حل مسئله درخت اشتاینر با استفاده از اتوماتاهای یادگیر

سمیرا نوفرستی محمدرضا میبدی

آزمایشگاه سیستمهای نرم افزاری
دانشکده مهندسی کامپیوتر و فناوری اطلاعات
دانشگاه صنعتی امیرکبیر
تهران ایران
Samira_noferesti@yahoo.com, meybodi@ce.aut.ac.ir

چکیده: در این مقاله ابتدا یک الگوریتم تکرارشونده مبتنی بر اتوماتاهای یادگیر برای حل مسئله اشتاینر ایستا پیشنهاد می‌شود و سپس با استفاده از این الگوریتم، الگوریتم دیگری برای حل مسئله اشتاینر پویا ارایه می‌گردد. در الگوریتم حل مسئله اشتاینر ایستا، هر گره از گراف به یک اتوماتای یادگیر مجهز است. هر اتوماتای یادگیر دارای دو عمل می‌باشد که حضور یا عدم حضور گره متناظر در درخت اشتاینر را مشخص می‌کند. در هر تکرار از الگوریتم اتوماتاهای یادگیر به صورت همزمان فعال شده و یک عمل را انتخاب می‌کنند. سپس بر روی گره‌هایی که اتوماتاهای یادگیر آنها عمل حضور در درخت اشتاینر را انتخاب کرده‌اند یک درخت پوشای کمینه ایجاد می‌شود. با توجه به هزینه درخت بدست آمده از هر تکرار، بردار احتمالات اتوماتاهای یادگیر بروز می‌شود. این روند به دفعات تکرار می‌گردد و در خاتمه الگوریتم، درخت بدست آمده به عنوان جواب نهایی انتخاب می‌شود. برای حل مسئله اشتاینر پویا در ابتدا الگوریتم ایستا بر روی گراف قبل از تغییرات اجرا و بردارهای احتمالهای اقدامهای اتوماتاهای یادگیر محاسبه می‌شود. بعد از انجام تغییرات در گراف (حذف کردن و یا اضافه کردن گره به گراف) الگوریتم حل مسئله اشتاینر ایستا با در نظر گرفتن آخرین مقادیر بردارهای احتمالهای اقدامهای اتوماتاهای یادگیر اجرا می‌شود. از این طریق با تعداد تکرارهای کمتری درخت اشتاینر مورد نظر تعیین می‌شود. نتایج شبیه‌سازی‌های انجام گرفته کارایی الگوریتم پیشنهادی را هم از لحاظ کیفیت جوابهای تولید شده و هم از لحاظ سرعت همگرایی به جواب در مقایسه با الگوریتمهای گزارش شده نشان می‌دهد.

کلمات کلیدی: درخت اشتاینر ایستا، درخت اشتاینر پویا، اتوماتاهای یادگیر، مسایل مشکل

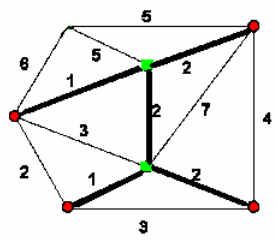
۱- مقدمه

برای پیاده‌سازی انتقال چندپخشی در شبکه‌های کامپیوتری ایستا و شبکه‌های کامپیوتری سیار[1] نیاز به ساخت درختهای چندپخشی می‌باشد. یکی از متداولترین راههای ساخت درختهای چندپخشی استفاده از الگوریتمهای ساخت درخت اشتاینر[2] در گراف است. در شبکه‌های سیار، توپولوژی شبکه در طی زمان بدلیل اضافه و یا حذف شدن گره‌ها تغییر می‌کند که در این شرایط مسئله درخت اشتاینر پویا مطرح می‌شود. در این موارد هدف یافتن یک درخت اشتاینر بعد از تغییرات انجام گرفته در گراف می‌باشد. تغییرات می‌تواند در قسمتهای مختلف گراف بطور همزمان ایجاد شود. ساخت درختهای چندپخشی در شبکه‌های سیار به مراتب دشوارتر از ساخت این درختها در شبکه‌های ایستا می‌باشد. یک الگوریتم کارا برای ساخت درختهای چندپخشی در شبکه‌های سایر بایستی بتواند درختهای مناسب را سریع تولید کند حتی اگر چند تغییر همزمان در گروه چندپخشی رخ دهد. تلاشهای زیادی در جهت ساخت درختهای اشتاینر در شبکه انجام شده است. اگرچه این الگوریتمها درختهای اشتاینر نزدیک به بهینه تولید می‌کنند ولی از سرعت همگرایی مناسبی برخوردار نیستند.

مسئله درخت اشتاینر در گراف به صورت زیر تعریف می‌شود: گراف $G(V,E)$ را در نظر بگیرید که V مجموعه گره‌ها و $E \subseteq V*V$ مجموعه یالهای گراف را نشان می‌دهد. به هر یال (i,j) در گراف یک هزینه $c(i,j)$ نسبت داده می‌شود. یک زیرمجموعه از گره‌ها به نام T نیز به عنوان مجموعه ترمینالها تعریف می‌شود. هدف مسئله اشتاینر یافتن درختی در گراف است که مجموعه ترمینالها را در برگیرد و هزینه آن حداقل باشد. این درخت، درخت اشتاینر کمینه [3](SMT) نامیده می‌شود. اگر درخت حاصل فقط شامل ترمینالها باشد، در این صورت درخت، درخت پوشای کمینه[4]

[1] Mobile Networks
[2] Steiner tree
[3] Steiner Minimum Tree
[4] Minimum Spanning Tree

نامیده می‌شود و الگوریتمهای چندجمله‌ای متعددی مانند الگوریتم پریم[5] برای حل آن وجود دارد. با این وجود در حالت کلی برای کاهش هزینه درخت از گره‌های غیرترمینال نیز استفاده می‌شود. این گره‌ها نقاط اشتاینر نامیده می‌شوند. مثالی از درخت اشتاینر در یک گراف نمونه در شکل ۱ نشان داده شده است.



شکل ۱: درخت اشتاینر در گراف

دو نوع کلی مسئله اشتاینر اقلیدسی[6] و مسئله اشتاینر افقی-عمودی[7] وجود دارد. در هر دو نوع هدف یافتن کم‌هزینه‌ترین درخت متصل کننده تعدادی نقطه در صفحه است. تفاوت این دو نوع در معیار اندازه‌گیری هزینه یالها می‌باشد. در مسئله اشتاینر اقلیدسی هزینه یک یال برابر فاصله اقلیدسی دو سر آن است. فاصله اقلیدسی دو نقطه $p_1=(x_1,y_1)$ و $p_2=(x_2,y_2)$ برابر $d_{p1,p2}=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$ می‌باشد. در مسئله اشتاینر افقی-عمودی اتصال نقاط تنها از طریق یالهای افقی یا عمودی امکان‌پذیر است. در این فضا از معیار L_2 برای محاسبه فاصله دو نقطه استفاده می‌شود. در این معیار فاصله دو نقطه p_1 و p_2 با استفاده از رابطه $d_{p1,p2}=|x_1-x_2|+|y_1-y_2|$ محاسبه می‌شود. البته در کاربردهای عملی نیاز به یک تابع هزینه می‌باشد که در هیچکدام از دو شکل فوق نمی‌گنجد. بطور مثال در شبکه‌های کامپیوتری معیار بهینه بودن می‌تواند یکی از پارامترهای کیفیت سرویس مانند تأخیر، هزینه انتقال یا بار ترافیک گره‌ها باشد.

به دلیل اهمیت مسئله درخت اشتاینر کمینه تاکنون الگوریتمهای متعددی برای حل آن گزارش شده است. این الگوریتمها را می‌توان به دو گروه کلی تقسیم کرد: الگوریتمهای دقیق[8] و الگوریتمهای تقریبی[9]. با توجه به اینکه مسئله درخت اشتاینر یک مساله NP_Complete می‌باشد الگوریتمهای دقیق که معمولاً از روشهای برنامه‌نویسی پویا[10] و برش‌وانشعاب[11] استفاده می‌کنند[۱] در بدترین حالت دارای پیچیدگی نمایی هستند و برای استفاده در کاربردهای عملی مناسب نمی‌باشند. به همین دلیل الگوریتمهای تقریبی متعددی برای حل مسئله اشتاینر در گراف گزارش شده است[۱، ۲، ۳ و ۴]. بسیاری از الگوریتمهای تقریبی گزارش شده برای حل مسئله درخت اشتاینر، درخت اشتاینر را از طریق ساختن یک درخت پوشای کمینه می‌سازند. برای به دست آوردن درخت پوشای کمینه به این صورت عمل می‌شود که ابتدا به کمک گراف G گراف G' طوری ساخته می‌شود که وزن هر یال آن معادل وزن کوتاهترین مسیر بین دو سر یال در گراف G باشد. سپس الگوریتم ساخت درخت پوشای کمینه بر روی گراف جدید اجرا می‌گردد. در پایان نیز درخت حاصل از طریق حذف گرههای غیرترمینالی که برگ هستند هرس می‌شود [۲].

تلاشهای زیادی نیز در جهت حل تقریبی مسئله اشتاینر پویا انجام گرفته است. الگوریتم حریصانه یکی از اولین الگوریتمهای ارائه شده در این زمینه است. در این روش فرض شده است که هیچ گرهی بعد از آشکار شدن حذف نمی‌شود. روش پیشنهادی به این صورت عمل می‌کند که گره جدید از طریق کوتاهترین مسیر به نزدیکترین گره در درخت متصل می‌شود. در [۵] نشان داده شده است که طول درختهای به دست آمده از این روش کمتر از $\frac{4}{3}$ برابر حالت بهینه می‌باشد. در [۶] روشی به نام DGA[12] بر اساس همین ایده معرفی شده است که علاوه بر درخواستهای افزودن گره‌های جدید، به درخواستهای حذف گره‌های موجود نیز پاسخ می‌دهد. در [۷] یک الگوریتم حریصانه وزن‌دار (WGA)[13] برای کاهش اثر منفی درخواستهای حذف در الگوریتم حریصانه معرفی شده است. ایده این الگوریتم به این صورت است که گره جدیدی که به درخت اضافه می‌شود نباید در فاصله خیلی زیاد از مبدأ قرار گیرد.

از جمله الگوریتمهای تقریبی برای حل مسئله اشتاینر الگوریتمهای تکرارشونده[14] هستند. در این روشها رسیدن به یک پاسخ بهینه تضمین نمی‌شود اما در اغلب موارد جوابهای تقریبی قابل‌قبولی تولید می‌کنند. از جمله الگوریتمهای تکرارشونده می‌توان به الگوریتم ژنتیکی [۸] و الگوریتم کلونی مورچه‌ها [۱۳] اشاره کرد. در این مقاله ابتدا یک الگوریتم تکرارشونده مبتنی بر اتوماتاهای یادگیر برای حل مسئله اشتاینر ایستا پیشنهاد می‌شود و سپس با استفاده از این الگوریتم، الگوردتم دیگری برای حل مسئله اشتاینر پویا ارایه می‌گردد. در الگوریتم حل مسئله اشتاینر ایستا، هر گره از گراف به یک اتوماتای یادگیر مجهز است. هر اتوماتای یادگیر دارای دو عمل می‌باشد که حضور یا عدم حضور گره متناظر در درخت اشتاینر را مشخص می‌کند. در هر تکرار از الگوریتم اتوماتاهای یادگیر به صورت همزمان فعال شده و یک عمل را انتخاب می‌کنند. سپس

[5] Prim
[6] Euclidean
[7] Rectilinear
[8] Exact
[9] Approximate
[10] Dynamic programming
[11] Branch and cut
[12] Dynamic Greedy Algorithm
[13] Weighted Greedy Algorithm
[14] Iterative

بر روی گره‌هایی که اتوماتاهای یادگیر آنها عمل حضور در درخت اشتاینر را انتخاب کرده‌اند یک درخت پوشای کمینه ایجاد می‌شود. با توجه به هزینه درخت بدست آمده از هر تکرار، بردار احتمالات اتوماتاهای یادگیر بروز می‌شود. این روند به دفعات تکرار می‌گردد و در پایان بهترین درخت بدست آمده به عنوان جواب نهایی انتخاب می‌شود. نتایج شبیه‌سازی‌های انجام گرفته کارایی الگوریتم پیشنهادی را هم از لحاظ کیفیت جوابهای تولید شده و هم از لحاظ سرعت همگرایی به جواب در مقایسه با تعدادی از الگوریتمهای گزارش شده نشان می‌دهد.

ادامه مقاله بدین صورت سازماندهی شده است. در ادامه ابتدا در بخش ۲ اتوماتاهای یادگیر به صورت اجمالی معرفی می‌گردد. در بخش ۳ الگوریتم پیشنهادی برای حل مسئله درخت اشتاینر ایستا و نتایج شبیه‌سازی‌های انجام گرفته ارائه می‌شود. در بخش ۴ الگوریتم پیشنهادی برای حل به مسئله درخت اشتاینر پویا نتایج شبیه‌سازی‌های ارائه می‌شود. بخش پایانی مقاله نتیجه‌گیری می‌باشد.

۲- اتوماتاهای یادگیر[15]

اتوماتای یادگیر [۹] یک مدل انتزاعی است که می‌تواند تعداد محدودی عمل را انجام دهد. هر عمل انتخاب شده توسط محیطی تصادفی ارزیابی می‌گردد و پاسخی به اتوماتای یادگیر داده می‌شود. اتوماتای یادگیر از این پاسخ استفاده نموده و عمل خود را برای مرحله بعد انتخاب می‌کند. محیط تصادفی را می‌توان با سه‌تایی $E \equiv \{\alpha, \beta, c\}$ تعریف نمود که

$\alpha \equiv \{\alpha_1, \alpha_2, \cdots, \alpha_r\}$ مجموعه ورودی‌ها

$\beta \equiv \{\beta_1, \beta_2, \cdots, \beta_r\}$ مجموعه خروجی‌ها

$c \equiv \{c_1, c_2, \cdots, c_r\}$ مجموعه احتمال‌های جریمه شدن

می‌باشند. هرگاه β_i دو مقداری باشد $\beta_i = 1$ به عنوان جریمه و $\beta_i = 0$ به عنوان پاداش در نظر گرفته می‌شود. c_i احتمال اینکه عمل α_i نتیجه نامطلوب داشته باشد می‌باشد. در محیط پایدار[16] مقادیر c_i بدون تغییر باقی می‌مانند، حال آنکه در محیط ناپایدار[17] این مقادیر در طی زمان تغییر می‌کنند. شکل ۲ ارتباط بین اتوماتای یادگیر و محیط را نشان می‌دهد.

[15] Learning Automata
[16] Stationary
[17] Non-Stationary

شکل ۲: ارتباط بین اتوماتای یادگیر و محیط

اتوماتاهای یادگیر به دو گروه اتوماتای یادگیر با ساختار ثابت[18] و اتوماتای یادگیر با ساختار متغیر[19] تقسیم می‌شوند. در ادامه به شرح مختصری درباره اتوماتاهای یادگیر با ساختار متغیر که در این مقاله استفاده شده است می‌پردازیم.

اتوماتای یادگیر با ساختار متغیر: اتوماتای یادگیر با ساختار متغیر توسط ۵ تایی $LA \equiv \{\alpha, \beta, p, T, c\}$ نشان داده می‌شود که

$\alpha \equiv \{\alpha_1, \alpha_2, ..., \alpha_r\}$ مجموعه عملهای اتوماتای یادگیر

$\beta \equiv \{\beta_1, \beta_2, ..., \beta_m\}$ مجموعه ورودی‌های اتوماتای یادگیر

$p \equiv \{p_1, p_2, ..., p_r\}$ بردار احتمال انتخاب عملها

$T \equiv p(n+1) = T[\alpha(n), \beta(n), p(n)]$ الگوریتم یادگیری

$c \equiv \{c_1, c_2, ... c_r\}$ احتمال جریمه شدن هر عمل

می‌باشند. اگر در اتوماتای یادگیر عمل α_i در مرحله nام انتخاب شود و پاسخ مطلوب از محیط دریافت نماید، احتمال $p_i(n)$ افزایش یافته و سایر احتمال‌ها کاهش می‌یابند و برای پاسخ نامطلوب احتمال $p_i(n)$ کاهش یافته و سایر احتمال‌ها افزایش می‌یابند. در هر حال، تغییرات به گونه‌ای صورت می‌گیرد تا حاصل جمع $p_i(n)$ ها همواره ثابت و مساوی یک باقی بماند.

الف- پاسخ مطلوب

$$p_i(n+1) = p_i(n) + a[1 - p_i(n)]$$
$$p_j(n+1) = (1-a)\, p_j(n) \qquad \forall j \quad j \neq i$$

ب- پاسخ نامطلوب

$$p_i(n+1) = (1-b)\, p_i(n)$$
$$p_j(n+1) = \frac{b}{r-1} + (1-b) p_j(n) \quad \forall j \quad j \neq i$$

در روابط فوق، a پارامتر پاداش و b پارامتر جریمه می‌باشد. با توجه به مقادیر a و b سه حالت مختلف را می‌توان در نظر گرفت. زمانیکه a و b با هم برابر باشند، الگوریتم را L_{RP}[20] می‌نامند. زمانیکه b از a خیلی کوچکتر باشد، الگوریتم را $L_{R\varepsilon P}$[21] و زمانیکه b مساوی صفر باشد، الگوریتم را L_{RI}[22] می‌نامند. برای مطالعه بیشتر در رابطه با اتوماتاهای یادگیر با ساختار ثابت و متغیر می‌توان به [۱۰ و ۱۴] مراجعه نمود.

۳- حل مسئله اشتاینر ایستا با استفاده از اتوماتاهای یادگیر

[18] Fixed Structure
[19] Variable Structure
[20] Linear Reward Penalty
[21] Linear Reward Epsilon Penalty
[22] Linear Reward Inaction

در الگوریتم پیشنهادی که آنرا *NLA*[23] می‌نامیم، گراف مسئله با یک شبکه از اتوماتاهای یادگیر مدل می‌شود. هر گره از گراف به یک اتوماتای یادگیر با دو عمل حضور گره در درخت اشتاینر و عدم حضور گره در درخت اشتاینر مجهز است. در ابتدای الگوریتم احتمال انتخاب این اعمال مساوی و برابر ۰/۵ می‌باشد. احتمال حضور گره‌های ترمینال همواره برابر یک می‌باشد.

در هر تکرار از الگوریتم همه اتوماتاهای یادگیر به صورت همزمان فعال شده و هر یک از آنها طبق بردار احتمالات انتخاب اعمال خود یکی از دو عمل خود را انتخاب می‌کند. سپس با استفاده از الگوریتم پریم یک درخت پوشای کمینه بر روی گره‌هایی که انتخاب شده‌اند ایجاد می‌شود. در پایان درخت حاصل هرس می‌شود به این صورت که گره‌های غیرترمینالی که در برگهای درخت واقع شده‌اند، حذف می‌گردند. در این روش با احتمال $1-q$ هر اتوماتای یادگیر گره j را با احتمال $p(j)$ انتخاب و با احتمال q گره j طبق رابطه $j = argmax\{p(j)\}$ انتخاب می‌کند. برای تعیین مقدار پارامتر q از یک اتوماتای یادگیر دیگر استفاده می‌شود [۱۵].

در هر تکرار از هزینه درخت بدست آمده برای ارزیابی عمل انتخابی اتوماتاها استفاده می‌شود. اگر هزینه درخت فعلی از هزینه بهترین درخت بدست آمده تا این مرحله کمتر باشد، اعمال انتخاب شده توسط اتوماتاهای یادگیر از طریق افزایش احتمال انتخاب آنها طبق رابطه زیر پاداش داده می‌شود. فرمول یادگیری برای پاداش عمل انتخابی بصورت زیر می‌باشد:

$$p(t+1) = p(t) + \theta.a.(1 - p(t))$$

که $\theta = |\frac{f(c) - c_{\min}}{f(c)}|$ و $f(c)$ هزینه درختی است که در تکرار tام تولید شده و C_{min} هزینه بهترین درخت بدست آمده تا این مرحله است. a پارامتر پاداش نامیده می‌شود.

اگر هزینه درخت فعلی از هزینه بهترین درخت اشتاینر بدست آمده تاکنون بیشتر باشد اعمال انتخاب شده توسط اتوماتاهای یادگیر گره‌های شرکت کننده در بهترین درخت حاصل شده تا این مرحله پاداش و اعمال انتخاب شده توسط اتوماتاهای یادگیر دیگر گره‌ها جریمه می‌شوند. برای جریمه عمل انتخابی از فرمول زیر استفاده می‌شود.

$$p(t+1) = p(t) - \theta.b.p_i(n)$$

که b پارامتر جریمه نامیده می‌شود. برای بهبود نتایج حاصل شده، در بعضی از تکرارها بر روی درختهای بدست آمده عمل کاهش انجام می‌گیرد. در این عمل برای هر یال از درخت به صورت زیر عمل می‌شود:

۱. با حذف یال از درخت، دو زیردرخت مجزا حاصل می‌شود.

۲. با اجرای الگوریتم جستجوی عمق اول نقاط اشتاینری که در برگهای دو زیردرخت واقع شده‌اند، حذف می‌شوند.

۳. با استفاده از الگوریتم دایکسترا[24] کوتاهترین مسیر بین گرههای دو زیردرخت مشخص می‌شود.

۴. اگر مجموع هزینه‌های دو زیردرخت و هزینه مسیر یافت شده کمتر از هزینه درخت قبلی باشد دو زیردرخت از طریق کوتاهترین مسیر متصل شده و درخت حاصل به عنوان درخت فعلی در نظر گرفته می‌شود.

از آنجا که عمل کاهش وقت‌گیر است و منجر به افزایش زمان‌اجرای الگوریتم می‌شود، این عمل می‌تواند بر روی تعداد کمی از درختهای بدست آمده انجام شود. شبه کد الگوریتم *NLA* در شکل ۳ نشان داده شده است.

جدول ۱: نتایج الگوریتم *NLA* با اتوماتاهای یادگیر L_{RP} و L_{ReP}

شماره گراف	هزینه بهینه	L_{RP} %	$L_{R\epsilon P}$ %
۱	۸۲	۰	۰
۲	۸۳	۰	۰
۳	۱۳۸	۰	۰
۴	۵۹	۲/۵۴	۰
۵	۶۱	۱/۶۴	۰
۶	۱۲۲	۰	۰
۷	۱۱۱	۵/۴۹	۰
۸	۱۰۴	۲/۸۸	۰
۹	۲۲۰	۰	۰
۱۰	۸۶	۲/۰۹	۰

نتایج شبیه سازی: در آزمایش اول الگوریتم پیشنهادی با الگوریتمهای یادگیری L_{RP} و $L_{R\epsilon P}$ پیاده‌سازی شده است. در L_{RP} مقدار پارامترهای a و b برابر ۰/۳ و در $L_{R\epsilon P}$ مقدار پارامتر a برابر ۰/۳ و مقدار پارامتر b برابر ۰/۰۱ در نظر گرفته شده است. نتایج میانگین ۱۰ بار اجرای این الگوریتمها با ۵۰۰ تکرار در هر اجرا در جدول ۱ ارائه شده است. این الگوریتمها بر روی گرافهای مجموعه B از دسته مسائل بسیلی[25] آزمایش شده‌اند [۱۰].

[23] Network of Learning Automata

[24] Dijkstra

[25] Beasly

ستون اول جدول شماره گراف در مجموعه B مسائل بیسلی، ستون دوم هزینه درخت اشتاینر بهینه برای هر گراف و ستونهای بعدی درصد خطای نسبی حاصل از اجرای الگوریتم پیشنهادی با اتوماتاهای یادگیر مختلف را نشان می‌دهد. برای محاسبه خطای نسبی از رابطه زیر استفاده می‌شود:

$$relative-error = \frac{C - C_{opt}}{C_{opt}}$$

که C هزینه درخت اشتاینر حاصل از الگوریتم و C_{opt} هزینه درخت اشتاینر بهینه در گراف است. عمل کاهش تنها در ۳۰ تکرار انتهایی انجام گرفته است. همان‌طور که از جدول ۱ مشخص است استفاده از اتوماتای یادگیر $L_{R\varepsilon P}$ منجر به نتایج بهتری می‌شود.

```
algorithm NLA
 Input:   G(V,E,c), T⊆V
 Output:  tree edges and tree cost

 MAX_ITR              // number of iterations
 REDUCTION_No         // number of reductions
 a                    // reward parameter
 b                    // penalty parameter
 selected_nodes       // fixed length bit string, one bit for each node in graph
 Cmin                 // cost of the best tree found so far

 begin
     itr=0                          // loop counter
     T=•
     while (itr < MAX_ITR) do{
        for each node p_i in graph do{
            action = the action choosen by LA_i in node  p_i
            if the action choosen exists in the tree then selected-nodes[itr] = true
            else selected_nodes[itr] = false
        }
        T = prim(selected-nodes)
        if (itr > MAX_ITR-REDUCTION_No) T = reduction(T)
        if (treeCos <= Cmin) then reward(T)
        else penalty(T)
        itr = itr+1
    }
end
 Procedure reward(T){
       teta = |(treeCost - Cmin) / treeCost|
       for each node j in T do
          p_j = p_j + teta * a * (1 - p_j)
       for each node j is not in T do
          p_j = p_j - teta * b *(1 - p_j)
 }
 Procedure penalty(T){
       teta=| (treeCost - Cmin) / treeCost|
       for each node j in T do
          p_j = p_j - teta * b * (1 - p_j)
       for each node j in best_tree do
          p_j = p_j + teta * a *(1 - p_j)
  }
Procedure Reduction(T){
     for each edge e in T do{
        -delete e from T and partition T into two parts
        -delete Steiner points that are placed in leaves
        -assume two parts as two nodes sets and find shortest path between them
        -if cost of this path is smaller than cost of T replace T with it
     }
     return T
  }
```

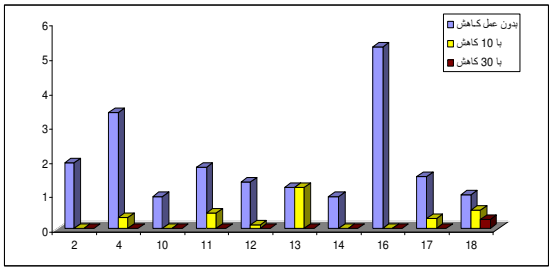
شکل۳- الگوریتم NLA برای حل مسئله اشتاینر

در جدول ۲ درصد خطای نسبی روش پیشنهادی با درصد خطای نسبی الگوریتم ACS[۲۶] که در [۱۳] گزارش شده و درصد خطای نسبی تعدادی از الگوریتمهای حل مسئله اشتاینر که در [۸] معرفی شده‌اند مقایسه شده است.

جدول ۲: مقایسه درصد خطای نسبی الگوریتمهای حل مسئله اشتاینر

شماره گراف	هزینه بهینه	SDG %	SPH %	ADH %	ACS %	NLA %
۱	۸۲	۰	۰	۰	۰	۰
۲	۸۳	۸۴/۳	۰	۰	۰	۰
۳	۱۳۸	۱/۴۵	۰	۰	۰	۰
۴	۵۹	۸/۴۳	۵/۰۸	۵/۰۸	۲/۷	۰
۵	۶۱	۴/۹۲	۰	۰	۰	۰
۶	۱۲۲	۴/۹۲	۳/۲۸	۱/۶۴	۱/۸	۰
۷	۱۱۱	۰	۰	۰	۰	۰
۸	۱۰۴	۰	۰	۰	۰	۰
۹	۲۲۰	۲۷/۲	۰	۰	۰/۰۵	۰
۱۰	۸۶	۱۳/۹۵	۴/۶۵	۴/۶۵	۳/۸۴	۰
۱۱	۸۸	۲/۲۷	۲/۲۷	۲/۲۷	۱/۵۹	۱/۸
۱۲	۱۷۴	۰	۰	۰	۰/۱۱	۰
۱۳	۱۶۵	۶/۰۶	۷/۸۸	۲/۲۴	۰	۴/۲
۱۴	۲۳۵	۱/۲۸	۲/۵۵	۰/۴۳	۰/۰۹	۰
۱۵	۳۱۸	۲/۲۰	۰	۰	۰	۰
۱۶	۱۲۷	۷/۸۷	۳/۱۵	۰	۰/۱۶	۱/۲
۱۷	۱۳۱	۳۴/۵	۳/۸۲	۳/۰۵	۱/۲۲	۱/۳
۱۸	۲۱۸	۴/۵۹	۱/۸۳	۰	۱/۴۲	۰

برای بررسی تأثیر عمل کاهش، الگوریتم پیشنهادی یکبار بدون عمل کاهش و بار دیگر با اجرای عمل کاهش بر روی تکرارهای نهایی الگوریتم (به تعداد ۱۰ و ۳۰) اجرا شده است. شکل ۵ درصد خطای نسبی الگوریتم NLA با تعداد عملهای کاهش متفاوت را نشان می‌دهد. همان طور که مشاهده می‌شود عمل کاهش باعث بهبود نتایج حاصل از الگوریتم می‌شود.



شکل۵: درصد خطای نسبی الگوریتم NLA با عملهای کاهش متفاوت

۴- حل مسئله اشتاینر پویا

در بخش قبل با استفاده از شبکه‌ای از اتوماتاهای یادگیر همکار روشی برای حل مسئله اشتاینر ایستا پیشنهاد شد. در الگوریتم حل مسئله اشتاینر پویا در ابتدا الگوریتم ایستا بر روی گراف قبل از تغییرات اجرا و بردارهای احتمالهای اقدامهای اتوماتاهای یادگیر محاسبه می‌شود. بعد از تغییرات (حذف و یا اضافه کردن گره به گراف)، الگوریتم درخت اشتاینر ایستا با در نظر گرفتن آخرین مقادیر بردارهای احتمالهای اقدامهای اتوماتاهای یادگیر اجرا می‌شود. از این طریق با تعداد تکرارهای کمتری درخت اشتاینر مورد نظر تعیین می‌شود.

نتایج شبیه سازی: جدول ۳ نتیجه الگوریتم *NLA* را بر روی گرافهای ۱،۳،۹، ۱۵ و ۱۸ از مجموعه *B* از دسته مسائل بیسلی با تعداد تکرارهای ۱۰ و ۲۰ برای هر درخواست را نشان می‌دهد. در این آزمایشها، ابتدا الگوریتم ایستا بر روی گرافهای اصلی با تعداد تکرار ۵۰۰ اجرا شده است. سپس هر یک از این گرافها به میزان ۲٪، ۵٪ و ۱۰٪ تعداد گره‌های گراف تغییر داده شده و برای هر درخواست الگوریتم حل مسئله اشتاینر ایستا اجرا شده است. برای محاسبه درصد خطای نسبی، الگوریتم ایستا با تعداد تکرارهای زیاد بر روی گراف تغییر یافته اعمال و کم‌هزینه‌ترین درخت بدست آمده به عنوان جواب در نظر گرفته شده است.

جدول ۳-الف نتیجه اجرای الگوریتم *NLA* با تعداد تکرار ۲۰ برای هر درخواست را نشان می‌دهد. در این جدول متوسط درصد افت کیفیت (بدتر شدن هزینه کل درخت اشتاینر) ۰/۰۹ درصد ثبت شده است. جدول ۳-ب نتیجه اجرای الگوریتم را با تعداد تکرار ۱۰ برای هر درخواست نشان می‌دهد. در این مرحله کیفیت درخت اشتاینر به میزان کمی بدتر شده است. متوسط خطای نسبی بدست آمده ۰/۱۳ درصد است. در این مرحله نیز متوسط خطا بیشتر شده است ولی نتایج همچنان قابل‌مقایسه می‌باشد و اختلاف قابل توجهی دیده نمی‌شود. بنابراین جدول ۳ نشان می‌دهد که پس از هر تغییر یا تغییرات، می‌توان با استفاده از الگوریتم حل مساله درخت اشتاینر ایستا به شرطی که مقادیر بردار احتمالات اقدامها قبل از تغییرات در نظر گرفته شود، با تعداد تکرارهای بسیار کمتر از تعداد تکرارهای حالتی که مقادیر بردار احتمالات اقدامها به طور تصادفی انتخاب شده باشند به جواب مطلوب و نزدیک به بهینه رسید. ماهیت تصادفی الگوریتم باعث می‌شود نسبت به نوع تغییر حساسیت زیادی نداشته باشد.

[26] Ant Colony System

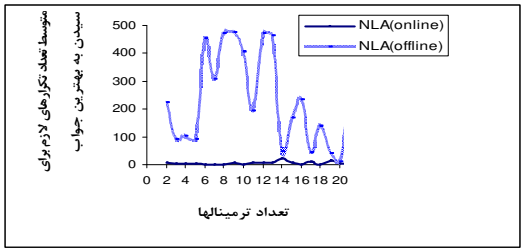
جدول ۳: درصد خطای نسبی وزن درخت اشتاینر پویا
با تعداد تکرارهای مختلف

الف) درصد خطای نسبی وزن درخت اشتاینر پویا با تعداد تکرار ۲۰

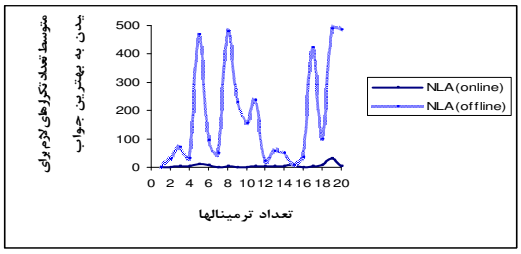
شماره گراف	تعداد گره	تغییر۲٪	تغییر۵٪	تغییر۱۰٪
۱	۵۰	۰	۰	۰
۳	۵۰	۰	۰	۰
۹	۷۵	۰	۰	۰
۱۵	۱۰۰	۰	۰	۰
۱۸	۱۰۰	۰/۴۴	۰	۰/۸۸

ب) درصد خطای نسبی وزن درخت اشتاینر پویا با تعداد تکرار ۱۰

شماره گراف	تعداد گره	٪۲	٪۵	٪۱۰
۱	۵۰	۰	۰	۰
۳	۵۰	۰	۰	۰
۹	۷۵	۰	۰	۰
۱۵	۱۰۰	۰	۰	۰/۰۶
۱۸	۱۰۰	۰/۸۸	۰	۰/۹۷



الف) گراف شماره ۲



ب) گراف شماره ۱۱

شکل ۶: متوسط تعداد تکرارهای لازم در الگوریتم NLA ایستا و پویا برای رسیدن به بهترین جواب برحسب تعداد ترمینالها

برای مقایسه زمان الگوریتم ایستا و پویا نمودار متوسط تعداد تکرارهای لازم برای رسیدن به بهترین جواب برحسب تعداد ترمینالها، برای گرافهای شماره ۲ و ۱۱ از مجموعه B مسائل بیسلی در شکل ۶ نشان داده شده است. منظور از بهترین جواب، کمترین هزینه بدست آمده از اجرای الگوریتم ایستا در تکرارهای بالاست. همان طور که در شکل دیده می‌شود در اکثر موارد الگوریتم *NLA* پویا نیاز به تعداد تکرارهای بسیار کمتری دارد.

۵- نتیجه‌گیری

در این مقاله ابتدا یک الگوریتم مبتنی بر اتوماتاهای یادگیر برای حل مسئله اشتاینر ایستا پیشنهاد شد و سپس این الگوریتم برای حل مسئله اشتاینر پویا استفاده گردید. نتایج بدست آمده از آزمایشها نشان داد که الگوریتم پیشنهاد شده در این مقاله برای حل هر دو مساله درخت اشتاینر ایستا و درخت اشتاینر پویا از کارایی بالایی برخوردار است. نتایج شبیه‌سازی‌های انجام گرفته کارایی الگوریتم پیشنهادی را هم از لحاظ کیفیت جوابهای تولید شده و هم از لحاظ سرعت همگرایی به جواب در مقایسه با تعدادی از الگوریتمهای گزارش شده نشان می‌دهد. به علاوه از آنجا که زمان اجرای این روش متناسب با تعداد گرههای گراف است برای گرافهای شلوغ (گراف با تعداد یالهای زیاد) سرعت همگرایی بسیار مناسبی دارد.

مراجع

[1] Adamatzky, A. *Computing in Nonlinear Media and Automata Collectives*. IOP Publishing, ISBN 0 7503 0751 X, 2001.

[2] Esbensen, H. Computing Near-Optimal Solutions to the Steiner Problem in a Graph Using a Genetic Algorithm. *Networks*, 26 (1995), 173-185.

[3] Hougardy, S., and Promel, H. J. *A 1.598 Approximation Algorithm for the Steiner Problem in Graphs. ACM-SIAM Symposium on Descrete Algorithm*. USA, (1999), 448-453.

[4] Diane, M., and Plesnik, J. Three New Heuristics for the Steiner Problem in Graphs. *Acta Math*., LX (1999), 105-121.

[5] Tsai, Y. T., Tang, Ch., and Chen, Y. Y. An Average Case Analysis of a Greedy Algorithm for the On-Line Steiner Tree Problem. *Computre Math. Applic*., 31, 11 (1996), 121-131.

[6] Chiu,L. K. *Comp 670K (Online Algorithms) Final Report On Steiner trees*. www.cs.ust.hk/faculty/rudolf/Courses/Online03/chiu_report.pdf, December 2003.

[7] Waxman,B. M. Routing of multipoint connections, *IEEE Journal on Selected Areas in Communications*. 6, 9 (Dec. 1998), 1611–1622.

[8] Ding, S., and Ishii, N. An Online Genetic Algorithm for Dynamic Steiner Tree Problem. *Symposium on Computational Geometry*, 337-343, 1995.

[9] Sutton, R.S., Barto, A.G., *Reinforcement Learning: An introduction*. MA: MIT Press, Cambridge,1998.

[10] Thathachar, M. A. L., Sastry P. S., Varieties of Learning Automata: An Overview. *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, 32, 6 (2002).

[11] Beasley,J. E. OR-Library: Distributing Test Problems by Electronic Mail. *Operational Research. SOC*., 41 ,11 (1990), 1096-1072.

[12] Rayward-smith, V. J., and Clare, A. On Finding Steiner vertices. *Networks*, 16, 283-294, 1986.

[۱۳] م. تشکری هاشمی، پ. ادیبی، ع. جهانیان، و ع. نورالـه، حل مسئله درخت اشتاینر پویا به کمک سیستم کولونی مورچه‌ها، **نهمین کنفرانس سالانه انجمن کامپیوتر ایران**، ۱۳۸۲.

[۱۴] م. ر. شیرازی، م. ر. میبدی، به کارگیری اتوماتای یادگیر در سیستمهای جندعامله همکار، **اولین کنفرانس بین المللی فناوری اطلاعات و دانش**، ص. ۳۳۸-۳۴۹، تهران، ۱۳۸۲.

[۱۵] ف. ابدالی، م. ر. میبدی، تطبیق پارامترهای الگوریتم کلونی مورچه‌ها با استفاده از اتوماتاهای یادگیر، **دهمین کنفرانس سالانه انجمن کامپیوتر ایران، مرکز تحقیقات مخابرات**، تهران، ایران، ۱۳۸۴.