مدرس: دکتر فاطمه منصوری

نیم‌سال اول ۱۴۰۲-۱۴۰۳

تهیه کننده : فاطمه حسین زاده

هوش مصنوعی

جزوه دوم

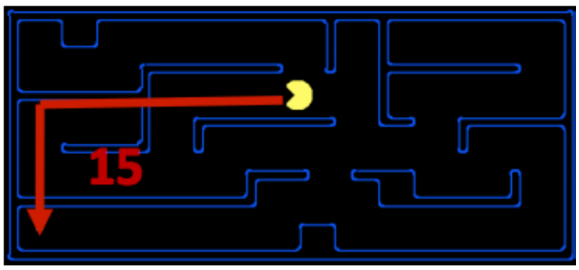
جستجوی آگاهانه:

جستجو با هزینه یکنواخت(Uniform cost search) به علت کامل بودن و بهینه بودن، الگوریتم خوبی است اما همان‌طور که در جزوه قبل اشاره کردیم ممکن است نسبتاً کند باشد زیرا برای جستجوی هدف از حالت شروع در هر جهتی گسترش می‌یابد. اگر ما آگاهی نسبی از شروع جستجو و ادامه راه داشته باشیم، می‌توانیم عملکرد را به طور قابل توجهی بهبود دهیم و به سرعت به هدف نزدیک شویم. این دقیقاً هدف جستجوی آگاهانه است.

توابع اکتشافی

روش‌های اکتشافی یا همان هیوریستیک(heuristic) نیروهای محرکه‌ای هستند که امکان تخمین فاصله تا حالت های هدف را فراهم می کنند. آنها توابعی هستند که یک حالت را به عنوان ورودی می‌گیرند و یک تخمین تا هدف را به عنوان خروجی می دهند. محاسباتی که توسط چنین توابعی انجام می شود، وابسته به مسئله جستجویی است که حل می شود. در ادامه در جستجوی A^* خواهیم دید که این توابع اکتشافی باید یک کران پایین از فاصله تا هدف را تخمین بزنند. بنابراین توابع اکتشافی معمولا راه‌حل‌هایی برای مسائل آرام شده[1] هستند(در مسئله آرام شده برخی از محدودیت‌های مسئله اصلی حذف شده‌اند). به عنوان مثال، مسئله پک من که قبلا بررسی کرده بودیم را دوباره در نظر بگیرید؛ یک روش اکتشافی رایج که برای حل این مسئله استفاده می شود فاصله منهتن است که برای دو نقطه $(x1.y1)$ و $(x2.y2)$ به صورت زیر تعریف می‌شود:

$$\text{Manhattan}(x1.y1.x2.y2) = |x1 - x2| + |y1 - y2|$$



[1] Relaxed Problems

اگر مسئله را به صورت آرام شده در نظر بگیریم، فرض می‌کنیم که هیچ دیواری در زمین بازی وجود ندارد و پک‌من می‌خواهد به نقطه پایین سمت چپ برسد. راه‌حل مسئله در این حالت فاصله منهتن از مکان فعلی پک من تا نقطه نهایی می باشد. این فاصله همان فاصله دقیق هدف در مسئله جستجوی آرام شده است، و به همین ترتیب فاصله تخمینی تا هدف در مسئله واقعی می‌باشد. با روش‌های اکتشافی، ما در عامل خود یک منطق برنامه‌ریزی برای انتخاب عمل ایجاد می‌کنیم. با استفاده از این منطق، عامل ما می‌تواند وضعیت‌هایی را که به نظر می‌رسد نزدیک‌تر به وضعیت هدف هستند، بررسی کند و این وضعیت‌ها را گسترش دهد. به این ترتیب، عامل ما به‌راحتی می‌تواند عمل مناسب برای رسیدن به وضعیت هدف را انتخاب کند. این مفهوم یکی از قدرتمندترین مفاهیم است که توسط الگوریتم‌های جستجوی حریصانه و A^* استفاده می‌شود و در این الگوریتم‌ها توابع اکتشافی را پیاده‌سازی می‌کنند. تابع اکتشافی نیز یک تابع است که در الگوریتم‌های جستجو، کران پایینی از هزینه‌ها و فاصله‌ها یا ویژگی‌های دیگر مانند زمان مورد نیاز برای رسیدن به هدف را محاسبه می‌کند.

[توابع هیوریستیک یا همان اکتشافی در جستجوی آگاهانه مفهومی است که در الگوریتم‌های جستجو مورد استفاده قرار می‌گیرد. این توابع اکتشافی به عنوان یک کران پایین عمل می‌کنند به طوری‌که اطلاعاتی درباره مسئله و دامنه جستجوی الگوریتم به ما می‌دهند. کار این توابع به اصطلاح مشاوره دادن و راهنمایی کردن الگوریتم است تا به کمک آن بهترین تصمیم را در هر مرحله بگیرد.

با استفاده از تابع اکتشافی، الگوریتم جستجو می‌تواند عملکرد بهتری در پیدا کردن راه‌حل داشته باشد. این تابع معمولاً بر اساس اطلاعات مسئله و مسئولیت الگوریتم تعیین می‌شود. به عنوان مثال، در مسئله جستجوی بهترین مسیر بین دو نقطه در یک نقشه، فاصله فیزیکی بین دو نقطه می‌تواند به عنوان تابع اکتشافی استفاده شود. این تابع بر اساس فاصله‌ای که باید طی شود، کران پایینی از کمیت‌های مسئله را به الگوریتم می‌دهد.

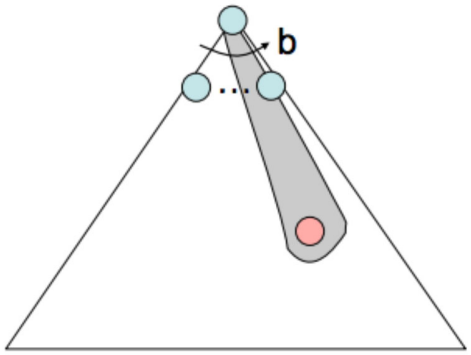
این توابع از اطلاعات جمع‌آوری شده در هر مرحله و کاهش فضای جستجو برای رسیدن به راه‌حل بهینه استفاده می‌کنند.]

- جستجوی حریصانه:

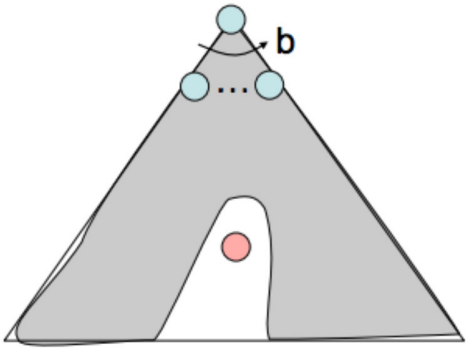
توضیح: یک استراتژی برای جستجو است که همیشه گره مرزی را با کمترین مقدار اکتشافی برای گسترش انتخاب می کند، که مربوط به حالتی است که معتقد است نزدیکترین حالت به یک هدف است.

نمایش حد: جستجوی حریصانه مانند UCS عمل می کند؛ با یک صف اولویت از نماینده های مرزی. با این تفاوت که به جای استفاده از هزینه معکوس (از گره شروع تا مکانی که عامل در حال حاضر در آن قرار دارد)(یا همان مجموع وزن یالهای در مسیر یک حالت) برای تعیین اولویت، جستجوی حریصانه از هزینه تخمینی پیش‌رو به صورت مقادیراکتشافی استفاده می‌کند.

کامل و بهینه بودن: جستجوی حریصانه تضمینی برای یافتن حالت هدف در صورت وجود یا بهینه بودن حالت پیدا شده نمی‌کند، به ویژه در مواردی که یک تابع اکتشافی بسیار بد انتخاب شده باشد. به طور کلی از سناریویی به سناریوی دیگر به‌طور غیرقابل پیش بینی عمل می‌کند.



الف: جستجوی حریصانه در حالت خوب



ب: جستجوی حریصانه در حالت بد

[جستجوی حریصانه یک الگوریتم جستجو است که در هر مرحله براساس ارزیابی اکتشافی، بهترین گزینه ممکن را انتخاب می‌کند و به سمت آن حرکت می‌کند. در این الگوریتم، هیچ بهبود قبلی‌ای در نظر گرفته نمی‌شود و فقط از اطلاعات فعلی بهره می‌برد. به همین دلیل نیز جستجوی حریصانه به عنوان یک الگوریتم مبتنی بر قرارداد بدون بازگشت (contract-based algorithm) شناخته می‌شود.

استفاده از توابع اکتشافی در جستجوی حریصانه اهمیت بسیار بالایی دارد. توابع اکتشافی، تخمینی برای میزان فاصله تا هدف یا درک عملکرد هدف در هر مرحله‌ی جستجو ارائه می‌دهند. این توابع باید قادر به سنجش و تخمین بهترین گزینه ممکن برای رفتن به نقطه هدف باشند.

مزیت استفاده از الگوریتم حریصانه در سرعت عملیات و اجرای سریع است. به دلیل انتخاب بهترین گزینه ممکن در هر مرحله، الگوریتم به سرعت به نقطه هدف نزدیک می‌شود. اما مشکل اصلی الگوریتم حریصانه در این است که اغلب در یک موقعیت محلی نیست، به این معنی که هنگامی که به یک نقطه هدف می‌رسد، ممکن است نتواند گزینه‌ی بهتری برای رسیدن به یک هدف دیگر پیدا کند.

بنابراین، جستجوی حریصانه فقط بهترین گزینه در هر مرحله را در نظر می‌گیرد و به سمت آن جهت می‌دهد. به‌طور خلاصه، جستجوی حریصانه الگوریتمی ساده و سریع است که بر پایه‌ی تخمین تابع اکتشافی بهترین گزینه را در هر مرحله انتخاب می‌کند؛ اما این الگوریتم دچار مشکل بهینگی محلی است و قادر به پیدا کردن بهینه‌ترین مسیر در تمام فضای جستجو نیست.]

- جستجوی A^*:

توضیح: جستجوی A^* یک استراتژی برای جستجو است که همیشه گره مرزی را با کمترین تخمین هزینه کل، برای رسیدن به گره هدف انتخاب می‌کند، که در آن هزینه کل، مجموع هزینه رسیدن از گره شروع تا گره هدف است.

نمایش حد: مانند جستجوی حریصانه و جستجوی UCS، جستجوی A^*نیز از یک صف اولویت برای ذخیره نمایندگان مرزی استفاده می کند. تفاوت اصلی در روش انتخاب اولویت است. جستجوی A^*همزمان از هزینه معکوس (جمع وزن یال‌ها در مسیر یک حالت) که توسط UCS استفاده می شود، و هزینه تخمینی پیشرو (مقدار اکتشافی) که توسط جستجوی حریصانه استفاده می شود ، استفاده می کند. با جمع این دو مقدار، هزینه تخمینی کل از شروع تا هدف به صورت مؤثر محاسبه می شود. به عبارت دیگر، با هدف کاهش کل هزینه تخمینی از شروع تا هدف، این رویکرد یک راه حل بسیار خوب است.

کامل و بهینه بودن: جستجوی A^* با استفاده از یک تابع اکتشافی مناسب، هم کامل و هم بهینه است(در جلوتر به آن اشاره می کنیم). این روش یک ترکیب از خوبی های همه استراتژی های جستجوی دیگر است که ما تا کنون پوشش داده‌ایم، که در کل سرعت بالای جستجوی حریصانه را با بهینه و کامل بودن UCS را در خود دارد!

پذیرش و سازگاری

با توجه به اینکه در مورد توابع اکتشافی و نحوه استفاده از آنها در الگوریتم‌های جستجوی حریصانه و A^*بحث کردیم، اکنون بیایید ببینیم چه شرایطی یک تابع اکتشافی مناسب را تشکیل می‌دهد. برای این منظور، ابتدا روش‌های استفاده شده برای تعیین ترتیب اولویت در الگوریتم هزینه یکنواخت (UCS)، جستجوی حریصانه و الگوریتم A^*را با استفاده از تعاریف زیر دوباره بیان کنیم:

- $g(n)$: تابعی است که کل هزینه از شروع تا این لحظه(هزینه معکوس) را محاسبه می‌کند و توسط UCS نشان داده می‌شود.
- $h(n)$: تابعی است که مقدار اکتشافی یا هزینه تخمینی پیش‌رو که توسط جستجوی حریصانه استفاده می‌شود را نشان می دهد.
- $f(n)$: تابعی است که کل هزینه تخمینی که توسط جستجوی A^* استفاده می‌شود را نشان می دهد. $f(n) = g(n) + h(n)$.

قبل از بحث در مورد اینکه چه چیزی یک هیوریستیک(تابع اکتشافی) "خوب" را تشکیل می‌دهد، باید ابتدا به این سؤال که آیا A^* ویژگی‌های کامل بودن و بهینه بودن خود را بدون توجه به تابع هیوریستیکی که استفاده می‌کنیم حفظ می‌کند، پاسخ دهیم. به راحتی می‌توان هیوریستیک‌هایی را پیدا کرد که این دو ویژگی ارزشمند را نقض می‌کنند. به عنوان مثال، تابع هیوریستیک(اکتشافی) $= h(n) = 1 - g(n)$ را در نظر بگیرید. صرف نظر از مسئله جستجو، با استفاده از این تابع داریم:

$$f(n) = g(n) + h(n)$$
$$= g(n) + (1 - g(n))$$
$$= 1$$

بنابراین، این الگوریتم، محدودیتی را بر روی جستجوی A^* می‌گذارد و آن را به جستجوی BFS تبدیل می‌کند، جایی که وزن همه یال‌ها معادل هم هستند. همانگونه که قبلاً نشان دادیم، در شرایط کلی که وزن‌های یال‌ها ثابت نیستند، BFS بهینه بودن را تضمین نمی کند.

شرط مورد نیاز برای بهینه‌سازی هنگام استفاده از جستجوی درختی A^* به عنوان "پذیرش"[2]، یک نوع محدودیت سازگاری شناخته می‌شود. این محدودیت سازگاری می‌گوید که مقدار تخمین‌زده‌شده توسط یک هیوریستیک سازگار، نباید هیچ‌گاه منفی و یا بزرگتر از مقدار واقعی باشد. به عبارت دیگر، مقدار تخمین‌زده‌شده باید به عنوان یک حد پایین قابل قبول برای هزینه بهینه‌ی حقیقی در

[2] admissibility

رسیدن به یک حالت هدف مورد قبول باشد. با تعریف $h^*(n)$ به عنوان هزینه بهینه پیش‌رو حقیقی برای رسیدن به یک حالت هدف از یک گره معین n، می توانیم محدودیت پذیرش را به صورت ریاضی به صورت زیر بیان کنیم:

$$\forall n.\ 0 \leq h(n) \leq h^*(n)$$

قضیه: برای یک مسئله جستجوی داده شده، اگر محدودیت پذیرش توسط یک تابع اکتشافی h برآورده شود، با استفاده از جستجوی درختی A^* با h در آن مسئله جستجو، یک راه حل بهینه به دست می‌آید.

[در صورتی که جستجوی درختی A^* با تابع اکتشافی h بر روی یک مسئله جستجوی داده شده، یک راه حل بهینه را ارائه دهد، این به این معنی است که راه حل پیدا شده تضمین می‌کند بهترین (بهینه) راه حل بر اساس محدودیت‌ها و تابع هزینه مسئله داده شده است.

در جستجوی درختی A^* ، الگوریتم گره‌ها را بر اساس تابع اکتشافی‌شان گسترش می‌دهد، که ترکیبی از هزینه تاکنون $(g(n))$ و تخمین هزینه $(h(n))$ است تا گره‌ها را برای گسترش اولویت ببخشد. تابع اکتشافی h قابل پذیرش است اگر هرگز هزینه رسیدن به حالت هدف از یک حالت داده شده را بالاتر از مقدار واقعی تخمین نزند.

هنگامی که محدودیت پذیرشی تابع اکتشافی h توسط تابع هزینه h برآورده می شود، جستجوی درختی A^*تضمین می کند که اولین راه حل پیدا شده بهترین راه حل خواهد بود. این به این معنی است که برای هر گره گسترش یافته، هزینه ترکیبی برای رسیدن به آن گره به علاوه هزینه تخمینی از آن گره تا هدف $(f(n) = g(n) + h(n))$همیشه بهترین کران پایین هزینه بهینه است. بنابراین، راه حل پیدا شده توسط جستجوی درختی A^*با تابع ارزیابی پذیرش شده بهینه خواهد بود زیرا کمترین مسیر هزینه‌بر برای رسیدن به هدف است.]

اثبات: فرض کنید دو حالت هدف قابل دستیابی در درخت جستجو برای یک مسئله جستجوی داده شده داریم، که یک هدف بهینه A و یک هدف غیربهینه B است. برخی از n اجداد A (که ممکن است خود A هم باشد) در حال حاضر باید در مرز باشند، زیرا A از حالت شروع قابل دسترسی است(n شامل بعضی از نسل‌های قبلیA (شامل خود A)است که در حال حاضر باید در محدوده شروع قرار داشته باشند زیرا A قابل دسترسی از حالت شروع است.). ما ادعا می کنیم n برای گسترش قبل از B انتخاب خواهد شد. با استفاده از سه عبارت زیر داریم:

۱. $g(A) < g(B)$: چون A بهینه و B غیر بهینه به ما داده شده، می توانیم نتیجه بگیریم که A هزینه معکوس کمتری از حالت شروع نسبت به Bدارد.

۲. $h(A) = h(B) = 0$: زیرا هیوریستیک ما که محدودیت پذیرش را برآورده می کند، به ما داده می شود. از آنجاییکه A و B هر دو حالت هدف هستند، پس هزینه بهینه حقیقی برای یک حالت هدف از A یا B به سادگی برابر $h^*(n) = 0$ است.بنابراین $0 \leq h(n) \leq 0$.

۳. $f(n) \leq f(A)$: چون از طریق قابل قبول بودن h، $f(n) = g(n) + h(n) \leq g(n) + h^*(n) = g(A) = f(A)$ ، هزینه کل از طریق گره n حداکثرهزینه معکوس واقعی A است که هزینه کل A نیز می باشد.

با ترکیب قسمت های ۱ و ۲ میتوانیم نتیجه بگیریم که $f(A) < f(B)$:

$$f(A) = g(A) + h(A) = g(A) < g(B) = g(B) + h(B) = f(B)$$

یک نتیجه ساده حاصل از نتیجه بالا و قسمت ۳ به صورت زیر است:

$$f(n) \leq f(A) \wedge f(A) < f(B) \Longrightarrow f(n) < f(B)$$

پس نتیجه می گیریم که n زودتر از B گسترش پیدا می کند، زیرا اثبات کردیم که برای یک n دلخواه، تمامی اجداد A (که شامل خود A نیز می‌تواند باشد) قبل از B گسترش پیدا می‌کند.

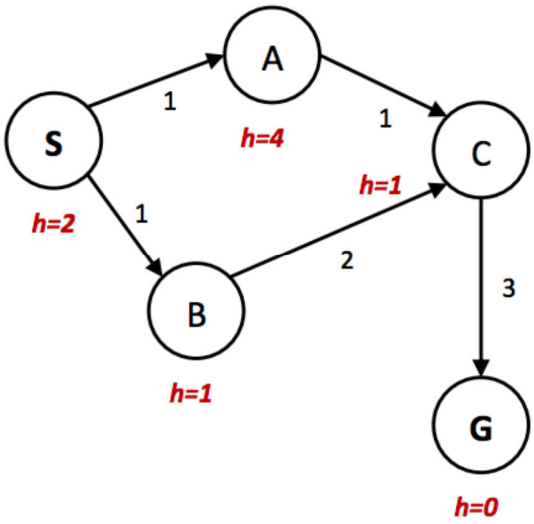
یکی از مشکلاتی که در بالا با جستجوی درختی پیدا کردیم، این بود که در برخی موارد ممکن است هرگز راه حلی پیدا نشود و در دور بی-نهایت در نمودار فضای حالت گراف، گیر بیفتد. حتی در شرایطی که روش جستجوی ما شامل چنین دورهای بی نهایت نمی‌شود، اغلب ممکن است که ما دوباره از همان گره چندین بار بازدید کنیم، زیرا راه های متعددی برای رسیدن به همان گره وجود دارد. این کار باعث افزایش تصاعدی کار ما می شود، و راه حل طبیعی این است که بررسی کنیم که کدام حالت‌ها را قبلاً گسترش داده‌ایم و هرگز دوباره آنها را گسترش ندهیم. به طور واضح تر، هنگام استفاده از جستجوی خود، مجموعه‌ای از گره های گسترش یافته را ذخیره کنیم. سپس، بررسی کنیم که هر گره از قبل در این مجموعه گسترش نیافته باشد و اگر نبود آن را بعد از گسترش، به مجموعه اضافه کنیم. جستجوی درختی با این روش بهینه سازی به عنوان **جستجوی گرافی**[۳] شناخته می شود وشبه کد آن در زیر آمده است:

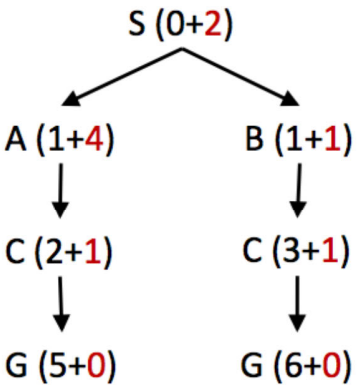
```
function GRAPH-SEARCH(problem, frontier) return a solution or failure
    reached ← an empty set
    frontier← INSERT(MAKE-NODE(INITIAL-STATE[problem]), frontier)
    while not IS-EMPTY(frontier) do
        node ← POP(frontier)
        if problem.IS-GOAL(node.STATE) then return node
        end if
        if node.STATE is not in reached then
            add node.STATE in reached
            for each child-node in EXPAND(problem, node) do
                frontier ← INSERT(child-node, frontier)
            end for
        end if
    end while

    return failure
```

توجه کنید که در پیاده سازی، بسیار مهم است که مجموعه گسترش یافته شده را به عنوان یک مجموعه مجزا ذخیره کنید و نه یک لیست. ذخیره آن به عنوان یک لیست نیاز به $O(n)$ عملیات برای بررسی عضویت دارد که این تعداد عملیات، بهبودی را که برای جستجوی گرافی ایجاد کرده بودیم حذف می کند. یک نکته دیگر در مورد جستجوی گرافی این است که بهینه بودن جستجوی A^* را حتی با وجود یک تابع اکتشافی قابل قبول از بین می برد. گراف فضای حالت ساده زیر و درخت جستجوی مربوطه، که دارای وزن و مقادیر اکتشافی است را در نظر بگیرید:

---

[۳] در دوره های دیگر، مانند CS70 و CS170، ممکن است در زمینه نظریه گراف با "درخت" و "گراف" آشنا شده باشید.به طور خاص، یک درخت نوعی گراف است که محدودیت های خاصی (متصل و بدون حلقه بودن) را دارد. این تمایزی بین جستجوی درختی و جستجوی گرافی که در این دوره ایجاد می کنیم نیست.





در مثال بالا، مشخص است که مسیر بهینه $S \to A \to C \to G$است که هزینه کل مسیر معادل با $1 + 1 + 3 = 5$ را به همراه دارد.
مسیر دیگر برای رسیدن به هدف، مسیر $S \to B \to C \to G$ با هزینه$1 + 2 + 3 = 6$ است.

از آنجایی که مقدار اکتشافی گره A بسیار بزرگتر از مقدار اکتشافی گره B است، پس گره C در امتداد مسیر دوم و غیربهینه به عنوان فرزند اول گره B گسترش می یابد. سپس در مجموعه "گسترش شده ها" قرار می گیرد، و بنابراین جستجوی گرافی A^* وقتی به عنوان فرزند A از آن بازدید می‌کند، نمی‌تواند آن را دوباره بسط دهد، بنابراین هرگز راه‌حل بهینه را پیدا نمی‌کند. از این رو، برای حفظ بهینه بودن در جستجوی گرافی A^*، به ویژگی قوی‌تری نسبت به قابلیت "پذیرش" نیاز داریم، یعنی ویژگی سازگاری[4]. ایده اصلی سازگاری این است که ما نه تنها این را اعمال کنیم که یک تابع اکتشافی، کل فاصله تا یک هدف از هر گره داده شده را دست کم برآورد کند، بلکه هزینه/وزن هر یال در گراف را نیز اعمال کند.

هزینه یک یال که توسط تابع اکتشافی اندازه‌گیری می شود برابر تفاوت دو مقدار اکتشافی برای دو گره متصل است. از نظر ریاضی، محدودیت سازگاری را می توان به صورت زیر بیان کرد:

$$\forall A.C\ h(A) - h(C) \leq cost(A.C)$$

قضیه: برای یک مسئله جستجوی داده شده، اگر محدودیت سازگاری توسط یک تابع اکتشافی h برآورده شود، استفاده از جستجوی گرافی A^* با h در آن مسئله، یک راه حل بهینه را به ما می دهد.

اثبات: برای اثبات این قضیه، ابتدا باید ثابت کنیم که هنگام اجرای جستجوی گرافی A^* با یک تابع اکتشافی ثابت، هر زمان که گره ای را برای گسترش حذف می‌کنیم، مسیر بهینه آن گره را پیدا کرده ایم.

با استفاده از محدودیت سازگاری، می‌توانیم نشان دهیم که مقادیر $f\ (n)$ برای گره‌ها در امتداد هر مسیری غیرکاهشی هستند.دو گره n و n' را تعریف می کنیم که در آن n' فرزندی از n است. سپس داریم:

$$f\ (n') = g(n') + h(n')$$
$$= g(n) + cost(n.n') + h(n')$$

[4] consistency

$$\geq g(n) + h(n)$$

$$= f\ (n)$$

اگر برای هر والد-فرزند دوتایی $(n\ .\ n')$ در طول یک مسیر، $f\ (n') \geq f\ (n)$ باشد، پس باید مقادیر $f\ (n)$در طول آن مسیر بدون کاهش باشند. می‌توانیم بررسی کنیم که گراف بالا این قانون را بین $f\ (A)$ و$f\ (C)$ نقض می‌کند. با این اطلاعات، اکنون می‌توانیم نشان دهیم که هرگاه یک گره n برای گسترش حذف شود، مسیر بهینه پیدا شده است.

با برهان خلف فرض می کنیم که این نادرست باشد – یعنی وقتی گره n از مرز حذف شود، مسیر یافت شده به n بهینه نیست. این بدان معنی است که باید برخی از اجداد مانند n، $n"$ در مرز وجود داشته باشند که هرگز گسترش نیافته اند اما در مسیر بهینه برای n قرار دارند.که این یک تناقض است! زیرا قبلاً نشان داده‌ایم که مقادیر f در طول یک مسیر کاهش نمی‌یابند، بنابراین $n"$ برای گسترش قبل از n حذف می‌شود.

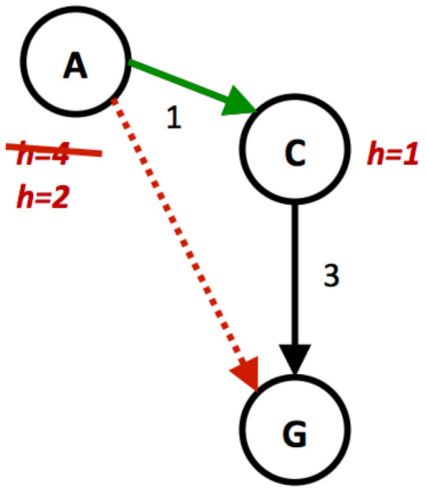
تنها یک چیز برای اثبات باقی مانده است، و آن، این است که یک هدف بهینه A همیشه برای گسترش حذف می شود و قبل از هر هدف غیربهینه B برگردانده می شود. که این بدیهی است زیرا می دانیم $h(A) = h(B) = 0$ پس داریم:

$$f\ (A) = g(A) < g(B) = f\ (B)$$

همانند اثبات بهینه بودن جستجوی درختی A^* تحت محدودیت پذیرش. از این رو، می‌توان نتیجه گرفت که جستجوی گرافی A^* تحت یک تابع اکتشافی سازگار بهینه است.

چند نکته مهم از قضیه بالا: برای معتبر بودن تابع‌های اکتشافی قابل پذیرش/سازگار، طبق تعریف باید برای هر حالت هدف G، $h(G) = 0$ باشد. علاوه بر این، سازگاری فقط یک محدودیت قوی‌تر از پذیرش نیست، بلکه سازگاری مستلزم پذیرش است ..این ازاین واقعیت ناشی می شود که اگر هزینه هیچ یالی بیش از حد تخمین زده نشود (که توسط سازگاری تضمین می شود)، کل هزینه برآورد شده از هر گره برای یک هدف نیز بیش از حد برآورد نخواهد شد.

شبکه سه گره زیر را برای مثالی از یک تابع اکتشافی قابل قبول اما ناسازگار در نظر بگیرید:



خط-نقطه قرمز مربوط به کل فاصله هدف برآورد شده است .اگر$h\ (A) = 4$ ، آنگاه تابع اکتشافی قابل قبول است، زیرا فاصله A تا هدف $h(A) \geq 4$ است و برای $h(C) = 1 \leq 3$ یکسان است. با این حال، هزینه اکتشافی از A تا C، برابر $h(A) - h(C) = 4 -$ برابر

$3 = 1$است. تابع اکتشافی ما هزینه یال بین A و C را ۳ تخمین می زند در حالی که مقدار واقعی هزینه ۱ است که مقدار کمتری است. از آنجایی که $\mathrm{h(A)} - \mathrm{h(C)} \nleq \mathrm{cost(A.C)}$ این تابع اکتشافی سازگار نیست. همان محاسبات را برای $\mathrm{h(A)} = 2$ ، انجام می‌دهیم، داریم :

$$\mathrm{h(A)} - \mathrm{h(C)} = 2 - 1 = 1 \leq \mathrm{cost(A.C)}$$

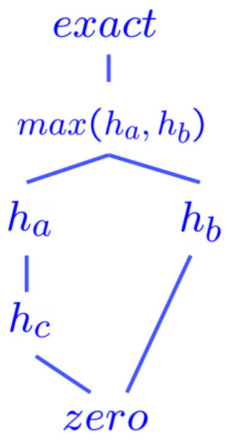
بنابراین، با استفاده از $\mathrm{h(A)} = 2$ تابع اکتشافی ما سازگارمی شود.

برتری[5]

اکنون که ویژگی‌های پذیرش و سازگاری و نقش آن‌ها در حفظ بهینه بودن جستجوی $\mathrm{A^*}$ را مشخص کردیم، می‌توانیم به مشکل اصلی خود در ایجاد تابع اکتشافی «خوب» و چگونگی تشخیص بهتر بودن یک تابع اکتشافی از دیگری برگردیم. معیار استاندارد برای این منظور، برتری است .اگر تابع اکتشافی a غالب بر تابع اکتشافی b باشد، پس فاصله هدف تخمینی برایa، بزرگتر از فاصله هدف تخمینیb، برای هر گره در گراف فضای حالت است. این از نظر ریاضی یعنی:

$$\forall n : h_a(n) \geq h_b(n)$$

برتری به طور کاملاً شهودی ایده یک موجود اکتشافی بهتر از دیگری را در بر می گیرد – اگر یک تابع اکتشافی قابل پذیرش/ سازگاری بر دیگری مسلط باشد، باید بهتر باشد، زیرا همیشه فاصله تا یک هدف را از هر حالت داده شده با دقت بیشتری تخمین می زند .علاوه بر این، **هیوریستیک بدیهی**[6]، به صورت $h(n) = 0$ تعریف می شود و استفاده از آن جستجوی $\mathrm{A^*}$ را به UCS کاهش می دهد .همه هیوریستیک‌های قابل قبول بر هیوریستیک بدیهی غالب هستند .هیوریستیک بدیهی اغلب در پایه یک **نیمه شبکه**[7]برای یک مسئله جستجو گنجانده می شود، یک سلسله مراتب برتر که در پایین قرار می‌گیرد . شکل زیر نمونه‌ای از یک نیمه شبکه است که شامل هیوریستیک‌های مختلفh_a، h_b، و h_c است که از هیوریستیک بدیهی در پایین تا فاصله هدف دقیق در بالا را شامل می‌شود:



به عنوان یک قاعده کلی، عملکرد بیشینه کردن توابع اکتشافی چندگانه قابل قبول، همیشه قابل پذیرش است. این فقط یک نتیجه از این است که تمام مقادیر خروجی یک استراتژی در هر حالت توسط شرط توجیهی، $0 \leq \mathrm{h(n)} \leq \mathrm{h^*(n)}$، محدود می‌شوند. بزرگترین عدد در این

[5] Dominance
[6] trivial heuristic
[7] semi-lattice

محدوده نیز باید در همان محدوده قرار بگیرد. در واقع، هدف ما در تولید استراتژی مناسب برای حل یک مسئله جستجو، از بین تمام استراتژی‌های منطقی، انتخاب یک استراتژی با عملکرد بهترین مقدار است.

جستجو: خلاصه

در این یادداشت، درباره مسئله‌های جستجو و اجزای آنها صحبت کردیم: یک فضای حالت، مجموعه ای از اعمال، یک تابع انتقال، یک هزینه عمل، یک حالت شروع و یک وضعیت هدف .عامل از طریق حسگرها و محرک های خود با محیط تعامل دارد .تابع عامل توصیف می کند که عامل در هر شرایطی چه کاری کند .عقلانیت عامل به این معنی است که عامل به دنبال به حداکثر رساندن مطلوبیت مورد انتظار خود است . در نهایت، ما محیط های وظیفه خود را با استفاده از توضیحات PEAS تعریف می کنیم.

با توجه به مسئله های جستجو، آنها را می توان با استفاده از انواع تکنیک های جستجو، اما نه محدود با استفاده از پنج مورد زیر حل کرد:

- جستجوی اول سطح(Breadth-first Search)
- جستجوی اول عمق(Depth-first Search)
- جستجو با هزینه یکنواخت(Uniform Cost Search)
- جستجوی حریصانه(Greedy Search)
- جستجوی A^*

سه تکنیک جستجوی اول ذکر شده در بالا نمونه هایی از جستجوی ناآگاهانه هستند، در حالی که دو روش آخر نمونه هایی از جستجوی آگاهانه هستند که از روش های هیوریستیک برای تخمین فاصله هدف و بهینه‌سازی عملکرد استفاده می کنند.

ما همچنین بین الگوریتم‌های جستجوی درختی و جستجوی گرافی برای تکنیک‌های بالا تمایز قائل شدیم.