بنام خدا

مریم رضوانی 9921160019

تمرین سری چهارم درس هوش مصنوعی

1) با استفاده از آروین فاصله مستقیم، عملکرد جستجوی *A را در مسأله رسیدن به Bucharest از Lugoj دنبال کنید. بعبارت دیگر، توالی گره هایی که الگوریتم در نظر می گیرد و مقادیر f,g,h را برای هرگره نشان دهید

گره هایی که در هر مرحله با توجه به کمترین هزینه بسط داده میشوند با رنگ زرد مشخص شده اند و در هرمرحله گره هایی ک از مرحله قبل بسط داده نشده را اضافه و طبق اون الگوریتم را سورت می کنیم. و گره هدف با رنگ سبز مشخص شده است.

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

1) حالت شروع L: 0+244=244
2) M: 70+241=311 , T: 111+329=440
3) L: 140+244=384 , D: 145+242=387 , T: 111+329=440
4) D: 145+242 =387, T: 111+329=440, M=210+241=451, T:251+329=580
5) C:265+160=425, T: 111+329=440, M=210+241=451, M: 220+241=461, T:251+329=580
6) T: 111+329=440, M=210+241=451, M: 220+241=461, P:403+100=503, T:251+329=580, R:411+193=604, D: 385+242=627
7) M=210+241=451,M:220+241=461,L:222+244=466,P:403+100=503, T:251+329=580, A:229+366=595, R:411+193=604, D: 385+242=627
8) M:220+241=461,L:222+244=466,P:403+100=503,L:280+244=524, D:285+242=527,T:251+329=580,A:229+366=595,R:411+193=604, D:385+242=627
9) L:222+244=466, P:403+100=503, L:280+244=524, D:285+242=527, L:290+244=534, D:295+242=537, T:251+329=580, A:229+366=595, R:411+193=604, D:385+242=627
10) P:403+100=503, L:280+244=524, D:285+242=527, M:292+241=533,

L:290+244=534, D:295+242=537, T:251+329=580, A:229+366=595,

R:411+193=604, D:385+242=627, T:333+329=662

11)B: 504+0 =504, L:280+244=524, D:285+242=527, M:292+241=533,

L:290+244=534, D:295+242=537, T:251+329=580, A:229+366=595,

R:411+193=604, D:385+242=627, T:333+329=662, R:500+193=693,

C: 541+160=701

2) الگوریتم مسیر آروینی یک جستجوی اول بهترین است که در آن تابع هدف $f(n) = (2-w)g(n) + wh(n)$ می باشد. برای چه مقادیری از w تضمین می شود که این الگوریتم بهینه باشد؟ اگر w=0 باشد، چه نوع جستجویی انجام می شود؟ اگر w=1 باشد چطور؟ اگر w=2 باشد چطور؟

اگر w بین 0 تا 2 باشد یعنی $0 \le w < 2$ کامل است. همچنین اگر w=0 باشد یعنی f(n) =2g(n) این حالت عملکرد جستجوی هزینه یکنواخت دارد. و اگر w=1 این همان جستجوی *A است. و w=2 یعنی f(n)=2h(n) جستجوی اول بهترین حریصانه را شامل می شود.و همچنین اگر w≤ 1 باشد تابع f(n) همواره کمتر از h(n) است بنابراین قابل قبول است.

3)اظهارات ذیل را اثبات کنید:

الف) جستجوی اول سطح، حالت خاصی از جستجوی هزینه یکنواخت است.

اگر هزینه تمام مراحل در جستجوی اول سطح یکسان باشد آنگاه جستجوی هزینه یکنواخت عملکردی شبیه جستجوی اول سطح خواهد داشت. g(n) ∝ depth(n)

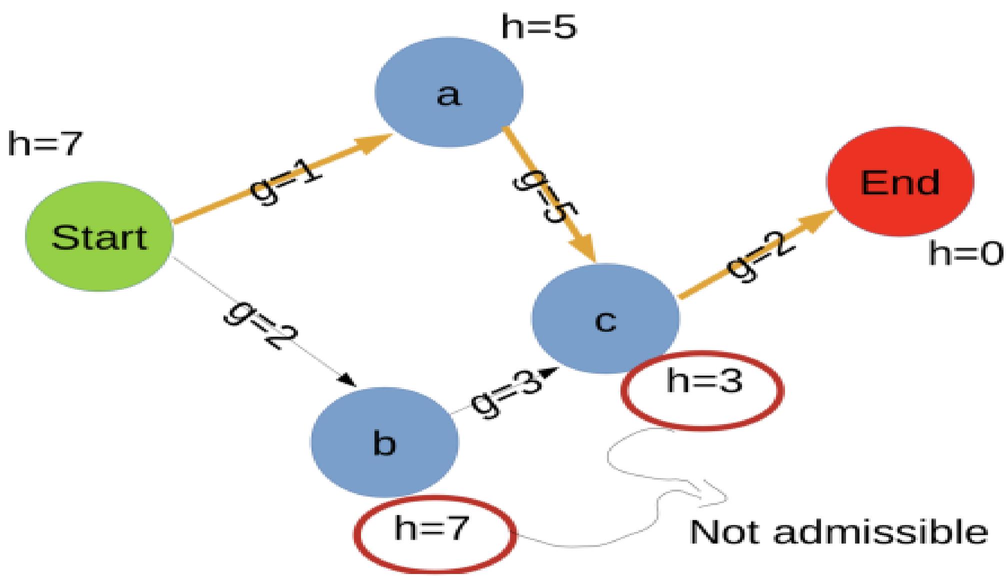
ب) جستجوی اول سطح ، اول عمق وجستجوی هزینه یکنواخت ، حالت های خاصی از جست و جوی اول بهترین هستند.

جستجوی هزینه یکنواخت با فرض f(n)=g(n) همان اول بهترین است و جستجوی اول سطح نیز اگر هزینه تمام مسیر ها یکسان باشد یعنی f(n)=depth(n)∝ $g(n)$ برابر جستجوی اول بهترین خواهد بود و همچنین جستجوی اول عمق هم با در نظر گرفتن f(n) = depth(n) همان اول بهترین خواهد بود.

ج) جست وجوی هزینه یکنواخت، حالت خاصی از جست و جوی *A است.

جستجوی اول بهترین همانند جستجوی هزینه یکنواخت است با این تفاوت که به جای تابع f از تابع g برای مرتب سازی صف اولویت استفاده می کند.

4) فضای حالتی را طراحی کنید که در آن *A با استفاده از Graph-Search جواب نیمه بهینه ای را با تابع قابل قبول ولی ناسازگار h(n) برگرداند.



مسیر بهینه: start->b->c->end=7

مسیر *A: start->a->c->end=8

5) در صفحه 115 دیدیم که آروین فاصله مستقیم، جستجوی اول بهترین حریصانه را در مسأله رفتن از Losi به Fagaras سرگردان می کند. درعین حال، آروین در مسأله برعکس رفتن از Fagaras به Losi عالی عمل می کند. آیا مسائلی وجود دارند که برای آنها این آروین در هر دو جهت گمراه کننده باشد؟

بله برای مثال در همین نقشه شهر رومانی اگر از Rimnicuvilcea به سمت هدف یعنی logoj برویم کوتاهترین مسیر باید از شهرهای Mehadia,Dobreta,Craiova عبور کند ولی اگر از روش اول بهترین حریصانه و تابع هیوریستیک استفاده کنیم در ابتدای مسیر اشتباه کرده واز شهر Rimnicuvilcea به Sibiu حرکت میکند و اگر بصورت برعکس حرکت کنیم از lugoj به سمت Mehadia میرود و دوباره

روش اول بهترین حریصانه آنرا به Lugoj بر می گرداند و باعث می شود در یک حلقه بی نهایت گیر بیفتیم.

6) تابع آروینی برای پازل 8تایی ابداع کنید که گاهی اوقات بیش از حد برآورد کند و نشان دهید چگونه در یک مسأله خاص می تواند به یک جواب نیمه بهینه منجر شود (درصورت تمایل می توانید از کامپیوتر استفاده کنید.) ثابت کنید که اگر h زیادی برآوردش هرگز بیش از c نشود، *A با استفاده از h جوابی را برمی گرداند که هزینه اش حداکثر به اندازه c بیشتر از هزینه جواب بهینه نیست.

تابع اکتشافی h=h1+h2 (h1 تعداد کاشی هایی که در جای خود قرار نگرفته اند وh2 مجموع فاصله کاشی ها از موقعیت هدفشان) در بعضی مواقع می تواند بیش از مقدار واقعی تخمین بزند.فرض کنید: $h(n) \leq h*(n)+c$ داده شده و همچنین می دانیم:k یک هدف نیمه بهینه با هزینه ای بیشتر از c باشد. اگر فرض شود: $g(k) > C*+c$ هرگره n که در مسیری به یک هدف بهینه وجود دارد در نظر بگیریم:

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

$\leq g(n)+h*(n)+c$

$\leq C* +c$

$\leq g(k)$

بنابراین k هرگز قبل از بسط هدف بهینه گسترش نخواهد یافت.

7)ثابت کنید یک آروینی سازگار، قابل قبول نیز هست. آروین قابل قبولی بسازید که سازگار نباشد.

یک تابع اکتشافی سازگار است:

$h(n) \leq h(n')+c(n,a,n')$

فرض کنید k گره از کوتاهترین مسیر به گره هدف که از گره n شروع شود اگر k=1 وn' گره هدف باشد آنگاه:

$h(n) \leq c(n,a,n')$

بعنوان استنتاج فرض کنید که n' روی کوتاهترین مسیر k مرحله ای از هدف قرار دارد و h(n') قابل قبول باشد آنگاه:

$h(n) \leq h(n')+c(n,a,n') \leq c(n,a,n')+h*(n') = h*(n)$

بنابراین تابع h(n) در k+1 مرحله ای هدف همچنان قابل قبول است.

8) مسأله فروشنده دوره گرد می تواند از طریق آروین درخت پوشای کمینه (MST) که برای برآورد هزینه تکمیل یک گشت استفاده می گردد حل شود، با این فرض که قسمتی از گشت از پیش ساخته شده باشد. هزینه MST یک مجموعه از شهرها، کوچکترین مجموع هزینه ارتباطات درختی است که تمامی شهرها را بهم متصل می کند.

الف) نشان دهید چگونه آروینی را می توان از نسخه تعدیل شده TSP بدست آورد.

مسئله TSP کوتاهترین مسیر بین شهرها که تشکیل یک حلقه بسته بدهند را حل میکند. طبق تعریف درخت پوشای مینیمال متوجه می شویم که همان نسخه تعدیل شده TSP است زیرا بدنبال گراف مینیمالی می گردد که حلقه بسته نداشته باشد ولی کاملا بهم متصل باشند.در نتیجه MST یک اکتشاف قابل قبول است.چون همواره مسیری کوتاهتر یا مساوی با یک حلقه بسته ارائه می دهد.

ب) نشان دهید که آروین MST برفاصله مستقیم برتری دارد.

در این مسئله باید از شهر انتهایی به شهر ابتدایی مجددا باز گردیم که معلوم است فاصله خط مستقیم این مقدار را کمتر از حد واقعی تخمین می زند اگر تعداد شهرها زیاد باشد و اگر تعداد شهر ها کم باشد نیز کارکرد کمتری نسبت به MST دارد. در MST همواره برای یک گره مقداری بالاتر ارائه می دهد(برای اکتشاف MST داریم برای وصل کردن گره هدف و گره جاری به دو روش می توانیم عمل کنیم که یکی از این روشها استفاده از همان فاصله خط مستقیم بین دو گره است یا روش دیگر اینکه توسط چند خط این کار را انجام دهیم که معمولا روش دوم صورت می گیرد (زیرا طبق قاعده مثلث این مقدار بیشتر از فاصله خط مستقیم خواهد بود.)

ج) مولد مسأله ای برای نمونه هایی از TSP بسازید که در آن، شهرها توسط نقاط تصادفی در مربع واحد بیان شوند.

راهی که پیشنهاد می شود برای پیاده سازی این است که هر گره دیده نشده را سعی کنیم به نزدیکترین همسایه اش متصل کنیم (طبق االگوریتم MST انجام می شود.)

د) برای ساخت MST الگوریتم کارآمدی در ادبیات بیابید و آن را همراه یک الگوریتم جستجوی قابل قبول برای حل نمونه های TSP مورد استفاده قرار دهید.

Cormen etal,1990, P50S الگوریتمی با پیچیدگی $O(E \log E)$ که E تعداد لبه های گراف است.

9)در صفحه 128 پازل 8تایی را به این شکل تعدیل کردیم که یک کاشی می تواند از مربع A به مربع B منتقل شود، اگر B خالی باشد، جواب دقیق این مسأله، **آروین گاشینگ(Gashing,1979)** رامعرفی می کند توضیح دهید چرا آروین گاشینگ حداقل به دقت h1 (کاشی هایی که در جای خود قرار نگرفته اند) می باشد و مواردی را نشان دهید که از هردوی h2, h1 (فاصله منهتن) دقیق تر است. آیا می توانید روش کارآیی برای محاسبه آروین گاشینگ پیشنهاد کنید؟

برای مسئله پازل 8تایی دقیقترین تابع اکتشاف همان h1است که تعداد کاشی هایی که در جای خود قرار نگرفته اند . ولی اکتشاف گاشینگ دقیق تر است زیرا در حالتی که این مسئله را ساده تر می کند یعنی انتقال در صورت خالی بودن کاشی B مقدار اکتشاف کمتر از مقدار اکتشاف h1 نیست و همواره قابل قبول می باشد.

همچنین اگر حالتی داشته باشیم تا در حالت هدف تا دو مربع همسایه را جلو و عقب کنیم حالتی خواهیم داشت که h1وh2 مقدار 2 را برمی گردانند ولی تابع اکتشاف گاشینگ مقدار3 را برمی گرداند

برای محاسبه اکتشاف گاشینگ این مراحل تکرار کنید تا به حالت هدف برسید.

فرض کنید برای رسیدن به هدف مربع B خالی باشد پس باید یکی از مربع هایی که در مکان اشتباه هستند را به خانه B منتقل کنیم.و اگر همه مربع ها سرجای درست باشند و مربع x درجای B قرار می گیرد.و این مراحل تکرار کنید تا مسئله حل شود خواهید دید این روش راه حل بهینه را پیدا می کند.

10) پیش از این، دو آروین ساده برای پازل 8 تایی بیان کردیم : فاصله منهتن وکاشیهایی که درجای خود قرار ندارند آروین های متعددی بیان کرده اند که این مسأله را بهبود بخشیده اند.(برای مثال به (1992) Hansson et al.(1992) ,Mostow and Prieditis(1989) ,Nilson(1971)) مراجعه کنید.

ادعاهای مذکور را با پیاده سازی این آروینها ومقایسه کارایی الگوریتمهای حاصل بررسی کنید.

از نظر زمان اجرا و تعداد گره تولیدی باهم مقایسه می شوند که برای مشخص شدن زمان اجرا بجای پازل 8 از پازل 15 یا پازل 24 تایی استفاده کنید.

- Hansson et al. (1992) یک الگوریتم جستجوی A* با تابع هزینهای ترکیبی از فاصله منهتن و کاشیهای نامناسب ارائه دادند. آنها نشان دادند که الگوریتم آنها به طور متوسط ۵۵/۱۹ گام برای حل پازل ۸ تایی نیاز دارد.
- Mostow and Prieditis (1989) چندین الگوریتم جستجوی A* با توابع هزینهای مختلف مثل فاصله منهتن، کاشیهای نامناسب و تعداد حرکات قابل انجام مقایسه کردند. آنها نشان دادند که الگوریتم با تابع هزینه فاصله منهتن به طور متوسط ۶/۱۹ گام برای حل پازل ۸ تایی نیاز دارد.
- Nilson (1971) یک الگوریتم جستجوی A* با تابع هزینه فقط کاشیهای نامناسب ارائه داد
- Nilson (1971) نشان داد که الگوریتم با تابع هزینه فقط کاشیهای نامناسب به طور متوسط ۴/۲۳ گام برای حل پازل ۸ تایی نیاز دارد.

بنابراین، بر اساس این نتایج، میتوان گفت که مقاله Nilson (1971) زمان اجرای بیشتری دارد و مقاله Hansson et al. (1992) زمان اجرای کمتری دارد. البته این مقایسه فقط بر اساس تعداد گامهای لازم

برای حل پازل است و ممکن است عوامل دیگری مثل پیچیدگی زمانی و حافظهای الگوریتمها نیز مهم باشند.

بهترین الگوریتم بستگی به هدف و شرایط شما دارد. اگر میخواهید کمترین تعداد گام را برای حل پازل پیدا کنید، الگوریتم Hansson et al. (1992) به نظر میرسد که کارایی بالاتری دارد. اما اگر میخواهید ساده ترین و راحتترین الگوریتم را پیاده سازی کنید، الگوریتم Nilson (1971) شاید گزینه مناسبتری باشد. همچنین باید در نظر داشته باشید که این الگوریتمها ممکن است برای پازلهای با ابعاد و تعداد مختلف کارایی متفاوتی داشته باشند و بنابراین نمیتوان به طور قطع گفت که کدام یک بهترین است.

11) نام الگوریتمی را که در هریک از موارد زیر بدست می آید، مشخص کنید.

الف) جستجوی پرتو محلی با k=1

جستجوی پرتو محلی یک الگوریتم فراابتکاری است که گراف را با استفاده از راسهایی که در یک مجموعه خاص احتمال وجود بیشتری دارند، میپیماید. در این الگوریتم، فقط k راس با بهترین مقدار تابع هزینه در هر مرحله نگهداری میشوند و بقیه حذف میشوند. جستجوی پرتو محلی با k=1: این الگوریتم همان جستجوی کاوشی (hill climbing) همان جستجوی تپه نوردی است.

که فقط یک راس را در هر مرحله نگهداری میکند و به سمت راسهای همسایه با مقدار بهتر حرکت میکند.

ب) جستجوی پرتو محلی با یک "حالت شروع " و بدون محدودیت روی تعداد حالت نگهداری شده.

• جستجوی پرتو محلی با یک "حالت شروع" و بدون محدودیت روی تعداد حالت نگهداری شده: این الگوریتم همان جستجوی اول سطح (breadth-first search) است که از یک راس شروع میکند و همهٔ راسهای همسایه را در هر مرحله نگهداری میکند.

پ) simulated annealing با T=0 درتمام موارد (وحذف تست خاتمه)

• simulated annealing یک الگوریتم فراابتکاری است که الهام گرفته از فرآیند خنکسازی فلزات است. در این الگوریتم، چنانچه حالت جانشین بهبود داده باشد، حالت جانشین قطعاً قبول میشود؛ ولي چنانچه حالت جانشین بهبود نیافته باشد، آن را با احتمال $e^{(-\Delta E/T)}$ قبول میکنیم که ΔE تغییر هزینه و T دمای فعلي است.

simulated annealing • با T=0 در تمام موارد (و حذف تست خاتمه): این الگوریتم همان جستجوی کاوشی (hill climbing) است که فقط حالات جانشین با مقدار بهتر را قبول میکند و هرگز به حالات بدتر نمیرود.

ت) simulated annealing با ∞ =T در تمام موارد.

• simulated annealing با $T=\infty$ در تمام موارد: این الگوریتم همان جستجوی تصادفی (random search) است که هر حالت جانشین را با احتمال یکسان قبول میکند و بدون هدف و جهت خاصی حرکت میکند.

ث) الگوریتم ژنتیک با جمعیتی به اندازه ی N=1

• الگوریتم ژنتیک یک الگوریتم فراابتکاری است که الهام گرفته از فرآیند تکامل زیستی است. در این الگوریتم، یک جمعیت از حالتها (ژنوتیپها) تولید میشود و با استفاده از عملگرهای تولید مثل، جهش و انتخاب، بهبود یافته و به سوی حالت بهینه (فنوتیپ) حرکت میکنند.

الگوریتم ژنتیک با جمعیتی به اندازه ی N=1: این الگوریتم همان جستجوی تصادفی (random search) است که فقط یک حالت را در هر مرحله تولید میکند و بدون استفاده از عملگرهای تولید مثل، جهش و انتخاب، آن را ارزیابی میکند.

12) گاهی اوقات توابع ارزیاب خوبی برای یک مسأله وجود ندارد ولی روش مقایسه خوبی موجود است: یعنی روشی برای تعیین اینکه یک گره درمقایسه با گره دیگر چقدر بهتر است، بدون آنکه به گره ها مقادیر عددی انتساب دهد. نشان دهید که همین مطلب برای انجام جستجوی اول بهترین کفایت می کند . آیا نسخه مشابهی برای *A وجود دارد؟

• جستجوی اول بهترین یک الگوریتم جستجو است که یک گراف را با بسط دادن محتملترین راس، که بنابر قوانین خاص انتخاب میشود، پیمایش میکند. در این الگوریتم، فقط گرهای را بسط میدهد که بهترین مقدار تابع ارزیاب را دارد

• *A یک الگوریتم جستجو است که با استفاده از یک تابع هزینه کل، هزینه رسیدن به هدف را حداقل میکند. تابع هزینه کل شامل دو قسمت است: هزینه واقعی رسیدن از راس شروع تا راس فعلی (g(n)) و هزینه تخمین زده شده رسیدن از راس فعلی تا راس هدف (h(n)). فرمول تابع هزینه کل به صورت زیر است: $f(n) = g(n) + h(n)$

• میتوان نشان داد که برای انجام جستجوی اول بهترین، کافی است که یک روش مقایسه خوب برای گرهها داشته باشیم، بدون آنکه به آنها مقادیر عددی انتساب دهیم. برای این کار، میتوان از یک رابطه ترتیبی (order relation) استفاده کرد که نشان میدهد که گره A به گره B ترجیح داده میشود یا خیر. این رابطه ترتیبی باید خاصیتهای زیر را داشته باشد:

• قطعیت (reflexivity): هر گره به خودش ترجیح داده میشود. $A \le A$

• ضعف تعدی (weak transitivity): اگر گره A به گره B و گره B به گره C ترجیح داده شود، آنگاه گره A به گره C هم ترجیح داده میشود. $A \le B$ و $B \le C \Rightarrow A \le C$

• ضعف تقارن (weak symmetry): اگر گره A و گره B به هم ترجیح داده شوند، آنگاه گره B و گره A هم به هم ترجیح داده میشوند. $B \leq A$ و $A \leq B => A = B$

با استفاده از این رابطه ترتیبی، میتوان در هر مرحله از جستجو، گرهای را بسط داد که به همهٔ سایر گرهها ترجیح داده شود. این روش باعث میشود که جستجو به سمت هدف حرکت کند و فضای حالت را کاهش دهد.

• برای الگوریتم A*، نمیتوان نسخه مشابهی با روش مقایسه خوب داشت، زیرا این الگوریتم نیاز دارد که تابع هزینه کل را برای هر گره محاسبه کند. این تابع هزینه کل شامل دو قسمت است: هزینه واقعی رسیدن از راس شروع تا راس فعلی (g(n)) و هزینه تخمین زده شده رسیدن از راس فعلی تا راس هدف (h(n)). بنابراین، برای این الگوریتم، باید به هر گره یک مقدار عددی انتساب داد که نشان دهندهٔ هزینه کل باشد. اگر فقط از یک روش مقایسه خوب استفاده کنیم، نمیتوانیم تفاوت بین هزینه واقعی و تخمین زده شده را در نظر بگیریم و ممکن است به گرهای برویم که هزینه واقعی آن زیاد باشد ولی هزینه تخمین زده شده آن کم باشد.

13) پیچیدگی زمانی *LRTA را به پیچیدگی فضایی اش ارتباط دهید.

LRTA* یک الگوریتم جستجوی هیوریستیک در زمان واقعی است که با استفاده از یک تابع هزینه کل، هزینه رسیدن به هدف را حداقل میکند. تابع هزینه کل شامل دو قسمت است: هزینه واقعی رسیدن از راس شروع تا راس فعلی (g(n)) و هزینه تخمین زده شده رسیدن از راس فعلی تا راس هدف (h(n)). فرمول تابع هزینه کل به صورت زیر است: $f(n) = g(n) + h(n)$. در این الگوریتم، عامل در هر مرحله، گرهای را برای حرکت انتخاب میکند که دارای کمترین مقدار تابع هزینه کل باشد. عامل همچنین مقادیر تابع هزینه کل را به صورت چندگانه در حافظه نگهداری میکند و با تجربه، آنها را بهبود میبخشد.

پیچیدگی زمانی LRTA*، برابر است با تعداد گرههای بازدید شده توسط عامل تا رسیدن به هدف. این تعداد به اندازهٔ مسافت بین راس شروع و راس هدف، تابع هزینه واقعی و تخمین زده شده، و خصوصیات گراف وابسته است. در بدترین حالت، پیچیدگی زمانی LRTA*، برابر است با O(b^d) که b عامل فرزندان و d عمق درخت جستجو است.

• پیچیدگی فضایی LRTA*، برابر است با حجم حافظهای که عامل برای نگهداری مقادیر تابع هزینه کل نیاز دارد. این حجم به اندازهٔ تعداد گرههای بازدید شده توسط عامل، وابسته است. در بدترین حالت، پیچیدگی فضایی LRTA*، برابر است با O(b^d) که b عامل فرزندان و d عمق درخت جستجو است.

• با توجه به این رابطه ها، می توان نتیجه گرفت که پیچیدگی زمانی و فضایی LRTA*، به صورت خطی به هم وابسته هستند. یعنی هرچه تعداد گرههای بازدید شده توسط عامل بیشتر شود، هم پیچیدگی زمانی و هم پیچیدگی فضایی بیشتر میشود. بنابراین، برای کاهش پیچیدگی زمانی و فضایی LRTA*، باید سعی کرد که تعداد گرههای بازدید شده را کمینه کنیم. این کار میتواند با انتخاب یک تابع هزینه تخمین زده شده مناسب و یا استفاده از روشهای بهبود یافتهٔ LRTA* انجام شود

14) فرض کنید که کارگزاری در محیط مارپیچ 3*3 مانند آنچه در شکل 18-4 نشان داده شد قراردارد.

کارگزار می داند که محل اولیه اش (1و1) است وهدف در (3و3) قرار دارد و اقدامهای UP,DOWN,RIGHT,LEFT تاثیر همیشگی شان را دارند. به جز مواقعی که دیواری سر راه باشد، کارگزار نمی داند که دیوار های داخلی کجا قرار گرفته اند.در نظر عامل $4096 = 2^{12}$ حالت مختلف برای پیکربندی وجود دارد. درهر حالت مفروض، کارگزار مجموعه اقدامات مجاز را درک می کند. همچنین می تواند بفهمد که این حالت را قبلا ملاقات کرده یا حالت جدیدی است.

الف) توضیح دهید چگونه می توان در فضای حالت باور، این مسأله جستجوی برخط را به صورت یک جستجوی برون خط دید که حالت باور اولیه اش شامل تمامی پیکربندیهای ممکن محیط باشد. بزرگی حالت باور اولیه چه اندازه است؟

فضای حالت باور اولیه شامل مجموعه ای ازتمام 4096 پیکربندی است.کل فضای حالت به تعداد زیر مجموعه های این مجموعه یعنی 2^{4096} حالت باور می شود.

هنگامی که عامل حرکت میکند و خانه های جدیدی را مشاهده میکند، فضای حالت باور عامل کاهش مییابد. چون عامل میتواند بفهمد که آیا خانهای را قبلا دیده است یا نه، پس فضای حالت باور عامل شامل تمام پیکربندیهای ممکن است که با تاریخچه حرکات عامل سازگار باشند. برای مثال، اگر عامل از خانه (۱،۱) به خانه (۱،۲) حرکت کند و ببیند که خانه (۱،۲) دارای دیوار در سمت راست است، پس فضای حالت باور عامل شامل تمام پیکربندیهای ممکن است که دیوار را در خانه (۱،۲) در نظر بگیرند.

بنابراین فضای حالت کاملا مشخص شده و تعداد3^{12}حالت باور قابل دسترس وجود خواهد داشت.عامل در هرحالت می تواند چهارسوی خود را مشاهده کند و وجود یا عدم وجود دیوارها در آن خانه را بررسی کند.و باتوجه به مشاهدات خود وضعیت برایش مشخص می شود و دیگر نیاز نیست تا باورهای زیادی را برای هرخانه حدس بزندوپس تمام زیر مجموعه های ممکن برای هرخانه $16=2^4$ حالت ممکن وجود دارد. به همین ترتیب، هر بار که عامل حرکت میکند و خانهای را مشاهده میکند، فضای حالت باور عامل کاهش مییابد. اگر عامل بتواند تمام خانهها را ببیند و تمام دیوارها را شناسایی کند، فضای حالت باور عامل به یک حالت منحصر به فرد کاهش مییابد که با حالت واقعی سیستم برابر است

ب) چند ادراک مختلف در حالت اولیه ممکن است؟

$4=2^2$ حالت مختلف با فرض دانستن دیوار خارجی در لحظه شروع دو دیوار داخلی مشاهده شده است

ج) چند شاخه اول یک طرح اقتضایی را برای این مسأله بیان کنید. بزرگی طرح کامل (حدوداً) چه اندازه است؟

در هر حالت باور عامل یک واکنش را انتخاب می کند تا به 8 حالت باور برسد.(در زمان ورود به مربع وسط)با تکرار گامهای عامل در رسیدن به پایان می بینیم که عامل کل محیط پرپیچ و خم را در حداکثر 18 مرحله سپری می کند.بنابراین نقشه کامل بیشتر از 8 گره نخواهد داشت. بعبارت دیگر 3^{12} گره وجود دارد.

توجه داشته باشید که این طرح اقتضایی، راه حلی برای تمامی محیطهای ممکنی است که در تعریف داده شده می گنجد. بنابراین یکی درمیان بودن جستجو و اجرا ، حتی در محیطهای ناشناخته هم ضروری نمی باشد.

15) در این تمرین کاربرد روشهای جستجوی محلی در حل مسائل TSP از نوعی که در تمرین 8 بیان گردید را بررسی می کنیم.

الف) یک رویکرد تپه نوردی برای حل TSP ها ابداع نمایید. نتایج این رویکرد را با جوابهای بهینه ای که از طریق الگوریتم A* همراه آروین MST (تمرین 8) بدست می آید مقایسه کنید.

رویکرد تپه نوردی یک روش حریصانه است که با انتخاب بهترین حالت محلی، سعی می کند به جواب بهینه برسد. اما این روش ممکن است در نقطه ای که بهبود نمی تواند انجام شود، گیر کند و جواب بهینه را پیدا نکند. برای حل TSP با روش تپه نوردی، می توان از الگوریتم زیر استفاده کرد:

۱. شروع کنید با یک جوله تصادفی از شهر ها.

۲. در هر مرحله، دو شهر را با هم عوض کنید و هزینه جوله جدید را محاسبه کنید.

۳. اگر جوله جدید بهتر از جوله فعلی بود، آن را به عنوان جوله فعلی قبول کنید و به مرحله بعد بروید.

۴. اگر جوله جدید بدتر یا مساوی جوله فعلی بود، آن را رد کنید و به مرحله بعد بروید.

۵. تکرار کنید تا زمانی که هیچ بهبود دیگری در جوله فعلی امکان پذیر نباشد.

این الگوریتم ساده و سریع است، اما نمی تواند تضمین کند که جواب بهینه را پیدا کند. بستگی به جوله اولیه دارد که ممکن است خوب یا بد باشد. برای مقایسه نتایج این روش با الگوریتم A* همراه آروین MST، می توان از شبکه های مختلف با تعداد شهر های مختلف استفاده کرد و زمان و هزینه حل هر دو الگوریتم را اندازه گیری کرد.

ب) یک رویکرد الگوریتم ژنتیک برای حل مسأله فروشنده دوره گرد ابداع نمایید. نتایج آن را با رویکردهای دیگر مقایسه کنید. می توانید برای بعضی پیشنهادات در زمینه بازنمایی به Larranaga et al.(1999) مراجعه کنید.

برای حل TSP با الگوریتم ژنتیک، می توان از الگوریتم زیر استفاده کرد:

۱. جمعیت اولیه را به صورت تصادفی تولید کنید. هر فرد یک جوله از شهر ها را نشان می دهد.

۲. سازگاری هر فرد را محاسبه کنید. سازگاری برابر با معکوس طول مسیر است. هر چه طول مسیر کوتاه تر باشد، سازگاری بالاتر است.

۳. تا زمان اتمام شرط توقف، این مراحل را تکرار کنید:

• والدین را با استفاده از روش چرخش گالوپ (Roulette Wheel) انتخاب کنید. در این روش، احتمال انتخاب هر فرد مستقیما به سازگاری آن بستگی دارد.

• فرزندان را با استفاده از عملگر تلاقی تولید کنید. عملگر تلاقی دو والد را با هم ترکیب می کند و ژن های جدید را به ارث می برد. مثلا، می توان از عملگر تلاقی دور (Cycle Crossover) استفاده کرد

• فرزندان را با استفاده از عملگر جهش تغییر دهید. عملگر جهش با احتمال کم، ژن های فرزند را تغییر می دهد و تنوع را در جمعیت حفظ می کند. مثلا، می توان از عملگر جابجایی ((Swap Mutation) استفاده کرد.

• سازگاری فرزندان را محاسبه کنید.

• فرزندان را به جمعیت فعلی اضافه کنید.

۴. بالاترین سازگاری را در بین جمعیت پیدا کنید و جوله متناظر آن را به عنوان جواب نهایی بدهید

روش های دیگر مقایسه کنید، می توان از شبکه های مختلف با تعداد شهر های مختلف استفاده کرد و زمان و هزینه حل هر الگوریتم را اندازه گیری کرد. برای بعضی پیشنهادات در زمینه بازنمایی، می توان به مقاله Larranaga et al. (1999) مراجعه کرد.

16) تعداد زیادی نمونه برای پازل 8تایی و 8 وزیر تولید کنید و با استفاده از تپه نوردی(گونه های تیزترین صعود و اولین گزینه) تپه نوردی با شروع مجدد تصادفی وشبه تاب کاری آنها (تاحدامکان) را حل کنید. هزینه و درصد مسائل حل شده را اندازه گیری کنید. و آنهارا همراه با هزینه جواب بهینه به صورت نمودار درآورید. درباره نتایج خود اظهار نظر نمایید.

پازل 8تایی یک مسئله از نوع پازل لغزان است که در آن هدف جابجایی تکه های شماره دار در یک صفحه 3×3 است تا به حالت نهایی برسند مسئله 8 وزیر یک مسئله از نوع جایگشتی است که در آن هدف قرار دادن 8 وزیر روی صفحه شطرنج است به طوری که هیچ دو وزیری یکدیگر را تهدید نکنند.

تپه نوردی یک الگوریتم جستجوی محلی است که با انتخاب بهترین حالت مجاور، سعی می کند به جواب بهینه برسد. اما این الگوریتم ممکن است در نقطه ای که بهبود نمی تواند انجام شود، گیر کند و جواب بهینه را پیدا نکند. برای رفع این مشکل، می توان از دو روش زیر استفاده کرد:

• تپه نوردی با شروع مجدد تصادفی: در این روش، هر بار که الگوریتم در یک نقطه بن بست گیر کند، یک حالت جدید تصادفی را به عنوان حالت فعلی قبول می کند و دوباره جستجو را از سر می گیرد. این روش با افزایش تعداد شروع های مجدد، احتمال پیدا کردن جواب بهینه را بالا می برد.

• شبه تاب کاری: در این روش، الگوریتم با استفاده از یک پارامتر دما، اجازه می دهد که گاهی اوقات حالات بدتر را هم قبول کند. با این کار، الگوریتم ممکن است از نقطه بن بست خارج شود و به منطقه بهتری برود. دما در طول جستجو کاهش می یابد و الگوریتم سخت گیرانه تر می شود.

برای حل پازل 8تایی با تپه نوردی می توان از الگوریتم زیر استفاده کرد:

۱. شروع کنید با یک حالت تصادفی از پازل.

۲. در هر مرحله، یک تکه را با خانه خالی عوض کنید و هزینه حالت جدید را محاسبه کنید. هزینه برابر با تعداد تکه هایی است که در جای اشتباه قرار دارند.

۳. اگر حالت جدید بهتر از حالت فعلی بود، آن را به عنوان حالت فعلی قبول کنید و به مرحله بعد بروید.

۴. اگر حالت جدید بدتر یا مساوی حالت فعلی بود، آن را رد کنید و به مرحله بعد بروید.

۵. تکرار کنید تا زمانی که هیچ بهبود دیگری در حالت فعلی امکان پذیر نباشد.

برای حل مسئله 8 وزیر با شبه تاب کاری، می توان از الگوریتم زیر استفاده کرد:

۱. شروع کنید با یک حالت تصادفی از صفحه شطرنج. هر سطر یک وزیر دارد.

۲. در هر مرحله، یک وزیر را در سطر خود به سمت راست یا چپ جابجا کنید و هزینه حالت جدید را محاسبه کنید. هزینه برابر با تعداد جفت های وزیر هایی است که یکدیگر را تهدید می کنند.

۳. اگر حالت جدید بهتر از حالت فعلی بود، آن را به عنوان حالت فعلی قبول کنید و به مرحله بعد بروید.

۴. اگر حالت جدید بدتر از حالت فعلی بود، با یک احتمال وابسته به دما، آن را قبول یا رد کنید. دما در طول جستجو کاهش می یابد و الگوریتم سخت گیرانه تر می شود.

۵. تکرار کنید تا زمانی که هزینه صفر شود یا دما به حد آستانه برسد.

برای اندازه گیری هزینه و درصد مسائل حل شده، می توان از معیارهای زیر استفاده کرد:

• هزینه: برابر با تعداد گام هایی است که الگوریتم برای رسیدن به جواب بهینه طی می کند.

• درصد مسائل حل شده: برابر با نسبت تعداد مسائلی که الگوریتم جواب بهینه آنها را پیدا کرده است به تعداد کل مسائل.

برای رسم نمودار، می توان از ابزارهای گرافیکی مانند Excel یا Matlab استفاده کرد. برای هر الگوریتم، یک نمودار خطی رسم می کنیم که در آن محور افقی تعداد نمونه ها و محور عمودی هزینه یا درصد مسائل حل شده را نشان می دهد. سپس، نمودارهای مختلف را با هم مقایسه می کنیم.

برای اظهار نظر درباره نتایج، می توان از معیارهای زیر استفاده کرد:

• سرعت: برابر با زمان لازم برای حل یک مسئله. هر چه زمان کمتر باشد، سرعت بالاتر است.

• دقت: برابر با اختلاف هزینه جواب بهینه با هزینه جواب الگوریتم. هر چه اختلاف کمتر باشد، دقت بالاتر است.

• قابلیت اطمینان: برابر با درصد مسائل حل شده. هر چه درصد بالاتر باشد، قابلیت اطمینان بالاتر است.

با توجه به این معیارها، می توان گفت که الگوریتم شبه تاب کاری معمولا سریع تر، دقیق تر و قابل اطمینان تر از الگوریتم تپه نوردی و تپه نوردی با شروع مجدد تصادفی است. زیرا این الگوریتم می تواند از نقاط بن بست خارج شود و به مناطق بهتری برود. البته، این مقایسه بستگی به پارامترهای مختلف مانند تعداد نمونه ها، شرط توقف، دمای اولیه و نرخ خنک شدن دارد.

17) در این تمرین تپه نوردی را در زمینه هدایت روبات، با استفاده از محیط شکل 3-22 بعنوان یک مثال بررسی می کنیم:

الف) تمرین 16-3 با استفاده از تپه نوردی مجددا حل کنید. آیا هیچ گاه کارگزارتان در یک کمینه محلی گیر می افتد؟ آیا ممکن است توسط موانع محدب گیر بیفتد؟

تپه نوردی روشی است که در یافتن مسیر قابل قبول در زمانیکه مسیربهینه وجود ندارد و هزینه محاسبه کم است، کارا است ولی در محیط های دوبعدی شکست میخورد. اگر از تپه نوردی مجددا حل کنیم، ممکن است کارگزار ما در یک کمینه محلی گیر بیفتد. زیرا این الگوریتم فقط به حالات مجاور نگاه می کند و اگر هیچ یک از آنها بهبود نداشته باشند، جستجو را متوقف می کند. برای رفع این مشکل، می توان از تپه نوردی با شروع مجدد تصادفی یا LRTA* استفاده کرد.

همچنین، ممکن است کارگزار ما توسط موانع محدب گیر بیفتد. زیرا این الگوریتم فقط به هزینه های محلی توجه می کند و از توپولوژی کلی گراف آگاه نیست. برای رفع این مشکل، می توان از الگوریتم های دیگری مانند A* یا RBFS استفاده کرد که با استفاده از یک تابع هیوریستیک، بهترین گره را برای گسترش انتخاب می کنند.

ب) محیط چند ضلعی غیرمحدبی بسازید که در آن کارگزار گیر می افتد؟

در محیطی باموانع گوشه داراحتمال گیرکردن زیاد است.

ج) الگوریتم تپه نوردی را به نحوی تغییر دهید که برای تعیین این که در گام بعدی کجا برود به جای انجام جستجویی به عمق 1، یک جستجو به عمق k انجام دهد. باید بهترین مسیر با k گام را پیدا کند و یک گام در طول آن مسیر بردارد. سپس این فرایند را مجددا تکرار کند.

دقت کنید این روش همان جستجوی عمق محدود است که در آن یک گام در مسیر بهینه انتخاب می کند حتی اگر یک راه حل نباشد.

برای تغییر الگوریتم تپه نوردی به گونه ای که برای تعیین این که در گام بعدی کجا برود به جای انجام جستجویی به عمق 1، یک جستجو به عمق k انجام دهد، می توان از الگوریتم زیر استفاده کرد:

// تپه نوردی با عمق k

۱. شروع کنید با یک حالت اولیه x

۲. در هر مرحله، یک جستجوی محدود به عمق k را از x شروع کنید و بهترین مسیر را پیدا کنید. اگر مسیر خالی بود یا به گره هدف رسید، جستجو را متوقف کنید.

۳. اگر مسیر خالی بود، بگویید که جواب وجود ندارد و خاتمه دهید.

۴. اگر مسیر به گره هدف رسید، بگویید که جواب پیدا شده است و خاتمه دهید.

۵. در غیر این صورت، یک گام در طول مسیر بردارید و x را به گره جدید تغییر دهید.

۶. به مرحله ۲ بروید.

د) آیا هیچ kای وجود دارد که برای این الگوریتم جدید فرار از کمینه های محلی را تضمین کند؟

اگر k را برابر حداکثر اضلاع چندضلعی درنظر بگیریم آنگاه همواره فرار صورت می گیرد.

ه) توضیح دهید چگونه LRTA* این کارگزار را قادر می سازد از کمینه های محلی در این حالت فرار کند؟

در این روش، الگوریتم با استفاده از یک تابع هیوریستیک، هزینه تخمینی حالات را به صورت آنلاین و بازگشتی به روز می کند. با این کار، الگوریتم ممکن است از نقطه بن بست خارج شود و به منطقه بهتری برود. LRTA* یک الگوریتم یادگیری در زمان واقعی است که با تغییر دادن تابع هیوریستیک، سعی می کند به جواب بهینه نزدیک شود

18) کارایی A* وRBFS را برروی مجموعه ای از مسائلی که بطور تصادفی در قلمرو پازل 8 (با فاصله منهتن) وtsp (با MST به تمرین 8 مراجعه کنید) تولید شده اند. مقایسه کنید. درباره نتایج خود بحث کنید. هنگامی که عدد تصادفی کوچکی به مقادیر آروین در قلمرو پازل 8 اضافه می شود. چه اتفاقی برای کارایی RBFS می افتد؟

برای مقایسه کارایی این دو الگوریتم، می توان از معیارهای زیر استفاده کرد:

• زمان: برابر با زمان لازم برای حل یک مسئله. هر چه زمان کمتر باشد، کارایی بالاتر است.

• حافظه: برابر با حافظه لازم برای نگهداری گره های تولید شده. هر چه حافظه کمتر باشد، کارایی بالاتر است.

• دقت: برابر با اختلاف هزینه جواب بهینه با هزینه جواب الگوریتم. هر چه اختلاف کمتر باشد، دقت بالاتر است.

برای رسم نمودار، می توان از ابزارهای گرافیکی مانند Excel یا Matlab استفاده کرد. برای هر الگوریتم، یک نمودار خطی رسم می کنیم که در آن محور افقی تعداد نمونه ها و محور عمودی زمان، حافظه یا دقت را نشان می دهد. سپس، نمودارهای مختلف را با هم مقایسه می کنیم.

برای اظهار نظر درباره نتایج می توان از معیارهای زیر استفاده کرد:

• سرعت: برابر با نسبت زمان حل مسئله به تعداد نمونه ها. هر چه سرعت بالاتر باشد، الگوریتم کارآمدتر است.

• حافظه: برابر با نسبت حافظه مصرفی به تعداد نمونه ها. هر چه حافظه کمتر باشد، الگوریتم بهینه تر است.

• دقت: برابر با نسبت هزینه جواب الگوریتم به هزینه جواب بهینه. هر چه دقت نزدیک به یک باشد، الگوریتم صحیح تر است.

با توجه به این معیارها، می توان گفت که الگوریتم A* معمولا سریع تر، دقیق تر و قابل اطمینان تر از الگوریتم RBFS است. زیرا این الگوریتم با استفاده از یک تابع ارزیابی، بهترین گره را برای گسترش انتخاب می کند و از پیمایش بی ربط جلوگیری می کند. البته، این مقایسه بستگی به پارامترهای مختلف مانند تابع هزینه، تابع هوریستیک، شکل گراف و شروط محدود کننده دارد.