هوش مصنوعی

حسین کارشناس

دانشکده ریاضی

ترم اول ۹۴ – ۹۳

جستجوی آگاهانه

جستجوی آگاهانه

- استفاده از اطلاعات خاص مسأله علاوه بر تعریف مسأله
- رویکرد کلی: جستجوی بهترین اول (best-first)
  - انتخاب گره‌ها برای بسط بر اساس یک تابع ارزیابی (f(n))
    - یک تخمین از هزینه هر گره (برای انتخاب گره با کمترین هزینه)
  - مانند الگوریتم جستجو با هزینه یکنواخت و جایگزینی g(n) با f(n)
- تابع اکتشافی (heuristic) – h(n): مولفه‌ای از تابع ارزیابی که اطلاعات اضافی در مورد مسأله را به الگوریتم جستجو می‌دهد
  - h(n): هزینه تخمین زده شده برای ارزانترین مسیر از گره n تا گره هدف
  - فقط به حالت مرتبط با گره n بستگی دارد
  - هر تابع دلخواه و نامنفی با **شرط** h(n)=0 اگر n گره هدف باشد

جستجوی بهترین اول حریصانه (greedy best-first)

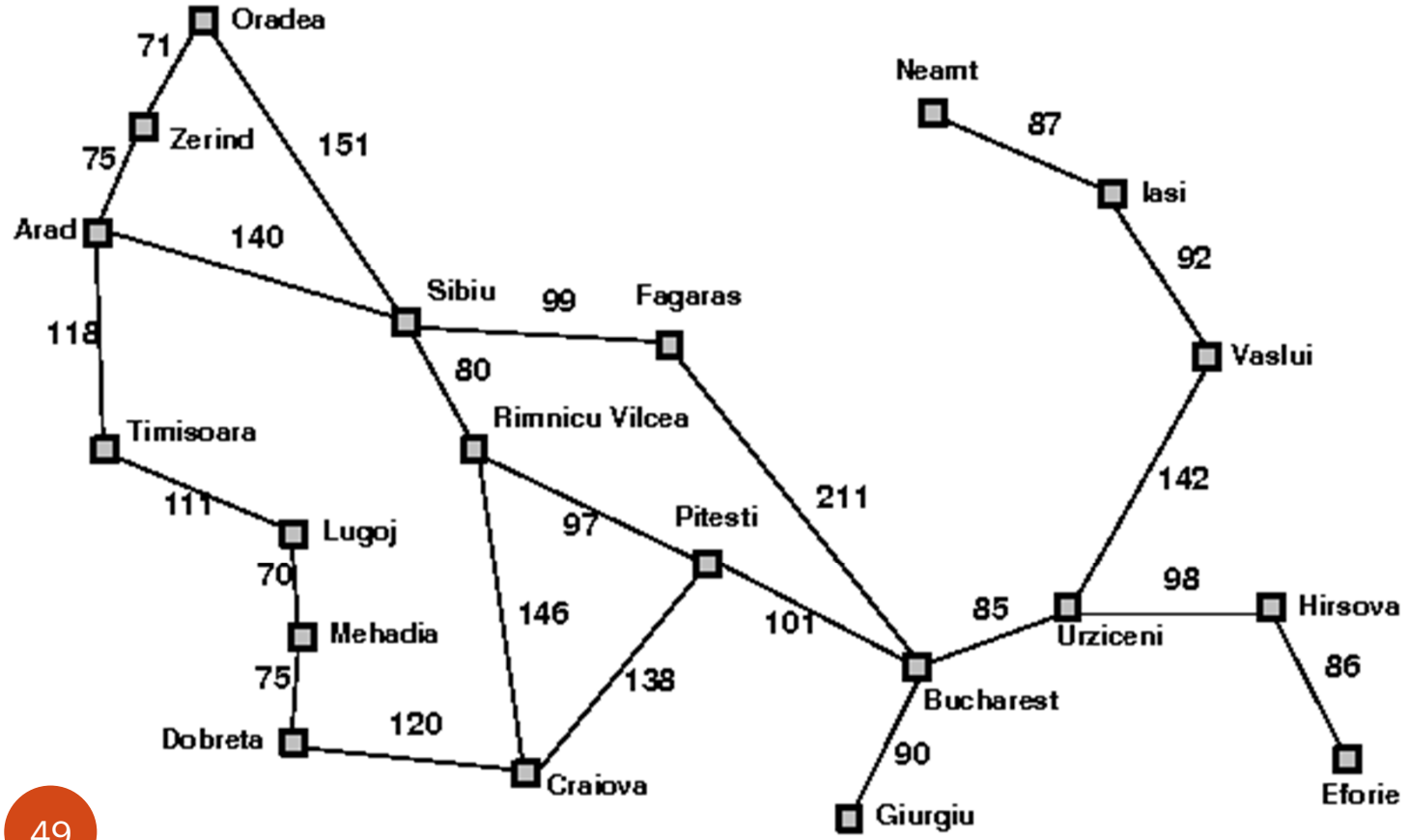
- هر بار نزدیک‌ترین گره به هدف را بسط می‌دهد
- تابع ارزیابی: معادل تابع اکتشافی در نظر گرفته می‌شود

$$f(n)=h(n)$$

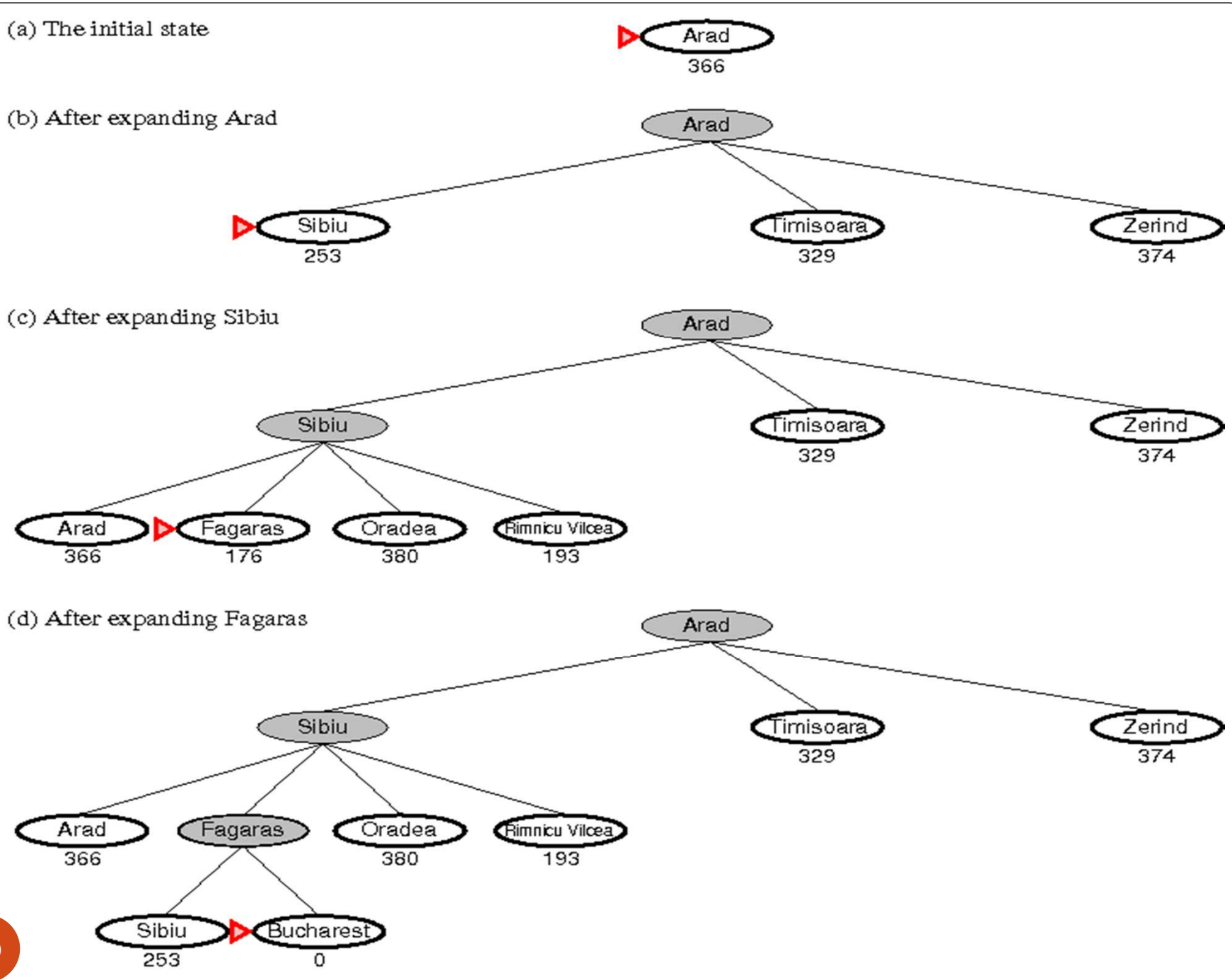
- مثال: مسیریابی در رومانی
  - تابع اکتشافی: فاصله خط مستقیم $h_{SLD}(n)$
    - قابل محاسبه از تعریف مسأله نیست
    - مربوط بودن این تابع اکتشافی با فاصله از طریق راه‌ها بصورت مستقیم قابل برداشت نیست و نیاز به تجربه دارد

جستجوی بهترین اول حریصانه

- مسیریابی در رومانی



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374


(a) The initial state
Arad
366
(b) After expanding Arad
Arad
Sibiu
253
Timisoara
329
Zerind
374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
176
Oradea
380
Rimnicu Vilcea
193
(d) After expanding Fagaras
Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
Oradea
380
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0

جستجوی بهترین اول حریصانه

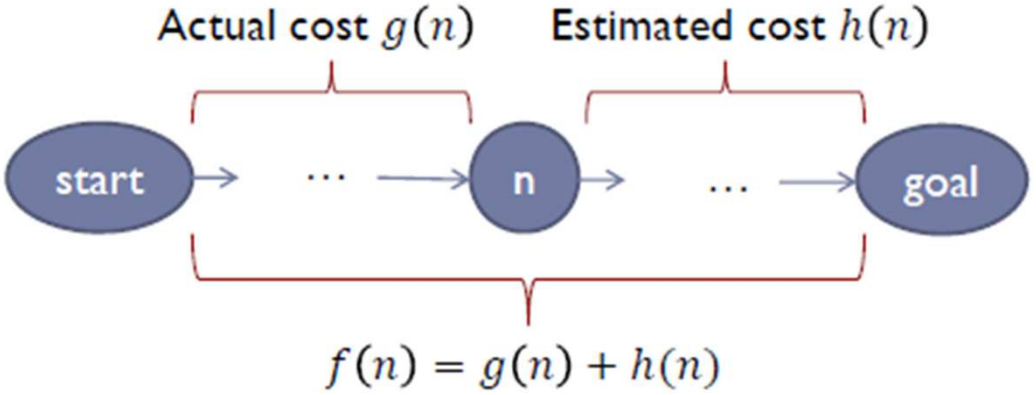
- ویژگی‌ها
  - نسخه مبتنی بر جستجوی درخت کامل نیست: حلقه‌های نامتناهی
    - مثال: رسیدن از شهر Ias به Fagaras
    - نسخه مبتنی بر جستجوی گراف در فضاهای حالت متناهی کامل است
  - بهینه نیست: یک الگوریتم حریصانه است
  - پیچیدگی زمانی: $O(b^m)$
    - با استفاده از یک تابع اکتشافی خوب می‌تواند بطور قابل توجهی کاهش پیدا کند
  - پیچیدگی فضایی: $O(b^m)$

جستجوی A^*

- کمینه کردن کل هزینه تخمین زده شده‌ی راه‌حل
- تابع ارزیابی: ترکیب هزینه‌ی از ریشه تا گره و هزینه‌ی گره تا هدف

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

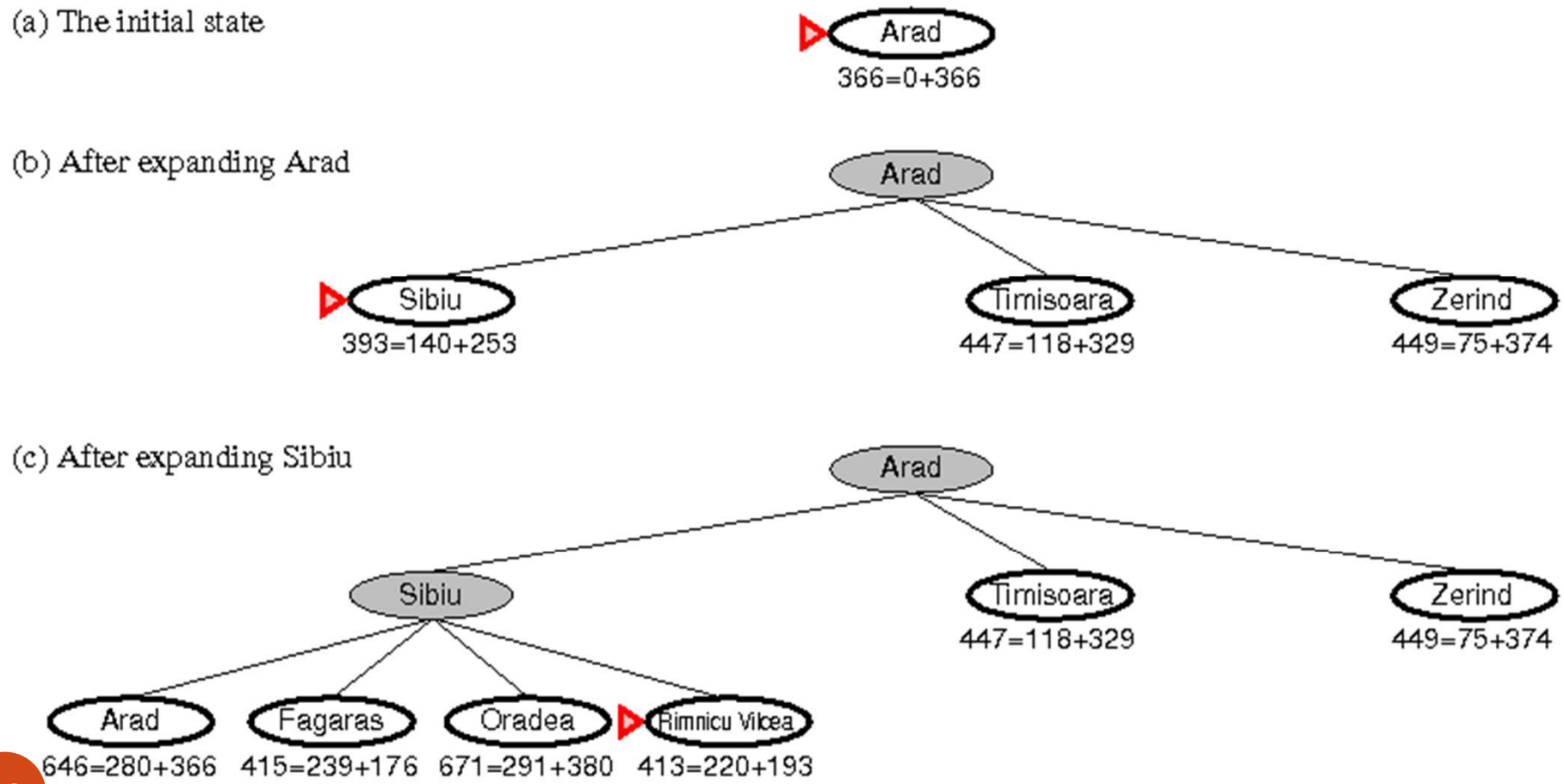
- هزینه تخمین زده شده برای ارزانترین راه‌حلی که از گره n می‌گذرد



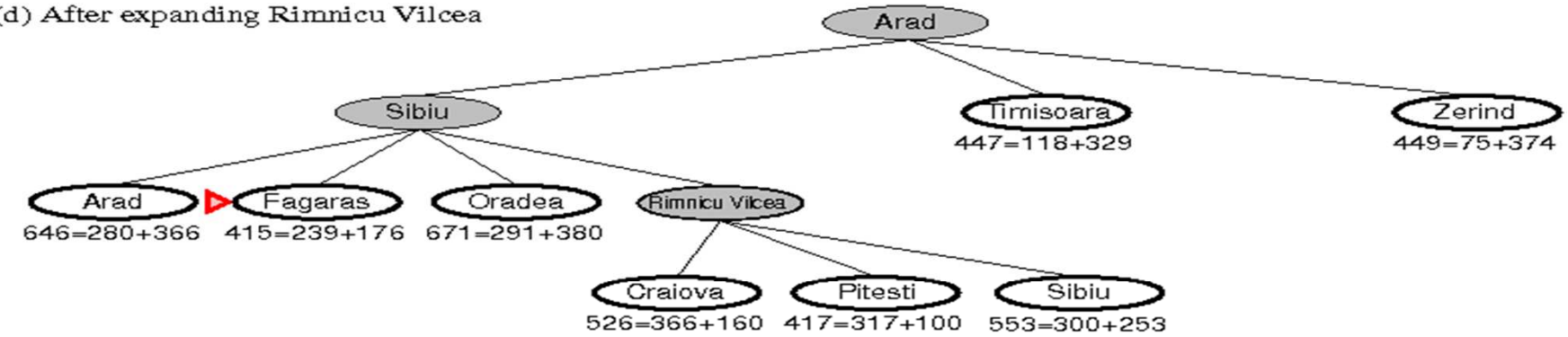
- مانند الگوریتم جستجو با هزینه یکنواخت و جایگزینی g(n) با g(n)+h(n)

جستجوی A^*

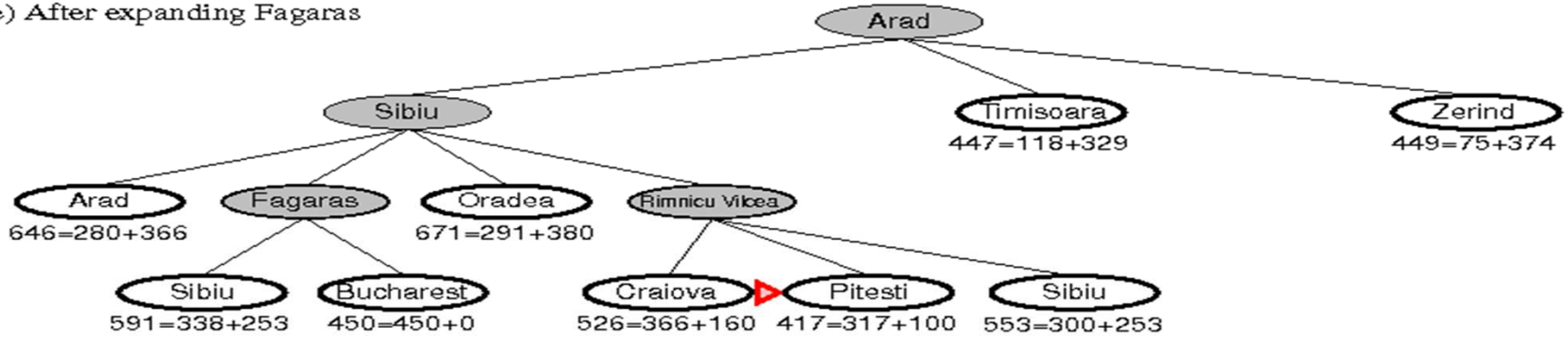
- مثال: مسیریابی در رومانی با تابع اکتشافی فاصله خط مستقیم $h_{SLD}(n)$



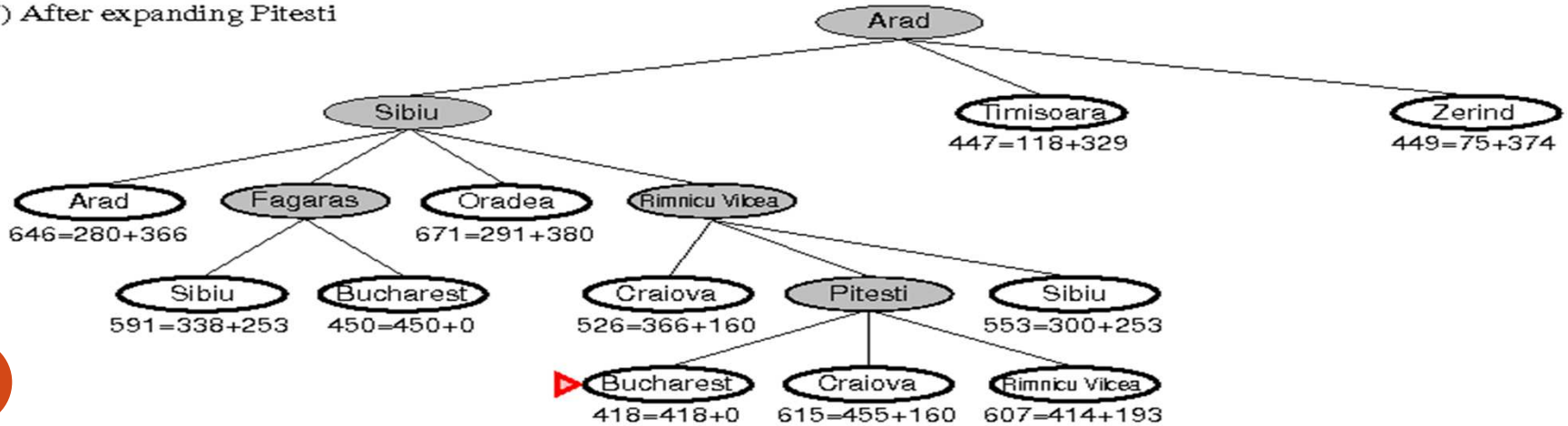
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras



(f) After expanding Pitesti



جستجوی A^*

- شرایط بهینگی
  - تابع اکتشافی پذیرفتنی (admissible) باشد: هزینه رسیدن به هدف را برای هر گره بیش از اندازه برآورد نکند

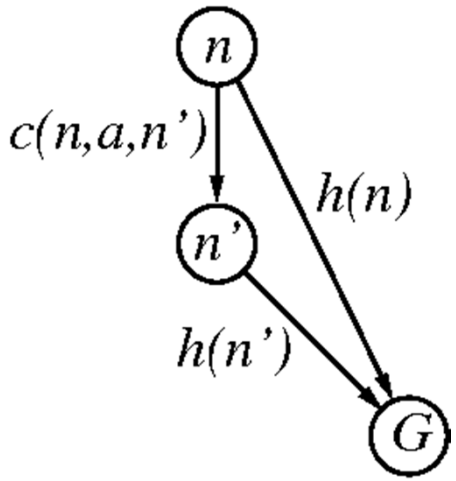
$$h(n) \leq h^*(n)$$

$h^*(n)$ هزینه واقعی گره n تا هدف است

- نتیجه: هزینه کل ($f(n)$) یک راه‌حل بیش از اندازه برآورد نمی‌شود
- برآوردهای خوشبینانه (optimistic): یک حد پایین برای هزینه مسیر
- شرط لازم و کافی برای بهینگی نسخه مبتنی بر جستجوی درخت
- مثال: تابع اکتشافی فاصله خط مستقیم در مسیریابی

جستجوی A^*

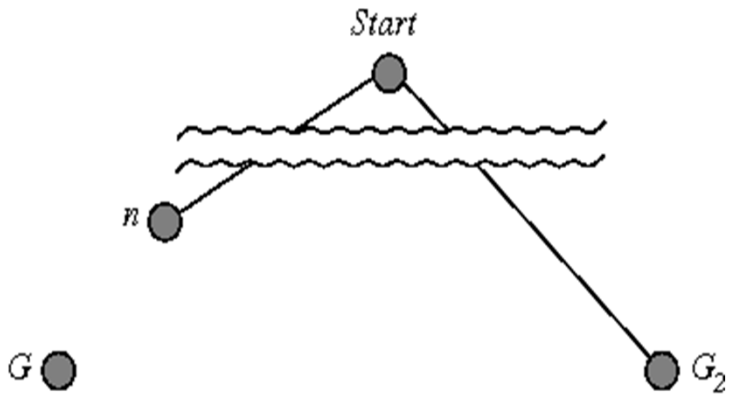
- شرایط بهینگی
  - تابع اکتشافی سازگار (consistent) باشد: $h(n) \leq c(n,a,n') + h(n')$
    - نامساوی مثلث (قضیه حمار)



- شرط بهینگی برای نسخه مبتنی بر جستجوی گراف
- هر تابع اکتشافی سازگار، پذیرفتنی هم هست (سازگاری شرط قوی‌تری است)
- اکثر توابع پذیرفتنی سازگار هستند
- مثال: تابع اکتشافی فاصله خط مستقیم در مسیریابی

جستجوی A^*

- اثبات بهینگی نسخه مبتنی بر جستجوی درخت با شرط پذیرفتنی بودن تابع اکتشافی



- G گره هدف بهینه است: $f(G)=g(G)$
- G_2 گره هدف غیربهینه است که در لیست مقدم قرار دارد: $f(G_2)=g(G_2)>g(G)$
- n یک گره بسط داده نشده در لیست مقدم است که در کوتاه‌ترین مسیر به گره هدف بهینه قرار دارد:

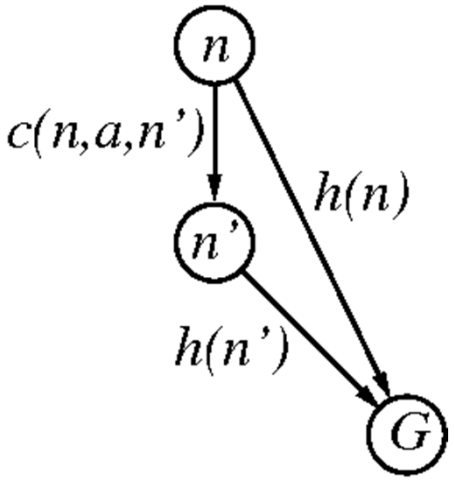
$$h(n) \le h^*(n) \Rightarrow f(n) \le f(G) \Rightarrow f(n) < f(G_2)$$

- الگوریتم هیچگاه گره G_2 را برای بسط دادن انتخاب نمی‌کند

جستجوی A^*

- اثبات بهینگی نسخه مبتنی بر جستجوی گراف با شرط سازگار بودن تابع اکتشافی
  - اثبات در دو مرحله

1. اگر تابع اکتشافی سازگار باشد، مقادیر تابع ارزیابی ((f(n)) در هر مسیری به سمت هدف غیر نزولی خواهد بود



- گره 'n پس‌آیند گره n باشد: $g(n') = g(n) + c(n,a,n')$
- $f(n') = g(n) + c(n,a,n') + h(n')$
- تابع اکتشافی سازگار است: $h(n) \leq c(n,a,n') + h(n')$
- $f(n)$ در هر مسیری غیرنزولی است: $f(n) \leq f(n')$

جستجوی A^*

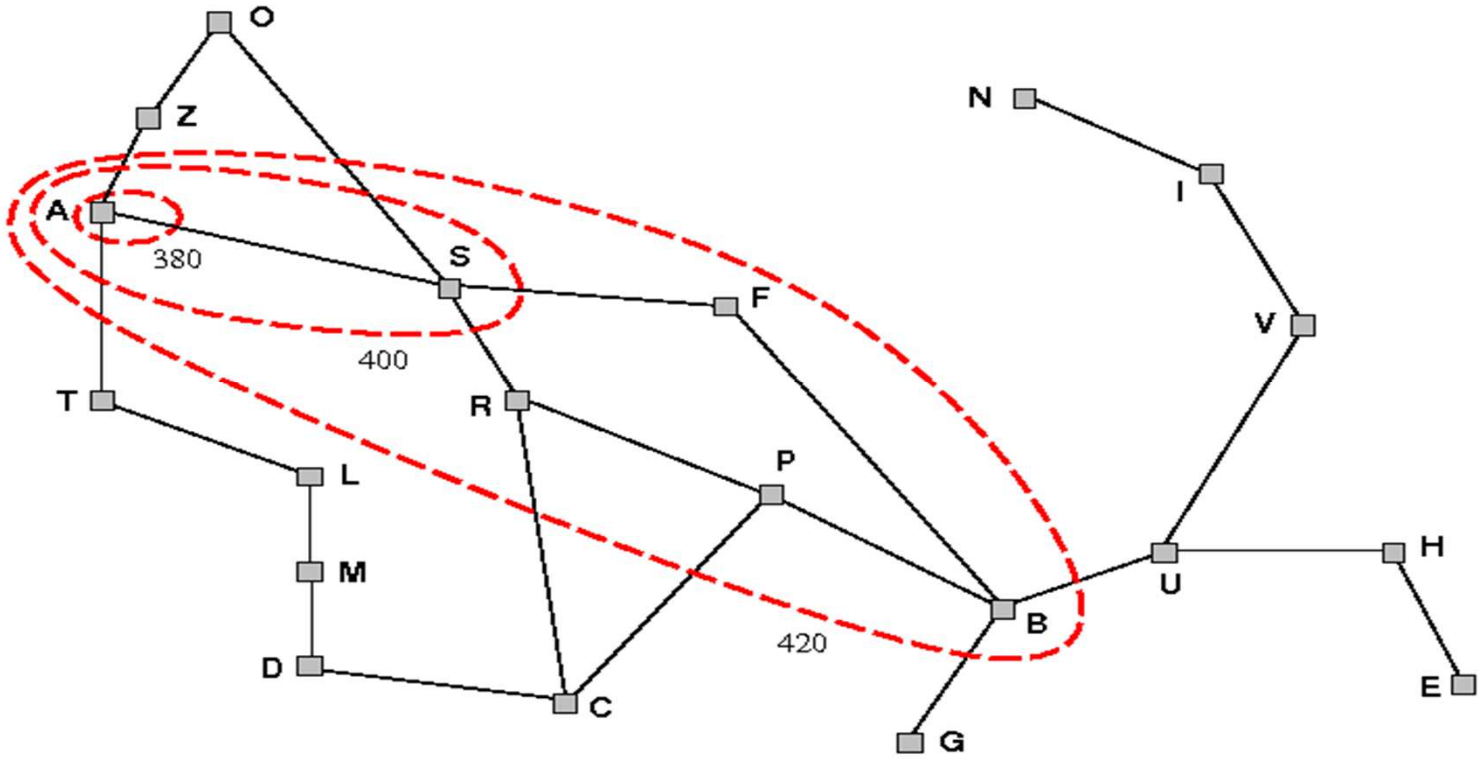
- اثبات بهینگی نسخه مبتنی بر جستجوی گراف با شرط سازگار بودن تابع اکتشافی (ادامه)

2. هرگاه گره‌ای (برای مثال n) برای بسط انتخاب شود، مسیر بهینه تا آن گره پیدا شده است
   - اگر چنین نباشد (برهان خلف) ← بر اساس ویژگی تفکیک فضای حالت در الگوریتم جستجوی گراف، باید گره n' در لیست مقدم وجود داشته باشد که در مسیر بهینه از ریشه به گره n قرار دارد
   - تابع ارزیابی در هر مسیری غیر نزولی است: $f(n') \leq f(n)$ ← گره n' قبل از گره n برای بسط انتخاب می‌شد

- پس گره‌ها به ترتیب غیر نزولی تابع $f(n)$ بسط داده می‌شوند
  - چون برای گره‌های هدف داریم $h(n)=0$، با بسط اولین گره هدف راه‌حل بهینه حاصل می‌شود

جستجوی A^*

- نحوه جستجوی نسخه مبتنی بر گراف، منحنی‌های تراز (contour) در فضای حالت تشکیل می‌دهد
  - در هر منحنی مقادیر کوچکتر یا مساوی f(n)



جستجوی A^*

- اگر C^* هزینه مسیر بهینه باشد
  - الگوریتم تمام گره‌ها با شرط $f(n) < C^*$ را بسط می‌دهد
  - الگوریتم برخی از گره‌ها با شرط $f(n) = C^*$ را بسط می‌دهد
    - گره‌های هم‌تراز گره‌ی هدف
  - الگوریتم هیچ گره‌ای با شرط $f(n) > C^*$ را بسط نمی‌دهد
    - شاخه مربوطه هرس (prune) می‌شود: چون تابع اکتشافی پذیرفتنی است
  - برای هر تابع اکتشافی سازگار، الگوریتم جستجوی A^* دارای کارایی بهینه (optimal efficiency) است
    - هیچ االگوریتم بهینه دیگر نمی‌تواند بطور قطع با بسط گره‌های کمتر (پیچیدگی زمانی کمتر) راه‌حل بهینه را پیدا کند

جستجوی A^*

- کامل است اگر
  - b متناهی باشد
  - هزینه تمام گام‌ها یک مقدار مثبت باشد
  - نتیجه: تعداد گره‌ها با شرط $f(n) \leq C^*$ متناهی باشد
- بهینه است اگر تابع اکتشافی پذیرفتنی (سازگار) باشد
- پیچیدگی زمانی برای مسأله‌ای با هزینه قدم‌های ثابت و فقط یک حالت هدف: $O(b^{\varepsilon d})$
  - ε نشان دهنده خطای نسبی تابع اکتشافی است: $\epsilon = \frac{h^* - h}{h^*}$
- دارای پیچیدگی فضایی نمایی است و تمام گره‌های تولید شده را در حافظه نگهداری می‌کند
  - مشکل اصلی الگوریتم ← نامناسب برای مسائل بزرگ

نسخه‌های حافظه محدود جستجوی A^*

- جستجوی A^* با افزایش مکرر عمق (iterative deepening A^* - IDA^*)
  - در هر مرحله حد هزینه کل ($f(n)$) افزایش می‌یابد
  - مقدار افزایش حد توسط گره با کمترین هزینه‌ی کل بیشتر از حد فعلی تعیین می‌شود
  - مناسب برای مسائل با هزینه گام یک
- جستجوی بهترین اول بازگشتی (recursive best-first)
  - نگهداری بهترین هزینه مسیر جایگزین از اجداد گره‌ی فعلی (متغیر f_{limit})
  - بهنگام‌سازی هزینه والدین با بهترین هزینه فرزندان هنگام برگشت به بالا

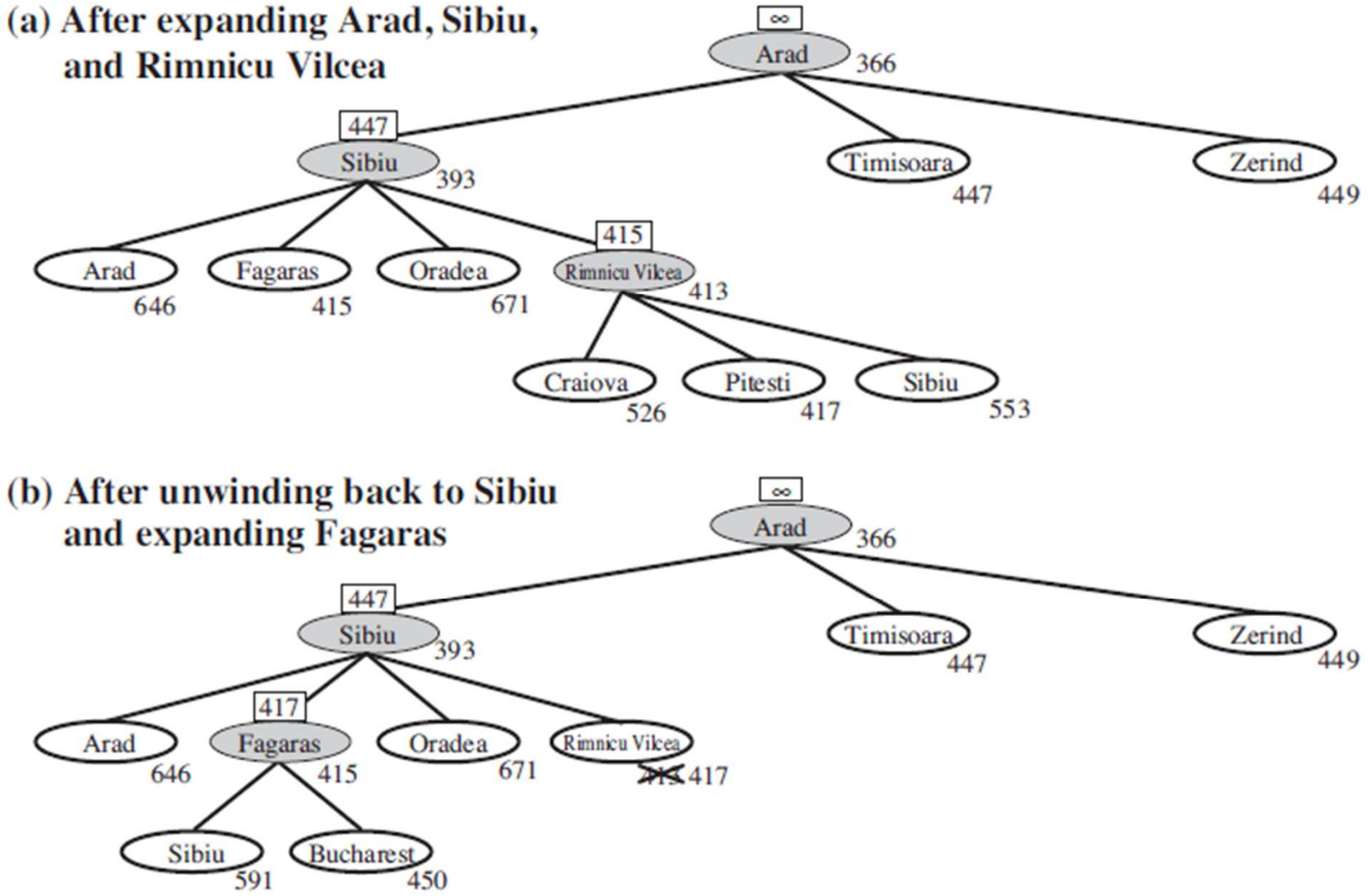
نسخه‌های حافظه محدود جستجوی A^*

- جستجوی بهترین اول بازگشتی

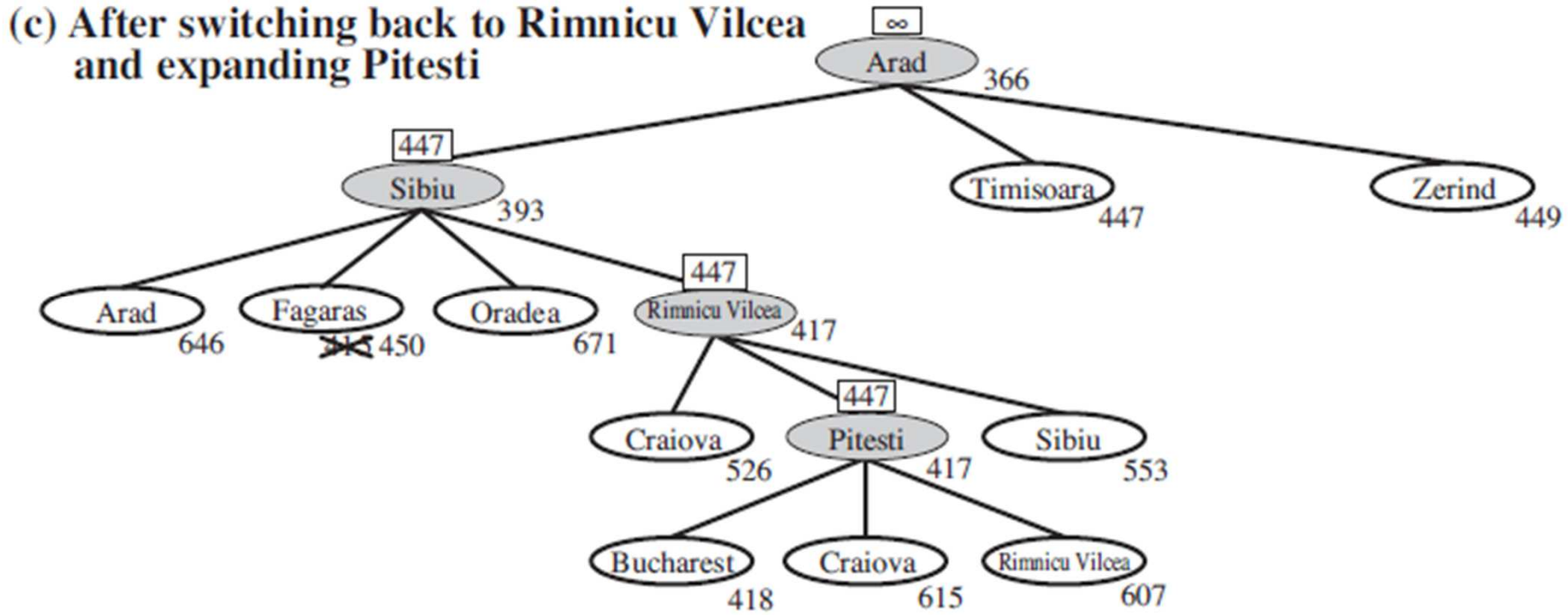
```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
    return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE), ∞)

function RBFS(problem, node, f_limit) returns a solution, or failure and a new f-cost limit
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors ← [ ]
  for each action in problem.ACTIONS(node.STATE) do
      add CHILD-NODE(problem, node, action) into successors
  if successors is empty then return failure, ∞
  for each s in successors do /* update f with value from previous search, if any */
      s.f ← max(s.g + s.h, node.f))
  loop do
      best ← the lowest f-value node in successors
      if best.f > f_limit then return failure, best.f
      alternative ← the second-lowest f-value among successors
      result, best.f ← RBFS(problem, best, min(f_limit, alternative))
      if result ≠ failure then return result
```

نسخه‌های حافظه محدود جستجوی A^*



نسخه‌های حافظه محدود جستجوی A^*



- الگوریتم نیاز به بازتولید بسیاری از گره‌ها دارد

نسخه‌های حافظه محدود جستجوی A^*

- جستجوی بهترین اول بازگشتی
  - کامل است
  - بهینه است اگر تابع اکتشافی بهینه باشد
  - پیچیدگی زمانی نمایی است » محاسبه دقیق به شرایط مختلف بستگی دارد
  - پیچیدگی فضایی نسبت به عمق عمیق‌ترین راه‌حل بهینه خطی است
- هر دو الگوریتم IDA^* و RBFS از حافظه کمی استفاده می‌کنند
  - عملکرد خود را فراموش می‌کنند ← نیاز به بسط دوباره بسیاری از گره‌ها
  - نمی‌توانند از تمام حافظه موجود استفاده کنند

نسخه‌های حافظه محدود جستجوی A^*

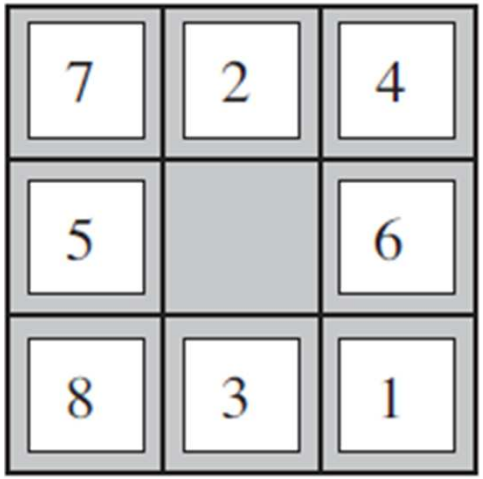
- جستجوی A^* حافظه محدود ساده شده (simplified memory-bounded A^* - SMA^*)
  - مانند نسخه اصلی الگوریتم A^* عمل می‌کند
  - با پر شدن حافظه برای اضافه کردن یک گره جدید به لیست مقدم، گره با بیشترین هزینه کل ($f(n)$) را حذف می‌کند و مقدار این هزینه را در پدر گره حذف شده نگهداری می‌کند
  - کامل و بهینه است اگر راه‌حل (بهینه) قابل دستیابی باشد: d کمتر از محدودیت حافظه باشد
  - برای بعضی مسائل جایگزین مناسب و قابل اتکا (robust) است
  - نامناسب برای مسائل خیلی پیچیده
  - از نظر زمان محاسبات، با محدودیت‌های حافظه‌ای یک مسأله می‌تواند رام‌نشدنی شود

یادگیری برای جستجوی بهتر

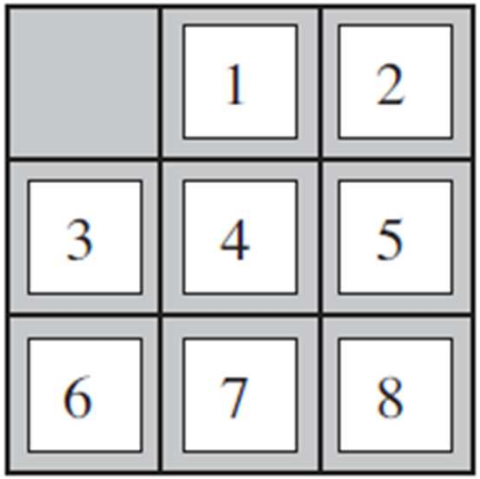
- فضای حالت فراسطحی (metalevel) و فضای حالت سطح اشیا (object-level)
  - هر حالت در فضای حالت فراسطحی نشان‌دهنده‌ی حالت درونی برنامه‌ای است که در فضای حالت سطح اشیا در حال جستجو است
    - درخت جستجوی فعلی
  - هر کنش یک قدم محاسباتی است که حالت درونی را تغییر می‌دهد
    - بسط یک گره و اضافه کردن فرزندان به درخت
  - مسیر در فضای حالت فراسطحی
- یادگیری فراسطحی از تجربه‌ها برای پیشگیری از کاوش شاخه‌های غیرامیدبخش
  - هدف یادگیری کاهش هزینه کل عامل است: هزینه جستجو + هزینه مسیر

انتخاب توابع اکتشافی

- مسأله معمای ۸



Start State



Goal State

- بطور متوسط، $b=3$ و راه‌حل در ۲۲ قدم قابل دستیابی است
- الگوریتم جستجوی درخت تقریباً $3^{22} \approx 3.1 \times 10^{10}$ حالت را بسط می‌دهد
- الگوریتم جسجوی گراف $9!/2 = 181440$ حالت را بسط می‌دهد
  - برای معمای ۱۵ این مقدار تقریباً 10^{15} است

انتخاب توابع اکتشافی

- توابع اکتشافی برای مسأله معمای ۸
  - تعداد مربع‌هایی که در جای خود قرار ندارند = h_1
  - مجموع فواصل (فاصله‌ی بلاکی) مربع‌ها از جای خود = h_2
- هر دو توابع پذیرفتنی هستند
- ارزیابی کیفیت عملکرد الگوریتم‌ها
  - ضریب انشعاب مؤثر (b^*): ضریب انشعاب یک درخت یکنواخت با عمق d و $N+1$ گره
    - N تعداد گره‌های تولید شده توسط الگوریتم جستجوی A^*

$$N = \sum_{i=1}^{d} (b^*)^i$$

    - برای الگوریتم با یک تابع اکتشافی خوب ضریب انشعاب مؤثر نزدیک به یک است

انتخاب توابع اکتشافی

- مقایسه‌ی الگوریتم‌ها روی مسأله‌ی معمای ۸
- مقادیر، میانگین روی ۱۰۰ نمونه مسأله را نشان می‌دهند

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

انتخاب توابع اکتشافی

- مفهوم چیرگی (dominance): اگر برای دو تابع اکتشافی پذیرفتنی، داشته باشیم:

$$h_1(n) \geq h_2(n)$$

آنگاه تابع h_1 بر تابع h_2 چیرگی دارد

  - در مسأله معمای ۸ تابع h_2 بر تابع h_1 چیرگی دارد
- توابع اکتشافی با چیرگی بیشتر بهتر هستند
  - در حالت کلی توابع اکتشافی با مقادیر بالاتر بهتر هستند
- چندین تابع اکتشافی که هیچیک بر دیگری چیرگی ندارد ⟵ بیشینه تمام توابع

$$h(n) = \max\{h_1(n), \cdots, h_m(n)\}$$

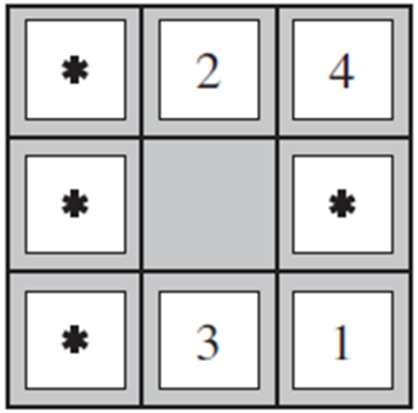
  - اگر تمام h_iها پذیرفتنی یا سازگار باشند، پذیرفتنی و سازگار خواهد بود
  - برتمام h_iها چیرگی دارد

تولید توابع اکتشافی پذیرفتنی

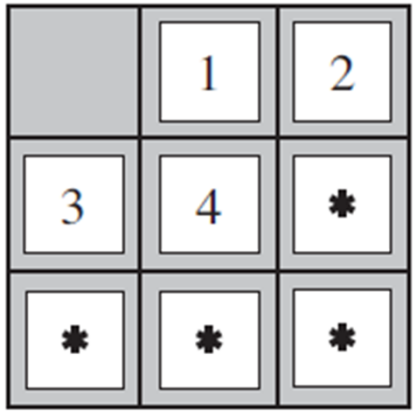
- استفاده از مسائل آسوده شده (relaxed)
  - مسائلی با محدودیت‌های کمتر بر روی کنش‌ها نسبت به مسأله‌ی اصلی
  - فضای حالت این مسأله‌ها، فضای حالت مسأله‌ی اصلی را در بر می‌گیرد
  - هر راه‌حل بهینه‌ای در مسأله‌ی اصلی یک راه‌حل در مسأله‌ی آسوده شده است ولی ممکن است دیگر در مسأله آسوده بهینه نباشد
  - **هزینه راه‌حل بهینه برای مسأله‌ی آسوده شده یک تابع اکتشافی پذیرفتنی و سازگار برای مسأله‌ی اصلی است**
  - آسوده سازی خودکار مسائل با استفاده از یک زبان رسمی برای تعریف مسائل
    - برنامه ABSOLVER
    - مسائل آسوده شده باید بدون جستجو حل شوند: محاسبه مقادیر تابع اکتشافی نباید هزینه‌بر باشد

تولید توابع اکتشافی پذیرفتنی

- استفاده از زیرمسأله‌ها



Start State

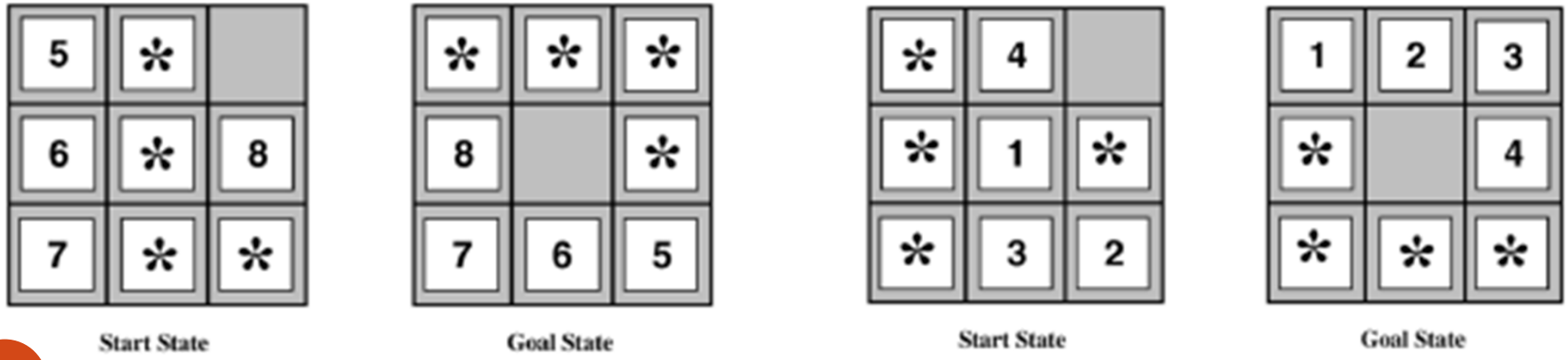


Goal State

- در نظر گرفتن هزینه راه‌حل زیرمسأله‌ی یک مسأله
- نگهداری هزینه‌ی دقیق راه‌حل‌ها برای تمام نمونه‌های ممکن یک زیرمسأله در پایگاه داده‌ی الگو (pattern database)
- مقدار تابع اکتشافی در هنگام جستجوی اصلی با پیدا کردن نمونه مسأله متناظر با آن در پایگاه داده‌ی الگو
  - جستجوی به عقب با شروع از حالت هدف برای محاسبه‌ی هزینه‌ها

تولید توابع اکتشافی پذیرفتنی

- استفاده از زیرمسأله‌ها
  - تلفیق توابع اکتشافی بدست آمده از پایگاه‌های داده‌ی مختلف با کمک بیشینه‌گیری
  - توابع اکتشافی جمع‌پذیر (additive): امکان جمع کردن توابع اکتشافی در صورت استفاده از پایگاه‌های داده‌ی الگوی مجزا (disjoint)
    - مسأله باید تجزیه‌پذیر باشد



تولید توابع اکتشافی پذیرفتنی

- یادگیری توابع اکتشافی از روی تجربه
  - از دید عامل حل مسأله، چگونه می‌توان توابع اکتشافی را ساخت؟
    - یادگیری از روی تجربه: حل تعداد زیادی از نمونه‌های مسأله
  - هر راه‌حل بهینه برای یک نمونه مسأله، چند مثال برای یادگیری تابع اکتشافی می‌دهد
    - برای پیش‌بینی هزینه‌ی گره‌ها در دیگر نمونه مسأله‌ها
  - روش‌های یادگیری استنتاجی (inductive):
    - هنگامی که حالات با یکسری ویژگی‌ها مشخص شده باشند (نمایش فاکتوربندی شده)
    - برآورد مقدار تابع اکتشافی بر اساس ترکیبی از مقادیر ویژگی‌ها

جمع‌بندی

- عامل حل مسأله ← جستجو
- مؤلفه‌های تعریف مسأله
- مسائل بازیچه و مسائل واقعی
- الگوریتم‌های مبنایی جستجو
  - معیارهای ارزیابی الگوریتم‌های جستجو
- الگوریتم‌های جستجوی کورکورانه
- الگوریتم‌های جستجوی آگاهانه
  - توابع اکتشافی