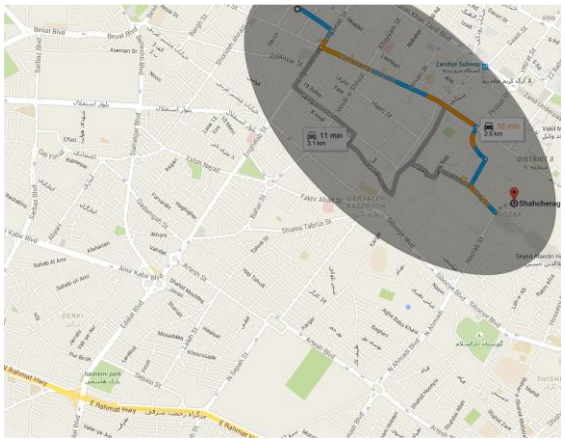
۳-۱-۱- روش هرس کردن شاخه

در یک گراف از شبکه‌ی راه وزنی که به لینک‌ها داده می‌شود تابعی از فاصله و یا زمان سفر در آن لینک است. بر این اساس می‌توان منطقه‌ای را در اطراف گره مبدأ و مقصد تعریف کرد که جستجو فقط در آن منطقه صورت بگیرد. به‌طور مثال، در [۵] از معادله‌ی ۲ برای تعیین لینک‌هایی که باید جستجو شوند استفاده شده است.

$$\boldsymbol{L_i + e_{(i,d)} \leq E_{(o,d)}} \qquad (۲)$$

Li مقدار کمترین زمان سفر بین مبدأ و مقصد، e کمترین زمان سفر تخمین زده شده بین گره i و مقصد است و E کران بالای تخمین زمان سفر بین مبدأ و مقصد است.

با اجرای این قید، فضای جستجو کاهش می‌یابد. نمونه اجرای این روش جستجو را می‌توان در شکل۴ مشاهده کرد [۴].



شکل۴- کاهش فضای جستجو با استفاده از روش هرس کردن شاخه‌ها

۳-۱-۲- روش A*

فرض کنید که علاوه بر اطلاعات موجود در گراف شبکه راه، اطلاعات اضافی نیز موجود است که می‌تواند به ما در تخمین فاصله بین هر گره تا گره مقصد کمک کند (h(i)). به‌عنوان مثال یکی از اطلاعاتی که می‌توان استفاده

۱ Branch pruning method

کرد، فاصله‌ی اقلیدسی بین دو گره است (با داشتن مختصات دو نقطه به‌عنوان اطلاعات اضافی).

روند انجام الگوریتم A* بدین شرح است. در قدم اول، تمام گره‌ها به‌جز گره مبدأ و مقصد برچسب بینهایت (∞) می‌گیرند. برچسب گره مبدأ صفر و برچسب گره مقصد برابر با مقدار پیش‌بینی شده‌ی فاصله‌ی بین مبدأ و مقصد می‌شود و S={s} می‌شود. سپس گره بعدی انتخاب می‌شود. برای این کار گرهی از S انتخاب می‌شود که مقدار d(v) آن از مقدار سایر گره‌ها کم‌تر است. مقدار d(v) از معادله‌ی ۳ محاسبه می‌شود.

$$\boldsymbol{d(v) = g(s) + h(s)} \qquad (۳)$$

مقدار g(s) برابر با مقدار پیش‌بینی شده فاصله گره مبدأ تا گره v است و مقدار h(s) مقدار پیش‌بینی شده برای فاصله گره v تا گره مقصد.

در مرحله‌ی بعد برای هر گره w که لینکی متصل به v داشته باشد، اگر $\boldsymbol{g(w) > g(v) + l(v,w)}$ باشد، پس $\boldsymbol{g(w) = g(v) + l(v,w)}$ و برچسب گره v برابر با $\boldsymbol{g(w) + h(w)}$ می‌شود. سپس اگر w عضو S نبوده باشد، به S اضافه می‌شود و دوباره به مرحله انتخاب گره از میان S بازمی‌گردد. شرط توقف در این الگوریتم، انتخاب شدن گره مقصد به‌عنوان گره بعدی برای بررسی است.

الگوریتم A* انواع مختلفی دارد که در ادامه برخی از آن‌ها را که از باقی شاخص‌تر هستند را بررسی می‌کنیم.

۳-۱-۲-۱- A* پویا[۱] (D*)

این الگوریتم کم هزینه‌ترین مسیر بین گره مبدأ و مقصد با امکان تغییر در هزینه هر لینک را محاسبه می‌کند. این تغییرات می‌توانند هم مثبت و هم منفی باشند. در این الگوریتم، ابتدا مقدار g(s) و rhs(s) برای گره‌ها به‌جز گره مقصد برابر با بینهایت در نظر گرفته می‌شود. g(s) مقدار فاصله بین گره s و مقصد است. مقدار rhs(s) از معادله ۴ محاسبه می‌شود.

$$\boldsymbol{rhs(s)} = \begin{cases} \boldsymbol{0} & \boldsymbol{if\ s = s_{goal}} \\ \boldsymbol{\min_{s' \in Succ(s)} (c(s,s') + g(s')} & \boldsymbol{otherwise} \end{cases} \qquad (۴)$$

همچون A* این الگوریتم نیز از مقدار پیش بینی شده فاصله بین هر گره از مقصد استفاده می‌کند (h(s)) که برای رسیدن به جواب صحیح، این مقدار باید از مقدار واقعی کمتر باشد. گره‌هایی که پایدار نیستند ($\boldsymbol{g(s) \neq rhs(s)}$) در لیست گره‌هایی قرار می‌گیرند که نیاز به بروز رسانی و پایدار شدن دارند. به‌محض اینکه تغییری در هزینه‌ها رخ می‌دهد، الگوریتم بلافاصله مقدار rhs تمامی گره‌هایی که تحت تأثیر قرار گرفته‌اند را بروز رسانی می‌کند و آن‌هایی که ناپایدار هستند را درون لیست جستجو قرار می‌دهد. زمانی که هزینه‌ها افزایش یافته‌اند، گره‌هایی که کمتر از حد پایداری هستند ($\boldsymbol{g(s) < rhs(s)}$) باید جستجو شوند و برعکس، زمانی که هزینه‌ها کاهش می‌یابند، گره‌هایی که بیشتر از حد پایداری هستند ($\boldsymbol{g(s) > rhs(s)}$) جستجو می‌شوند [۶].

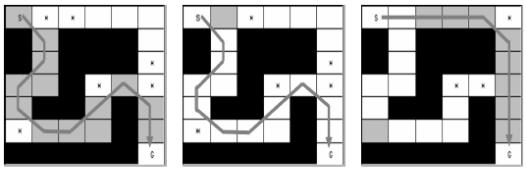
می‌توان از الگوریتم A* هم برای شبکه‌های پویا استفاده کرد، به این‌گونه که در هر تغییری که ایجاد می‌شود، الگوریتم از پایه دوباره اجرا شود، اما با توجه به تحقیقات، الگوریتم D* و مشتقات آن (D* Lite، Delayed D* و ...) همگی عملکرد بهتری نسبت به این روش داشته‌اند [۷].

۳-۱-۲-۲- Anytime D*

در الگوریتم AD* از قابلیت‌های الگوریتم D* و الگوریتم ARA* استفاده شده است [۸]. الگوریتم D* در بخش ۳-۱-۲-۱ توضیح داده شد. الگوریتم ARA* الگوریتمی است که در مدت زمان کوتاهی یک جواب بهینه‌ی محلی را به کاربر می‌دهد تا کاربر بتواند تصمیم‌گیری سریع‌تری داشته باشد، سپس در زمان باقیمانده تا رسیدن به گره بعدی، جواب مرحله‌ی قبل را بهبود می‌دهد تا زمانی که به بهینه‌ی کلی برسد. به‌منظور این هدف، این الگوریتم جستجو را با ضریب تورمی ($1<\epsilon$) انجام می‌دهد که منجر به جواب بهینه‌ی محلی می‌شود. این جواب به کاربر ارائه می‌شود، ضریب تورم کاهش می‌یابد و جستجو ادامه می‌یابد و در هر مرحله کاهش ضریب تورمی، جواب به کاربر ارائه می‌شود. این روند تا زمانی که ($\epsilon = 1$) انجام می‌شود (شکل ۵) [۹].

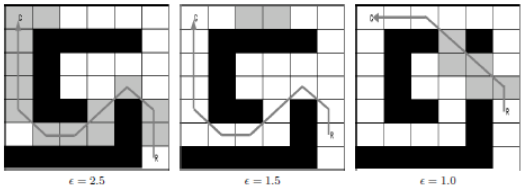
---

۱ Dynamic A*



شکل۵- الگوریتم *ARA در حال کاهش ضریب تورمی و بهبود جواب [۹]

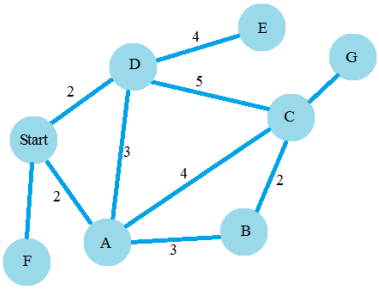
در ترکیب الگوریتم *ARA با الگوریتم *D، الگوریتم به وجود می‌آید که در زمان کوتاهی به کاربر پاسخ می‌دهد و سپس در زمان باقیمانده آن را بهبود می‌دهد. در همین حین زمانی که تغییری در محیط باعث عوض شدن هزینه در لینک‌ها می‌شود، گره‌های تأثیر پذیرفته در لیست اولویت جستجو قرار می‌گیرند و الگوریتم نسبت به تغییر مسیر کاربر اقدام می‌کند (شکل ۶) [۸].



شکل۶- الگوریتم *AD در حال کاهش ضریب تورم و تصحیح مسیر با توجه به تغییر محیط مسیریابی [۸]

۳-۱-۲-۳ -IDA*

این الگوریتم یک نسخه از روش پیمایش گراف جستجوی عمق اول[۱] است. در این نوع پیمایش، جستجو از یک گره شروع شده و رئوس مجاور انتخاب و بررسی می‌شوند. در صورت برخورد به گرهی که قبلاً بررسی نشده است، آن گره به‌عنوان گره اصلی انتخاب می‌شود و گره‌های مجاورش بررسی می‌شوند. در صورتی که گره قبلاً بررسی شده باشد، جستجو یک سطح به عقب منتقل می‌شود. با این روش جستجو در هر مرحله تا جایی که امکان پذیر است در عمق وارد می‌شود (شکل۷).



شکل۷- نحوه‌ی عملکرد DFS

در شکل۷ گرافی نمایش داده شده است که اگر DFS را روی آن اجرا شود و قابلیت به یاد سپردن گره‌های بررسی شده را وجود داشته باشد، ترتیب بررسی نقاط به‌صورت (Start, F, A, B, C, G, D, E) خواهد بود. خروجی این جستجو یک درخت پوشا است.

الگوریتم *IDA که بر مبنای DFS توسعه یافته است، به‌جای استفاده از شرط رسیدن به پایین‌ترین عمق برای هر تکرار، از تابع $\boldsymbol{f(n) = g(n) + h(n)}$ به‌منظور توقف تکرار استفاده می‌کند. g(n) هزینه سفر از گره مبدأ به گره n است و h(n) یک پیش‌بینی شده برای هزینه‌ی سفر از n به گره مقصد است. لازم به ذکر است که همانند الگوریتم *A اگر بخواهیم از جواب الگوریتم *IDA مطمئن باشیم باید مقدار h(n) از مقدار واقعی آن کمتر باشد.

۳-۱-۲-۴- ALT

این الگوریتم بر پایه‌ی الگوریتم *A است که از عوارض شاخص[۲] برای محاسبه‌ی کران پایین، و نامعادله‌ی مثلثاتی استفاده می‌کند. در نامعادله‌ی مثلثی، برای هر ضلع x، y و z معادله‌ی ۵ را خواهیم داشت.

$$\boldsymbol{Z \leq X + Y} \qquad (۵)$$

این الگوریتم می‌تواند تقریبی استاندارد، کران‌دار و غیر منفی را در زمان کوتاهی تولید کند که در الگوریتم *A به‌عنوان h(s) استفاده می‌شود. برای این کار از گره‌هایی تحت عنوان گره شاخص(L) استفاده می‌شود. برای هر سه گره موجود در گراف، تابع *dist مقدار واقعی فاصله بین هر یک از این سه گره است که در معادله ۶ صدق می‌کند.

$$\boldsymbol{dist^*(u,w) \leq dist^*(u,v) + dist^*(v,w)} \qquad (۶)$$

برای محاسبه‌ی مقدار تقریبی فاصله برای هدف بخشیدن به جستجو، از معادله‌ی۷ در قسمت پیش‌پردازش استفاده می‌شود. این معادله در نامساوی مثلثاتی صدق می‌کند (شکل ۸).

$$\boldsymbol{h_l(v) = \max\{0, \max\{dist^*(L,d) - dist^*(L,v) | L \in \mathbb{L}\}\}} \qquad (۷)$$

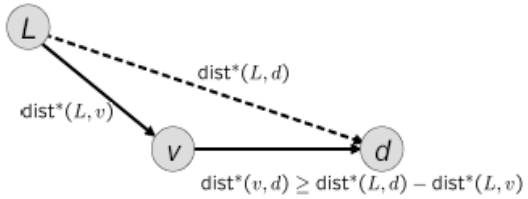
---

۱ Depth-First Search (DFS)

۲ Landmarks

که d گره مقصد است و v می‌تواند هریک از گره‌های موجود در گراف باشد و همچنین $\mathbb{L}$ مجموعه‌ی گره‌های شاخص است. یکی از روش‌های پر کردن مجموعه $\mathbb{L}$ انتخاب یک گره از گراف، یافتن دورترین گره از نظر اقلیدسی به آن و افزودن گره یافته شده به مجموعه $\mathbb{L}$ است. این روند تا زمانی ادامه پیدا می‌کند که تعداد گره‌ها در مجموعه گره‌های شاخص برابر با تعداد گره‌های گراف شود [۱۰]. بهترین گره‌های شاخص، گره‌هایی هستند که زمانی که از مبدأ به سمت مقصد می‌نگریم، در فضای پشت مقصد واقع شده باشند.

این روش جستجو در سرعت بخشیدن به الگوریتم A* زیاد تأثیری ندارد، اما این روش در ترکیب با دیگر روش‌ها (به‌طور مثال در الگوریتم REAL) بسیار مؤثر واقع می‌شود [۱۱]. همچنین در تز [۱۰] از این الگوریتم برای گراف‌های پویا استفاده شده است که هزینه لینک‌ها در طول زمان می‌تواند متغیر باشد.



شکل۸- نمایشی از نحوه‌ی محاسبه مقدار پیش‌بینی شده برای فاصله [۱۰]

۳-۱-۲-۵- REAL

این الگوریتم ترکیبی از جستجو به‌وسیله روش‌های ALT و REACH (توضیح در بخش ۳-۲-۲) و الگوریتم مسیریابی A* است. همچنین از دو بخش پیش‌پردازش تشکیل‌شده است. در پیش‌پردازش اول، مقدار reach برای هر گره محاسبه می‌شود و در پیش‌پردازش دوم، گره‌های شاخص تولید می‌شوند. این دو بخش پیش‌پردازش هم می‌توانند به یکدیگر مرتبط باشند و هم می‌توانند مستقل از یکدیگر باشند [۱۲].

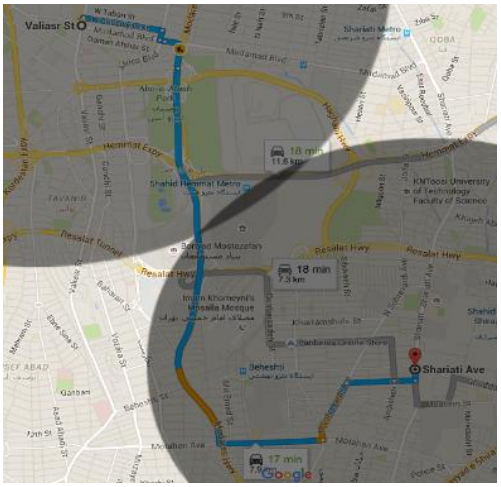
برای مقایسه این روش با روش ALT که در بخش قبل توضیح داده شد، در مقاله با داده‌های واقعی آزمایش انجام شده است. در شبکه راه کشور ایالات متحده آمریکا با ۲۳,۹۴۷,۳۴۷ گره و ۵۸,۳۳۳,۴۴۴ لینک، پیش پردازش الگوریتم ALT در ۱۸,۶ دقیقه انجام می‌شود و الگوریتم REAL بهبود یافته در ۶۳,۹ دقیقه، تعداد گره‌های بررسی شده در ALT در هنگام مسیریابی اصلی برابر با ۱۸۷,۹۸۶ و در REAL بهبود یافته برابر با ۳,۰۱۱ عدد است؛ زمان اجرای مسیریابی اصلی در ALT در حدود ۴۰۰ میلی‌ثانیه و در REAL بهبود یافته این مقدار برابر با ۱ است. همان‌طور که ذکر شد، مشاهده می‌شود که الگوریتم REAL بسیار بهتر از الگوریتم ALT عمل می‌کند اما در مرحله پیش‌پردازش نیازمند زمان بیشتری است [۱۲].

۳-۲- تجزیه مسئله‌ی جستجو[۱]

زمانی که برای محاسبات کوتاه‌ترین مسیر نیاز است وابسته به تعداد گره‌هایی است که تا قبل از رسیدن به گره مقصد جستجو شده‌اند. به همین دلیل اگر بتوان به روشی این گره‌ها را به زیر مسئله‌هایی جداگانه تقسیم بندی کرد، می‌توان پیشرفت بسیار خوبی در زمینه‌ی کاهش زمان محاسبات انتظار داشت [۴].

۳-۲-۱- روش جستجوی دو جهته[۲]

بیشتر روش‌هایی که تاکنون مطرح شده‌اند یک جهته هستند که یا از مبدأ به مقصد و یا از مقصد به مبدأ حرکت می‌کنند اما در روش جستجوی دوطرفه مسئله را به دو قسمت مجزا تقسیم بندی می‌کند (شکل۹). در این الگوریتم دو عامل بسیار مهم هستند، عامل اول روش‌های مسیریابی هستند که در هریک از دو قسمت استفاده می‌شوند و دومین عامل که مهم‌تر است، شرط توقف جستجو است. بااینکه معیار توقف بسیار مورد پژوهش قرار گرفته است (مانند [۱۳])، اما حتی با استفاده از بهینه‌ترین معیارها نیز،جواب‌های این روش جستجو نسبت به روش‌های تک جهته، در بسیاری از موارد، نامرغوب‌تر است [۴].



شکل۹- تقسیم مسئله‌ی مسیریابی به دو قسمت مجزا در روش جستجوی دو جهته

۱ Decompose the search problem
۲ Bi-directional search algorithm

از این روش جستجو در الگوریتم‌های متفاوتی می‌توان استفاده کرد. به‌طور مثال در ترکیب با الگوریتم *A، الگوریتم BHPA توسعه داده‌شده است [۱۴].

۳-۲-۲- روش هدف فرعی

اهداف فرعی در یک مسئله‌ی مسیریابی در شبکه راه، گره‌ها و یا لینک‌هایی تعریف می‌شوند که میان گره مبدأ و مقصد تعریف واقع شده‌اند. اگر هدف فرعی مشخص باشد، می‌توان با تقسیم کردن مسئله‌ی مسیریابی به دو زیر مسئله کوچک‌تر، عملکرد سریع‌تری داشت. البته بهبود عملکرد وابسته به مکان و تعداد اهداف فرعی است. در [۱۵] پس از تقسیم مسئله مسیریابی به‌وسیله‌ی هدف فرعی به دو زیر مسئله، یک بار بهترین مسیر از مبدأ تا هدف فرعی محاسبه می‌شود و بار دیگر برای هدف فرعی تا مقصد.

یک مشکل که ممکن است در اجرای این عملکرد پیش بیاید، عدم یافتن بهترین مسیر به دلیل عدم وجود گره هدف فرعی در دنباله بهترین مسیر است [۱۴].

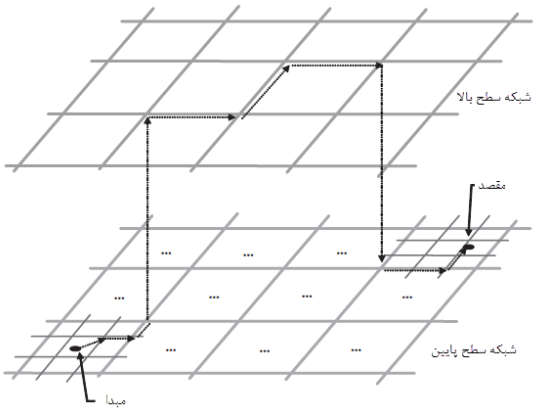
۳-۳- محدود کردن لینک‌های جستجو شده

در روش‌های معمول زمانی که یک گره برچسب می‌خورد و مقداری به آن اختصاص داده می‌شود، تمامی لینک‌های ورودی و خروجی آن در مرحله بعدی بررسی می‌شوند. ایده‌ی اصلی روش‌های محدود کردن لینک‌های جستجو شده، انتخاب تعداد خاصی از لینک‌های متصل به یک گره است تا ادامه‌ی جستجو، تنها بر روی آن‌ها صورت بگیرد و نه تمامی لینک‌ها [۱۴].

۳-۳-۱- روش جستجوی سلسله مراتبی

ایده‌ای که در این روش مطرح‌شده است، این است که برای یافتن بهترین جواب در مسئله‌ای پیچیده، ابتدا باید بر روی کلیات تمرکز کرد و سپس به درون جزئیات بیشتر رفت (شکل۱۰). با اینکه این ایده بسیار ساده است اما پیاده‌سازی آن بسیار پیچیده است. سناریوی زیر را در نظر بگیرید. شما می‌خواهید مبدائی را به سمت مقصدی ترک کنید. ابتدا، خیابان‌های اصلی که شما را می‌توانند از مبدأ به مقصد ببرند را پیدا می‌کنید، سپس جزئیات را بیشتر می‌کنید و راه رسیدن هر یک از گره‌ها به خیابان‌های اصلی را می‌یابید. پیچیدگی این روش به دو مسئله بازمی‌گردد. مسئله‌ی اول نحوه تبدیل به ساختار سلسله مراتبی و مسئله‌ی دوم چگونگی انتقال جستجو بین لایه‌های مختلف است [۴].

از این روش جستجو نیز می‌توان در روش‌های مختلف مسیریابی استفاده کرد. به‌طور مثال در ترکیب این روش با الگوریتم Dijkstra، الگوریتم Hirearchical Dijkstra را خواهیم داشت.



شکل۱۰- روش جستجوی سلسله مراتبی [۴]

۳-۳-۲- روش جستجوی برد مبنا[۱]

مقدار برد (Reach) مقدار کمینه‌ی فاصله از مبدأ به گره مدنظر و از گره مدنظر به مقصد است. برای هر گره v بر روی مسیر P از مبدأ s به مقصد t، مقدار reach برای گره v در مسیر P، کمینه مقدار بین فاصله s تا v و فاصله v تا t است. این مقدار r(v) برای هر گره باید بیشینه مقدار را در تمامی کوتاه‌ترین مسیرهای عبور کرده از گره v داشته باشد. در صورتی که گرهی در معادله ۸ صدق کند، آنگاه الگوریتم مسیریابی (به‌طور مثال Dijkstra یا *A) نیازی به بررسی آن گره ندارند.

$$\overline{r(v)} < \underline{dist(s,v)} \quad \textbf{AND} \quad \overline{r(v)} < \underline{dist(v,t)} \qquad (۸)$$

مقدار $\overline{r(v)}$ کران بالای r(v) و $\underline{dist(x,y)}$ کران پایین فاصله بین x وy است [۱۶]. پس از مشخص شدن گره‌هایی که باید بررسی شوند، می‌توان از الگوریتم‌هایی

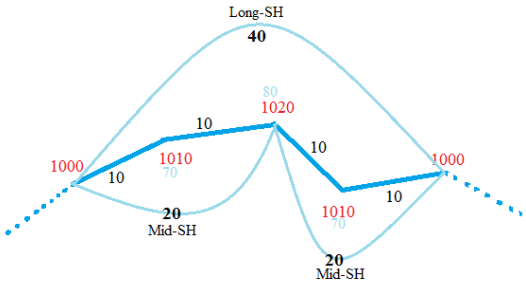
۱ Reach-Based

همچون Dijkstra و یا A* برای یافتن کوتاه‌ترین مسیر استفاده کرد [۱۱].

۳-۳-۳- روش استفاده از میانبر[۱]

یکی دیگر از روش‌های ابتکاری که می‌توان آن را در بخش مربوط به محدود کردن لینک‌های جستجو شده قرار داد، استفاده از میانبرها در ساده‌سازی شبکه راه است [۱۱].

برای توضیح ساده‌ی این روش از شکل ۱۱ استفاده می‌کنیم. همان‌طور که در شکل مشاهده می‌شود، گراف اصلی شامل ۴ لینک با طول مساوی و مقدار reach (مراجعه شود به بخش ۳-۳-۲) بالا است. می‌توان به‌جای این لینک‌ها از یک لینک میان‌بر (Long-SH) و یا دو لینک (Mid-SH) استفاده کرد. پس از اینکه میان‌برها تشکیل شد، باید مقدار reach را در گره‌های میانی کاهش داد تا در زمان جستجو، وارد جستجو نشوند.



شکل ۱۱- نحوه استفاده از میان‌بر در گراف شبکه راه

جدول ۱- مقایسه روش میان‌بر و Dijkstra دو جهته

روش	پیش پردازش		مسیریابی	
	زمان	حجم	متوسط گره‌های جستجو شده	زمان
Bidirectional Dijkstra	-	MB ۲۸	۵۱۸،۷۲۳	ms ۳۴۰
Shortcuts	۱۷ دقیقه	MB ۳۴	۵۳،۸۸۸	ms ۳۰

با این روش، هم می‌توان سرعت مسیریابی را افزایش داد و هم می‌توان در بخش پیش‌پردازش باعث افزایش سرعت شود. در مقابل این مزایا، این الگوریتم سبب افزایش کمی در حجم داده‌های ذخیره شده می‌شود (به دلیل افزایش لینک‌های گراف). در جدول ۱ می‌توان نتایج مقایسه با الگوریتم Dijkstra دو جهته را مشاهده نمود.

۱ Shortcuts

۴- مقایسه الگوریتم‌های قطعی و ابتکاری

الگوریتم‌هایی که تاکنون بررسی شده‌اند جز الگوریتم‌های ساده قطعی و ابتکاری هستند که در سیستم اطلاعات مکانی از آن‌ها استفاده می‌شود. معمولاً برای ذکر پیچیدگی الگوریتم‌ها از نماد O بزرگ[۲] استفاده می‌شود. به‌صورت خیلی ساده الگوریتمی که پیچیدگی آن ۱ است (O(1)) با افزایش اندازه داده‌های ورودی، زمان اجرای آن تغییری نمی‌کند، پس در نتیجه الگوریتم که پیچیدگی آن O(N) است، زمان اجرای آن به‌صورت خطی با اندازه داده-های ورودی تغییر می‌کند. به همین صورت الگوریتم‌هایی با پیچیدگی $O(N^2)$ و توابع دیگر است، تشریح می‌شوند. در جدول ۲ سعی شده است که اطلاعات الگوریتم‌هایی که در بخش‌های مختلف این مقاله ذکر و بررسی شده‌اند، به صورتی یکپارچه جمع‌بندی شوند. در این جدول نماد |V| معرف تعداد گره‌های گراف ورودی، |E| تعداد لینک‌های گراف ورودی، k ضریب انشعاب گره‌های گراف و m فاصله-ی مبدأ و مقصد است.

همان‌طور که مشاهده می‌شود، الگوریتم A* با مقدار ابتکاری که در محدود کردن گره‌های بررسی استفاده می‌کند، توانسته است مقدار زمان محاسبات را در الگوریتم Dijkstra کاهش دهد. الگوریتم Bellman-Ford با اینکه محاسبات نسبتاً سنگین‌تری نسبت به الگوریتم‌های Dijkstra و A* دارد اما توانایی کشف حلقه‌ی منفی و کار با اعداد منفی در گراف را دارد. الگوریتم Floyd-Warshall در مقایسه با الگوریتم Bellman-Ford نیاز به محاسبات بیشتری دارد زیرا این الگوریتم برای تمام جفت گره‌های موجود در گراف، کوتاه‌ترین مسیر را محاسبه می‌کنند. الگوریتم Johnson نیز مانند الگوریتم Bellman-Ford برای تمام جفت گره‌های موجود در گراف کوتاه‌ترین مسیر را محاسبه می‌کند اما توان استفاده در گراف‌هایی با دور منفی را ندارد. این الگوریتم در بخشی از فرآیند خود از الگوریتم Bellman-Ford استفاده می‌کند تا لینک‌های منفی را به لینک مثبت تبدیل کند. سپس از الگوریتم مسیریابی همچون Dijkstra برای یافتن کوتاه‌ترین مسیر بین نقاط استفاده می‌کند.

۲ Big O notation

الگوریتم‌های دیگری که ذکر شده‌اند، الگوریتم Dijkstra دو جهته و الگوریتم Dijkstra سلسله مراتبی است که برای بیان پیچیدگی محاسباتی آن از دو پارامتر ضریب انشعاب گره در گراف (k) و فاصله‌ی بین مبدأ و مقصد (m) استفاده می‌شود و به نظر می‌رسد با توجه به نوع مسئله‌ای که با آن روبرو هستیم می‌توانند باعث بهبود الگوریتم Dijkstra شوند. در الگوریتم Dijkstra دو جهته، فضای جستجو تقریباً به نصف کاهش می‌یابد که باعث بهبود سرعت عملکرد الگوریتم خواهد شد. همین‌طور در الگوریتم سلسله مراتبی نیز به دلیل اینکه بخش اعظمی از مسیریابی در گراف سطوح بالا انجام می‌شود که جزئیات کمتری دارند، و سپس با جزئیات بیشتر مسیریابی برای بخش محدودی از فضا انجام می‌شود، می‌توان انتظار عملکرد بهتری را نسبت به الگوریتم اصلی داشت.

الگوریتم *IDA با الهام گیری از الگوریتم *A در روش پیمایش گراف DFS توانسته است کوتاه‌ترین مسیر را بین گره مبدأ ورودی و دیگر گره‌های شبکه بیابد. دیگر الگوریتم‌هایی که بر پایه‌ی الگوریتم *A توسعه یافته‌اند و توان عمل در شبکه‌های پویا را دارند، الگوریتم‌های *D و *AD هستند که *AD توسعه یافته‌ی الگوریتم *D است. این الگوریتم کوتاه‌ترین مسیر را برای کاربر فراهم می‌کند، سپس تا زمانی که کاربر به مقصد نرسیده است، به انتظار تغییرات در هزینه‌های گراف شبکه راه می‌نشیند و در صورت یافتن تغییری مسیر را از مکان فعلی کاربر اصلاح می‌کند. الگوریتم *AD در زمانی به کار گرفته می‌شود که کاربر خواستار مسیریابی در مدت زمان کوتاهی باشد. این الگوریتم مسیری که بهینه‌ی کلی نیست را ابتدا به کاربر می‌دهد و سپس در مدت زمانی که فرصت دارد به اصلاح آن اقدام می‌کند. در این حالت کاربر می‌تواند حرکت خود را آغاز کند و در بین راه مسیر او رفته‌رفته اصلاح می‌شود. این در حالی است که الگوریتم توانایی تعامل با هزینه‌های متغیر را هم دارد و در صورت بروز تغییر هزینه، مسیر را اصلاح می‌کند.

روش‌های ابتکاری دیگری برای هرس کردن گراف شبکه راه نیز ذکر شدند که از آن جمله می‌توان به روش ALT اشاره کرد که در این الگوریتم گره‌هایی به‌عنوان گره شاخص به الگوریتم شناسانده می‌شوند و از نامساوی مثلثی عددی برای استفاده در مقدار پیش‌بینی فاصله بین گره تا مقصد در الگوریتم *A استخراج می‌شود. این الگوریتم تا حدودی توانسته بود الگوریتم *A را بهبود دهد. الگوریتم REACH نیز برای هرس کردن گره‌هایی که باید در الگوریتم مسیریابی بررسی شوند، ذکر شد که در ترکیب آن با الگوریتم ALT، الگوریتم REAL توسعه یافته بود و نتایج آن با الگوریتم ALT در بخش ۳-۱-۲-۵ بررسی شده بود.

روش جستجوی میان‌بر نیز روش دیگری بود که برای هرس کردن لینک‌هایی که باید بررسی بشوند مورد استفاده قرار گرفته بود. این الگوریتم با جایگزینی لینک‌هایی که در یک جهت قرار دارند و میزان reach بالایی دارند، با یک یا چند لینک توانسته بود به‌خوبی سرعت محاسبات را هم در مرحله پیش‌پردازش و هم در مرحله مسیریابی افزایش دهد. مقایسه این روش با الگوریتم Dijkstra دو جهته در جدول ۱ آورده شده است.

۵- مسائل پیچیده و پیشرفته در مسیریابی

مسائل جهان واقعی بسیار پیچیده‌تر از مدل‌هایی هستند که الگوریتم‌های ذکر شده در این مقاله توان حل آن را داشته باشند. در دنیای واقعی هزینه‌ی در نظر گرفته شده برای لینک‌ها تابع بسیاری از فاکتورهای دیگر هستند. به‌طور مثال زمان سفر در یک خیابان نه تنها فقط به طول بستگی ندارد بلکه به معیارهای بسیاری همچون هندسه راه، تعداد خودروی عبوری، اتفاقات غیرمترقبه، وضعیت آب‌وهوایی، زمان و دیگر معیارها وابسته است. همچنین، فرض کردن تنها یک معیار برای وزن دهی به لینک‌ها بسیار دور از واقعیت است، زیرا افراد مسیر رفت-وآمد خود را تنها با یک معیار انتخاب نمی‌کنند و این انتخاب وابسته به معیارهایی همچون زمان سفر، طول سفر، وضعیت ترافیکی، مصرف سوخت، امنیت، منظره قابل دید در مسیر و غیره است. از طرف دیگر، تمامی راه‌ها برای همه‌ی افراد قابل استفاده نیستند. به‌عنوان مثال یک شهروند عادی نمی‌تواند در شرایط عادی از خطوط ویژه اتوبوس و تاکسی استفاده کند، اما این امکان برای خودروهای پلیس، آمبولانس و آتش‌نشانی وجود دارد.

مسیریابی در شبکه‌های حمل‌ونقل عمومی نیز پیچیدگی‌های خاص خود را دارد. امکان انجام مسیریابی در یک شبکه‌ی چند ساختی[۱] که شبکه‌ای شامل شبکه خطوط

[۱] Multi-Modal

اتوبوس، تاکسی و مترو است، یکی از حوزه‌های مسیریابی پیشرفته است. این شبکه‌ها زمانمند هستند به این معنا که در مسیریابی بر روی این شبکه‌ها باید به جداول زمانی حرکت وسایل نقلیه‌ی عمومی دسترسی داشت تا بتوان به کاربر مسیری قابل اطمینان را ارائه داد [۱۱].

امروزه با افزایش دستگاه‌های همراه، باید امکان انجام مسیریابی بر روی دستگاه‌های ضعیف‌تر در مدت زمانی بهینه نیز ممکن باشد، پس باید هرچه بیشتر با کاهش پیچیدگی الگوریتم‌هایی که استفاده می‌شوند، امکان حرکت به سمت مسائل دنیای واقعی را افزایش داد [۱۱].

۶- نتیجه‌گیری

در این مقاله روش‌های مسیریابی قطعی و ابتکاری در سیستم اطلاعات مکانی مورد بررسی قرار گرفت که به‌طور کلی در ۶ بخش جمع‌بندی شد. بخش اول مقدمه در رابطه با مفاهیم الگوریتم‌های مسیریابی و سیستم اطلاعات مکانی توضیحاتی ذکر شد. در بخش دوم الگوریتم‌های قطعی و در بخش سوم الگوریتم‌های ابتکاری مورد بررسی قرار گرفتند. با مقایسه‌ای که در جدول ۲ انجام شد می‌توان دید که از بین الگوریتم‌ها، الگوریتم A* بدون استفاده از هیچ پیش پردازشی عملکرد بهتری در میان الگوریتم‌های تک منبع دارد و در بین الگوریتم‌های تمام-زوج، الگوریتم Floyd-Warshall پیچیدگی محاسباتی کمتری بدون انجام هیچ پیش‌پردازشی دارد.

برای اینگونه الگوریتم‌ها روش‌های دیگری نیز برای بالا بردن سرعت استفاده می‌شود. از آن جمله روش‌های جستجوی دو جهته و سلسله مراتبی به نظر می‌رسد که می‌توانند با توجه به شرایط مسئله، سرعت الگوریتم‌های مسیریابی را تا حد زیادی بهبود دهند.

با انجام پیش‌پردازش (به طور مثال در الگوریتم ALT، REAL و میان‌بر)، مسیریابی که در مرحله‌ی بعدی انجام می‌شود، با سرعت بسیار بیشتری انجام خواهد شد. در پایان با بیان مسائل پیچیده و نزدیک به دنیای واقعی مشاهده شد که الگوریتم‌های ذکر شده برای اینکه بتوانند پاسخ‌گوی نیاز مسائل دنیای واقعی باشند باید بهبود پیدا کنند.

جدول۲- بررسی اجمالی الگوریتم‌های مسیریابی قطعی و ابتکاری

الگوریتم	پیچیدگی محاسبات	نوع درخت	وزن منفی	دور منفی	روش حل مسئله
Dijkstra	$O(\|V\|^2)$	Single-Source	خیر	خیر	LS
A*	$O(\|E\|)$	Single-Source	خیر	خیر	LS
IDA* [17]	$O(2^{2\|V\|})$	Single-Source	خیر	خیر	LS
Bellman-Ford	$O(\|V\| \times \|E\|)$	Single-Source	بله	خیر	LC
Floyd-Warshall	$O(\|V\|^3)$	All-pairs	بله	خیر	LC
Johnson	$O(\|V\|^2 \log\|V\| + \|V\|\|E\|)$	All-pairs	بله	خیر	LC
Bi-directional Dijkstra [18]	$O(k^m)$	Single-Source	خیر	خیر	LS

مراجع

[1] K. Magzhan and H. Mat Jani, "A Review And Evaluations Of Shortest Path Algorithms," INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH, 2013.

[2] S. Hougardy, "The Floyd–Warshall algorithm on graphs with negative cycles," Information Processing Letters, vol. 110, pp. 279-281, 2010.

[3] P. E. Black, "Johnson's algorithm," Dictionary of Algorithms and Data Structures, 17 December 2004. [Online]. Available: https://xlinux.nist.gov/dads//HTML/johnsonsAlgorithm.html. [Accessed 2016 February 20].

[4] L. Fu, D. Sun and L. Rilett, "Heuristic shortest path algorithms for transportation applications: State of the art," Computer and Operation Research, vol. 33, pp. 3324-3343, 2006.

[5] F. L, "Real-time vehicle routing and scheduling in dynamic and stochastic traffic networks," Ph.D. Dissertation, Vols. University of Alberta, Edmonton, Alberta, 1996.

[6] D. Ferguson and . A. Stentz, "The Delayed D* Algorithm for Efficient Path Replanning," in Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, 2005.

[7] A. Stentz, "The Focussed D* Algorithm for Real-Time Replanning," IJCAI, vol. 95, 1995.

[8] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz and S. Thrun, "Anytime Dynamic A*: An Anytime, Replanning Algorithm," in ICAPS, 2005.

[9] M. Likhachev, "ARA*: Anytime A* with Provable Bounds on Sub-Optimality," Advances in Neural Information Processing Systems, 2003.

[10] T. OHSHIMA, "A Landmark Algorithm for the Time-Dependent Shortest path problem," Department of Applied Mathematics and Physics Graduate School of Informatics Kyoto University, 2006.

[11] R. Geisberger, "Advanced Route Planning in Transportation Networks," Dissertation, 2011.

[12] H. K. a. R. F. W. Andrew V. Goldberg, "Better Landmarks within Reach," in 6th International Workshop, WEA, Rome, Italy, 2007.

[13] N. JAT, "Finding the shortest route between two points in a network," Computer Journal , vol. 9, p. 275, 1966.

[14] I. POHL, "Bi-directional search," Machine Intelligence, vol. 6, pp. 127-140, 1971.

[15] D. JF and P. Nelson, "Improving search efficiency using possible subgoals," Mathematical and Computer Modelling, vol. 22, p. 397–414, 1995.

[16] A. V. H. K. a. R. F. W. Goldberg, "Reach for a: Efficient point-to-point shortest path algorithms," in Proceedings of the Meeting on Algorithm Engineering & Expermiments, 2006.

[17] Mahanti, A, Pal, A.K., Ghosh, S. and Kanal, L.N., "Performance of A* and IDA*-a worst case analysis," in IEEE, Kansas City, MO, Applied Computing, 1991., [Proceedings of the 1991] Symposium on.

[18] M. Dramski, "Bi-directional search in route planning in navigation," Zeszyty Naukowe/Akademia Morska w Szczecinie, vol. 39, no. 111, pp. 57-62, 2014.

[19] A. Car and A. Frank, "General principles of hierarchical spatial reasoning—the case of way-finding," in Proceedings of the sixth international symposium on spatial data handling, 1994.