Subject:
Year Month Date

زهرا توکلی 401200 63 «به نام تعالی» تکلیف سوم هوش مصنوعی

سوال ۱:

مقایسه الگوریتم ها:

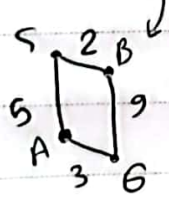
الف) الگوریتم Uniform cost search از الگوریتم های جست‌وجوی ناآگاهانه است که frontier آن صف اولویتی است که عضوی از آن اول برداشته می‌شود که فاصله‌ی آن تا نقطه‌ی شروع (s) کمتر باشد ولی A* یک الگوریتم جست‌وجوی آگاهانه است که مانند uniform cost search، frontier آن صف اولویت است ولی عضوی ابتدا از آن حذف می‌شود که مقدار تابع f(n) آن کمتر باشد. تابع f(n) برای هر نود گراف حاصل جمع دو جزء است جزء اول فاصله‌ی نود تا نود شروع (s) است و جزء دوم یک تابع هیوریستیک است که فاصله آن نود تا نود مقصد (G) را برآورد می‌کند.

ب) الگوریتم uniform cost search همیشه جواب بهینه را می‌یابد ولی A* در صورتی جواب بهینه را پیدا می‌کند که تابع هیوریستیکی که فاصله‌ی نود تا نود goal را برآورد می‌کند consistent و یا admissible باشد پس A* الزاماً جواب بهینه را نمی‌یابد. شاید A* از نظر سرعت هم الزاماً از uniform cost search بهتر نباشد چون باید در هر نود مقدار تابع هیوریستیک در آن نود هم حساب کند که اگر محاسبه‌ی آن زمان زیادی نیاز داشته باشد و یا تابع هیوریستیکی داشته باشیم که فاصله تا نود مقصد را به درستی و نزدیک به مقدار واقعی آن برآورد نکند، هزینه‌ای که برای محاسبه‌ی تابع هیوریستیک در نود های دهیم ممکن است با کاهش فضای جست‌وجو که تابع هیوریستیک به ارمغان می‌آورد جبران نشود.

Heuristic Admissibility:

الف) در نسخه های tree search الگوریتم A* برای یافتن جواب بهینه h(n) باید admissible باشد یعنی فاصله نود تا نود goal را overstimate نکند. در نسخه graph search الگوریتم h(n) باید consistent باشد یعنی به ازای هر همسایه از نود n مانند n' باید $h(n) \leq w_{nn'} + h(n')$ باشد که $w_{nn'}$ وزن یال بین دو نود n و n' است.

ب) مثلاً گراف زیر را در نظر بگیرید. فاصله‌ی واقعی نود ها تا یکدیگر روی گراف مشخص شده اند. فرض کنیم $h(n)$ ، admissible ست overstimate می‌کند. $h(A)=8$ و $h(B)=9$. در نتیجه $f(A)=5+8=13$ و $f(B)=2+9=11$ و الگوریتم نود B که f کمتری را دارد انتخاب می‌کند در حالی که مسیر بهینه از نود A می‌گذرد.



Subject
Year Month Date

الگوریتم های حریصانه و A^*:

الف) الگوریتم greedy best-first search چون فاصله هر نود تا نود start را در نظر می گیرد و صرفا مقدار تابع هیوریستکی که فاصله تا نود goal را برآورد می کند در نظر می گیرد ممکن است بهینه رفتار نکند و ما نودی داشته باشیم که مقدار $h(n)$ آن از نود دیگری کمتر باشد و الگوریتم حریصانه آن را انتخاب کند ولی فاصله این نود تا نود start از دیگری بیش تر باشد و کل طول مسیر بیش تر شود مانند گراف زیر:

خواهیم واقعی روی گراف مشخص شده است. فرض کنیم $h(A)=3$ و $h(B)=4$ پس greedy-bestfirst search نود A که $h(n)$ آن کمتر است را انتخاب می کند ولی مسیر بهینه از نود B می گذرد.

ب) A^* هم فاصله نود با مبدأ را در نظر می گیرد و هم از تابع هیوریستیک استفاده می کند. بنابراین اگر نودی مقدار تابع $h(n)$ کمتری داشته باشد ولی فاصله آن تا مبدأ خیلی زیاد باشد انتخاب نمی شود و مشکل بالا برای الگوریتم greedy best-first را نخواهد داشت.

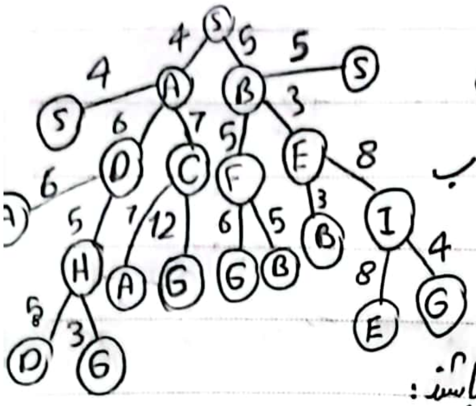
تحلیل پیچیدگی: الگوریتم A^* در فضاهای حالات ناشناخته با تعدادی زیادی goalstate زیر بهینه که مقدار $f(n)$ آنها به $f(n)$ نود goal بهینه نزدیک باشد بدترین پیچیدگی زمانی و حافظه ای را خواهد داشت که می توان آن را به صورت $O(b^d)$ در نظر گرفت.

با استفاده از نسخه های دیگر A^* مثل IDA*، MA* و SMA* می توان پیچیدگی حافظه را کاهش داد مثلاً:

IDA*(iterative deepening A*): الگوریتم IDS و A^* را ترکیب کرده است. یک threshold برای حداکثر میزان مقدار تابع $f(n)$ در هر مرحله تعیین کرده و نودهایی که $f(n)$ آنها از آن threshold بیش تر است را در آن مرحله بررسی نمی کند. اگر در یک مرحله نودهای با $f(n)$ کمتر مساوی threshold نبودند یا هیچ کدام آنها نود goal نبودند threshold را افزایش می دهد و به مرحله بعدی رود. در این روش در هر مرحله همسایه های یک نود را تنگه می دارد و میزان حافظه مورد نیاز کاهش می یابد همان طور که IDS میزان حافظه کمتری نسبت به سایر الگوریتم هایی مثل BFS مصرف می کند.

SMA*(simplified memory bounded A*) & MA*(memory bounded A*): مثلاً SMA* ها ن الگوریتم A^* را اجرا می کند با این تفاوت که یک مقدار ثابتی برای حافظه در نظر می گیرد که وقتی پر شد دیگر می تواند نود جدیدی به درختمان اضافه کند. پس نودی از صف که بیش ترین مقدار f را دارد حذف می کند و مقدار جدید را در آن جایگزین می کند (البته نود حذف شده به همراه مقدار تابع f آن را به گونه ای نگهداری می کند مثلاً با استفاده از پدر آن نود تا اگر از کمترین نودهای فعلی صف به حالاتی رسیدیم که f آنها بیش از f نود حذف شده بود، یکی از آنها را حذف کند و نود حذف شده را جایگزین کند.)

سوال 3 :



• جستجو DFS : frontier یک stack است که ابتدا نود S در آن است و هواره نودی برای expand شدن از frontier انتخاب می شود که نسبت به سایر نودها در عمق بیشتری قرار دارد. از نسخه graph search الگوریتم DFS باید استفاده کنیم یعنی از فرزندان یک نود آنهایی به frontier افزوده می شوند که در frontier یا explored نباشند:

frontier	explored
{S} → top استک را سمت چپ در نظر گرفتیم	{ } پس مسیر طی شده به صورت SADHG می باشد
{A,B}	{S} و هزینه مسیر برابر 4+6+5+3 = 18 است.
{D,C,B}	{A,S}
{H,C,B}	{D,A,S}
{G,C,B} وقتی نود G برای expand شدن	{H,D,A,S}
{C,B} → انتخاب می شود چون goal است الگوریتم خاتمه می یابد.	{G,H,D,A,S}

• جستجو BFS : frontier یک صف FIFO است و هواره سطحی ترین نود frontier برای expand شدن انتخاب می شود. از نسخه graph search این الگوریتم استفاده می کنیم:

frontier	explored
{S}	{ } پس مسیر طی شده به صورت SACG است
{A,B}	{S} و هزینه مسیر برابر 4+7+12=23 است.
{B,D,C}	{A,S}
{D,C,F,E}	{B,A,S}
{C,F,E,H}	{D,B,A,S}
{F,E,H,G}	{C,D,B,A,S}
{E,H,G,G}	{F,C,D,B,A,S}
{H,G,G,I}	{E,F,C,D,B,A,S}
{G,G,I,G}	{H,E,F,C,D,B,A,S}
{G,I,G} → نود G برای expand شدن	{G,H,E,F,C,D,B,A,S}

خارج می شود چون goal است الگوریتم خاتمه می یابد.

Subject:
Year Month Date

• جست‌وجو uniform cost search : frontier یک صف اولویت است و همواره نودی برای expand شدن انتخاب می‌شود که فاصله‌ی آن تا مبدأ کمتر است:

frontier	explored
{(S,0)}	{}
{(A,4),(B,5)}	{S}
{(B,5),(D,10),(C,11)}	{A,S}
{(E,8),(D,10),(F,10),(C,11)}	{B,A,S}
{(D,10),(F,10),(C,11),(I,16)}	{E,B,A,S}
{(F,10),(C,11),(H,15),(I,16)}	{D,E,B,A,S}
{(C,11),(H,15),(I,16),(G,16)}	{F,D,E,B,A,S}
{(H,15),(I,16),(G,16)} → (G,23) فرزند C به شکل	{C,F,D,E,B,A,S}
{(I,16),(G,16)} → (G,18) فرزند H که	{H,C,F,D,E,B,A,S}
{(G,16)} → (G,20) فرزند I هم	{I,H,C,F,D,E,B,A,S}
{}	{G,I,H,C,F,D,E,B,A,S}

هزینه از مبدأ

پس مسیر طی شده به صورت SBFG است و هزینه‌ی مسیر 16 می‌باشد که optimal است.

بود که مقدار (G,16) را آپدیت می‌کند

هم مقدار (G,16) را آپدیت می‌کند

است که مقدار (G,16) را آپدیت می‌کند.

در این مرحله (G,16) برای expand شدن انتخاب می‌شود که چون goal است الگوریتم خاتمه می‌یابد.

• جست‌وجو Greedy best-first search : frontier یک صف اولویت است و همواره نودی که کمترین برآورد فاصله تا مقصد را دارد برای expand شدن انتخاب می‌شود:

frontier	explored
{S} فاصله تخمینی تا مقصد	{}
{(A,10),(B,12)}	{S}
{(D,8),(C,9),(B,12)}	{A,S}
{(H,4),(C,9),(B,12)}	{D,A,S}
{(G,0),(C,9),(B,12)}	{H,D,A,S}
{(C,9),(B,12)} → (G,0) در این مرحله نود	{G,H,D,A,S}

پس مسیر طی شده به صورت SAPHG است و هزینه‌ی مسیر برابر 4+6+5+3=18 می‌باشد

استخاب شده که چون goal بوده الگوریتم خاتمه می‌یابد.

Subject:
Year Month Date

• جست و جو A*: frontier صف اولویتی است که دوباره بودن با f(n)=g(n)+h(n) کمتر برای expand شدن انتخاب می شود:

frontier	explored
{(S,14)} ← f(n)	{ }
{(A,14),(B,17)}	{S}
{(B,17),(D,18),(C,20)}	{A,S}
{(E,14),(F,17),(D,18),(C,20)}	{B,A,S}
{(F,17),(D,18),(I,18),(C,20)}	{E,B,A,S}
{(G,16),(D,18),(I,18),(C,20)}	{F,E,B,A,S}
{(D,18),(I,18),(C,20)}	{G,F,E,B,A,S}

پس مسیر طی شده به صورت SBFG و هزینه‌ی مسیر 16 است.

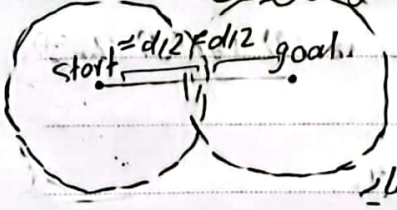
در این مرحله G انتخاب شده و الگوریتم خاتمه می یابد.

* مقادیر Heuristic داده شده admissible نیستند چون فاصله برخی نودها مانند F از H را overstimate کرده اند.

سوال 4:

بخش اول: در جست و جو دوطرفه نیاز داریم goal state مشخص باشد و الگوریتم مثلا دو الگوریتم BFS از start state و goal state اجرایی کند (به طور همزمان) و اینقدر پیش می رود تا frontier های دو الگوریتم BFS اشتراک پیدا کند. آن موقع می توانیم یک جواب از start state به goal state پیدا کنیم.

اگر در هردو جهت این الگوریتم از BFS استفاده کنیم پیچیدگی زمانی نسبت به وقتی که از start state شروع کنیم و BFS بزنیم از $O(b^d)$ به $O(b^{d/2}+b^{d/2})$ که معادل $O(2\times b^{d/2})\equiv O(b^{d/2})$ است کاهش می یابد که b همان branching factor و d عمق است که goal در آن عمق نسبت به start واقع شده است.)



بخش دوم: اگر action هایی که ما را از یک state به stateهای همسایه می برند غیر قابل بازگشت باشند می توانیم به راحتی همسایه های goal را پیدا کنیم و از goal یک BFS بزنیم. یکی دیگر از چالش های پیاده سازی این الگوریتم تشخیص زمانی است که frontier ها اشتراک و تقاطع پیدا کرده اند و می توانیم به جواب برسیم. هم چنین BFS ها از نظر حافظه هم پیچیدگی $O(b^{d/2})$ را به ارمغان می آورند که اگر جواب در عمق زیادی نسبت به start باشد به مشکل برخورد می کنیم. در صورتی که اعمال غیر قابل بازگشت باشند شاید با ساده سازی مسئله به مسئله ای با اعمال قابل بازگشت مشکل حل سوال که آن هم می تواند دشوار باشد.

Subject:
Year Month Date

سوال 2:

مورد 4: درجهتی گره ها مقدار h1 از h2 بیشتر است اگر در باقی گره ها مقدار h1 و h2 برابر باشد در این صورت h2 < h1 را dominate می کند و در هر نودی h2 < h1 است و باید حتما از h1 استفاده کنیم نه max(h1,h2) چون هزینه بیشتری جابجا شده h2 خواهیم پرداخت. اگر در باقی گره ها، گره هایی باشند که h1 < h2 باشد آنگاه تابع ترکیبی max(h1,h2) از h1 و یا h2 بهتر است و باعث گسترش تعداد کمتری از نودهای سوگ و بهتر فاصله تا مقصد را تخمین می زند. (فرض کردیم h1 و h2، admissible هستند). پس می توان این جمله را نادرست در نظر گرفت چون اگر در برخی نودها h2 < h1 باشد دلیل می سوگ در برخی دیگر h1 < h2 باشد به max نیاز داشته باشیم.

مورد 3: نادرست. استفاده از توابع هیوریستیکی که consistent هستند می تواند باعث گسترش تعداد کمتری از گره ها شود.

مورد 1: اگر h1 و h2 دو تابع هیوریستیک admissible باشند (فرض می کنیم همواره مقادیر h1 و h2 مثبت است) آنگاه تخمین هردو برای نودها از مقدار واقعی فاصله آن نود تا goal کمتر است، پس قدر مطلق اختلاف آنها هم همواره از مقدار واقعی کمتر و admissible است ← درست.

البته اگر h1 و h2 بتوانند یکی + و دیگری - باشد |h1-h2| ممکن است admissible نباشد و جمله نادرست می شود.

مورد 2: نادرست. اگر h1 و h2 توابعی consistent باشند برای هر دو نود همسایه n و n' در یک گراف بدون جهت وزن دار داریم:

$h1(n) \le w_{nn'} + h1(n')$ و $h2(n) \le w_{nn'} + h2(n')$ *

$h1(n') \le w_{nn'} + h1(n)$ و $h2(n') \le w_{nn'} + h2(n)$ **

اگر $f(n) = \min(h1(n), h2(n)) + \max(h1(n), h2(n))$ را در نظر بگیریم، چه $h1(n) \ge h2(n)$ و چه $h1(n) < h2(n)$ باشد f(n) می تواند به صورت $f(n) = h1(n) + h2(n)$ نوشته شود: اگر $h1(n) = \alpha$ و $h2(n) = \beta$ (h1 و h2 دو تابع ثابت باشند) آنگاه $f(n) = \alpha + \beta$ هم تابعی ثابت می شود. در این صورت h1 و h2 در نامساوی های خطوط * و ** صدق می کنند و consistent است. f(n) هم در دو نامساوی زیر صدق می کند و consistent خواهد بود اگر همهٔ وزن های گراف اعدادی + باشند):

$f(n) \le w_{nn'} + f(n')$ و $f(n') \le f(n) + w_{nn'}$

$\alpha+\beta \le w_{nn'} + \alpha+\beta$ $\quad \alpha+\beta \quad \alpha+\beta+w_{nn'}$