هوش مصنوعی

فصل سوم

جست و جوی آگاهانه و اکتشاف

جست و جوی آگاهانه و اکتشاف

- روش های هوشمند برای این طراحی شده اند تا به جای جستجوی تمامی گره ها، با هدف و جهت معین بخشی از گره ها را جستجو کرده و پاسخ را بین آنها بیابید. برای حصول این هدف دو کار باید صورت گیرد:
  - تابعی معین کند که گره جاری، گره مناسبی برای رسیدن به جواب هست یا خیر (تابع کشف کننده)
  - استراتژی جستجویی که قادر به استفاده از این تابع کشف کننده باشد.

جست و جوی آگاهانه و اکتشاف

متدهای جستجوی آگاهانه

- بهترین جستجو
  - حریصانه
  - A*
  - IDA*
  - RBFS
  - MA* و SMA*
- جستجوی محلی و بهینه سازی
  - تپه نوردی
  - شبیه سازی حرارت
  - پرتو محلی
  - الگوریتمهای ژنتیک

جست و جوی آگاهانه و اکتشاف

تعاریف

- تابع هزینه مسیر، g(n) : هزینه مسیر از گره اولیه تا گره n
- تابع اکتشافی، h(n) : هزینه تخمینی ارزان ترین مسیر از گره n به گره هدف
- تابع ارزیابی، f(n) : هزینه تخمینی ارزان ترین مسیر از طریق n

f(n): g(n) + h(n)

اگر n گره هدف باشد در نتیجه h(n) = ۰ خواهد بود

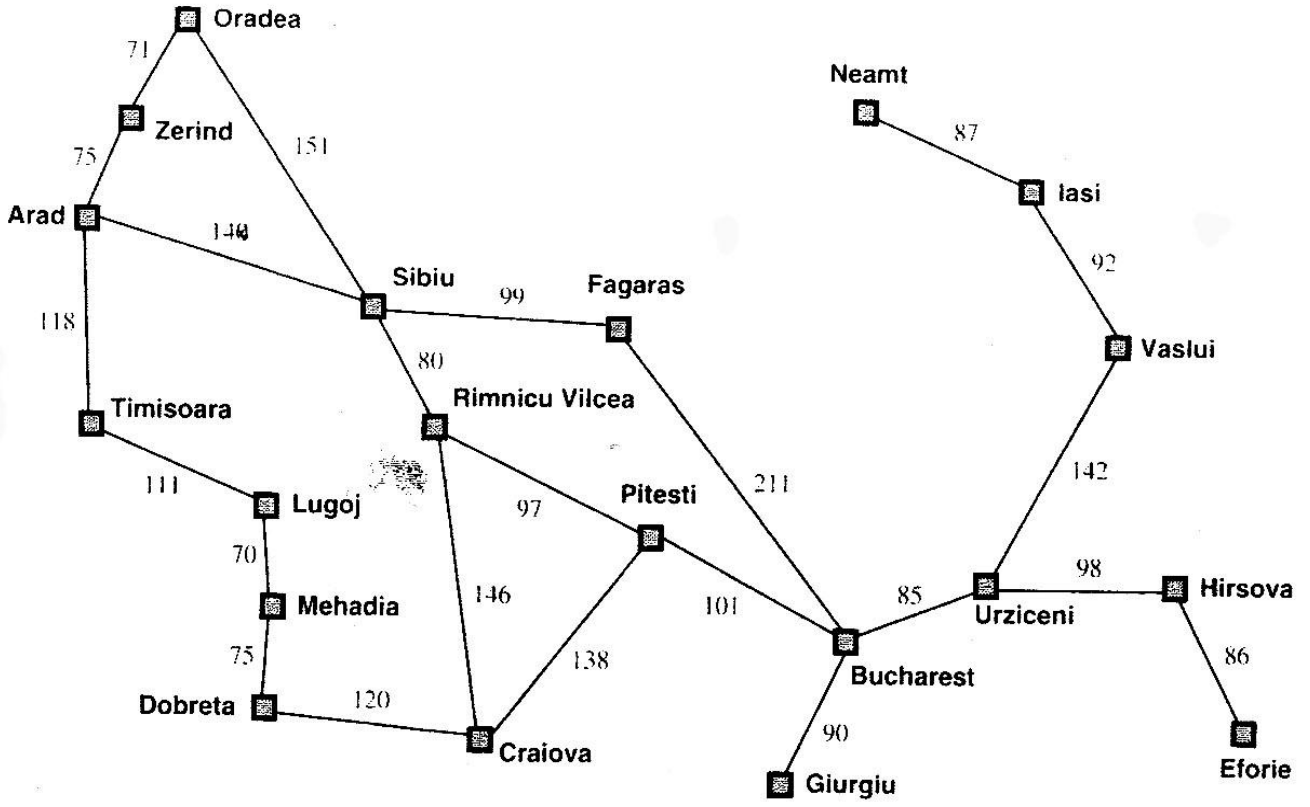
جستجوی اول- بهترین حریصانه

- جستجوی اول- بهترین حریصانه تلاش می کند گرهی را که نزدیک ترین به هدف است توسعه دهد، یعنی مشابه آنکه به سرعت به دنبال جواب برویم.
- بنابراین گره ها را تنها بر مبنای تابع اکتشافی ارزیابی می کند:

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

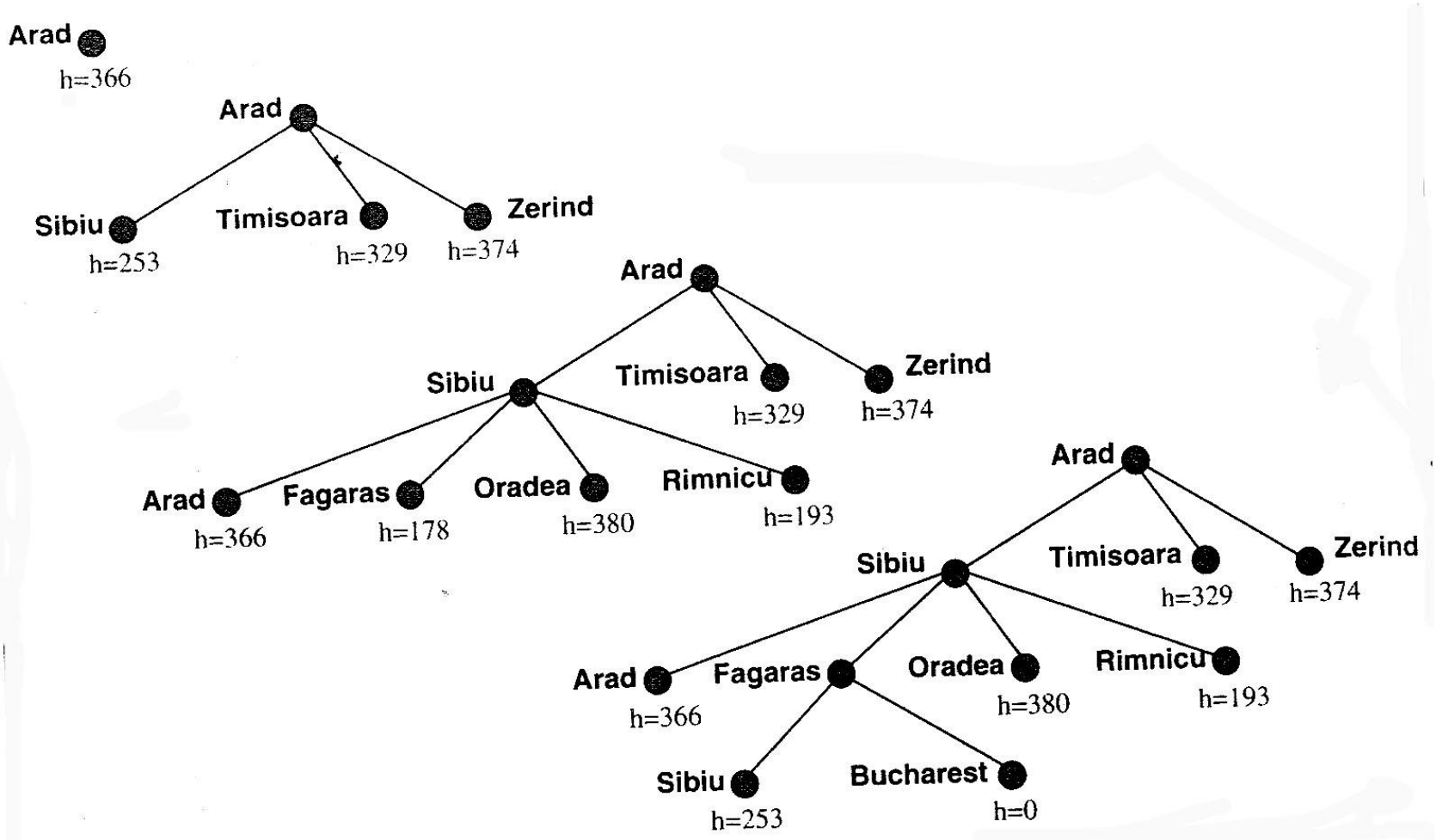
جستجوی اول- بهترین حریصانه

- برای یافتن مسیر در نقشهٔ رومانی ازتابع اکتشافی خط مستقیم استفاده کرده ایم.



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

جستجوی اول- بهترین حریصانه



جست و جوی آگاهانه و اکتشاف

جستجوی حریصانه

- **کامل بودن: خیر** **بهینگی: خیر**

زیرا ممکن است در مسیر بي نهايتي بيفتد و هرگز ديگر گزينه ها را انتخاب نکند

- **پیچیدگی زمانی:** $O(b^m)$
- **پیچیدگی فضا:** $O(b^m)$

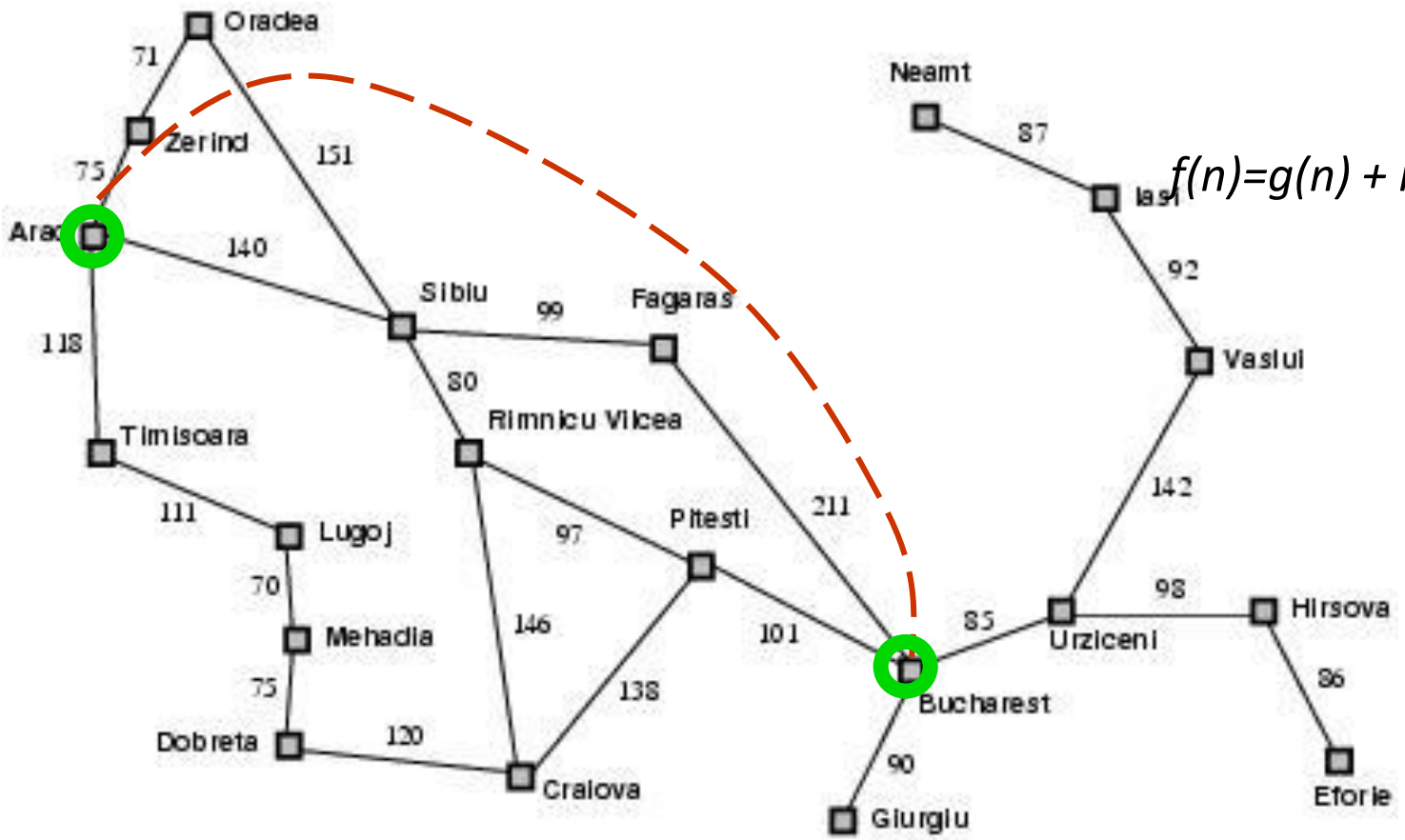
جستجوی A* حداقل سازی هزینهٔ پاسخ تخمین زده شده

- **A*** گره ها را با ترکیب:
  - g(n) هزینهٔ رسیدن به گره n
  - h(n)هزینهٔ رفتن به هدف ارزیابي مي كند:

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

  - f(n) تخمین هزینهٔ کوتاه ترین مسیر پاسخ با فرض عبور از n
- تابع اكتشافي قابل قبول (Admissible heuristic function) : توابعي هستند كه هزينه حل مسئله را از مقدار واقعي آن كمتر در نظر مي گيرند .

مثال از جستجوی A*



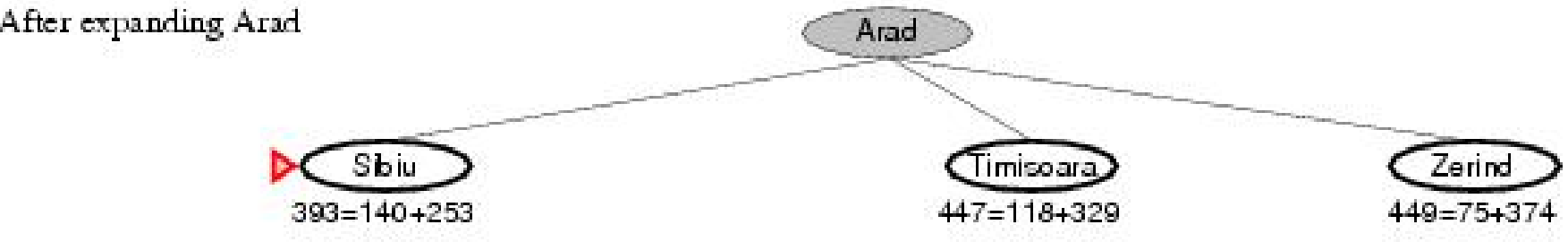
جستجوی A* در نقشه رومانی

(a) The initial state

جستجوی Bucharest با شروع از Arad

f(Arad) = g(Arad)+h(Arad)=0+366=366

جستجوی A* در نقشه رومانی



Arad را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

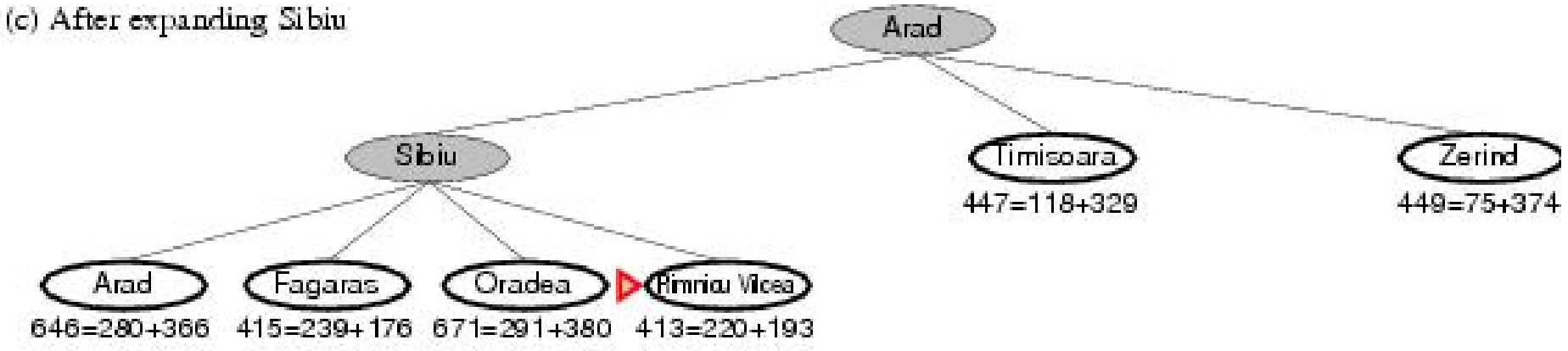
f(Sibiu)=c(Arad,Sibiu)+h(Sibiu)=140+253=393

f(Timisoara)=c(Arad,Timisoara)+h(Timisoara)=118+329=447

f(Zerind)=c(Arad,Zerind)+h(Zerind)=75+374=449

بهترین انتخاب شهر Sibiu است

جستجوی A* در نقشه رومانی



Sibiu را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

f(Arad)=c(Sibiu,Arad)+h(Arad)=280+366=646

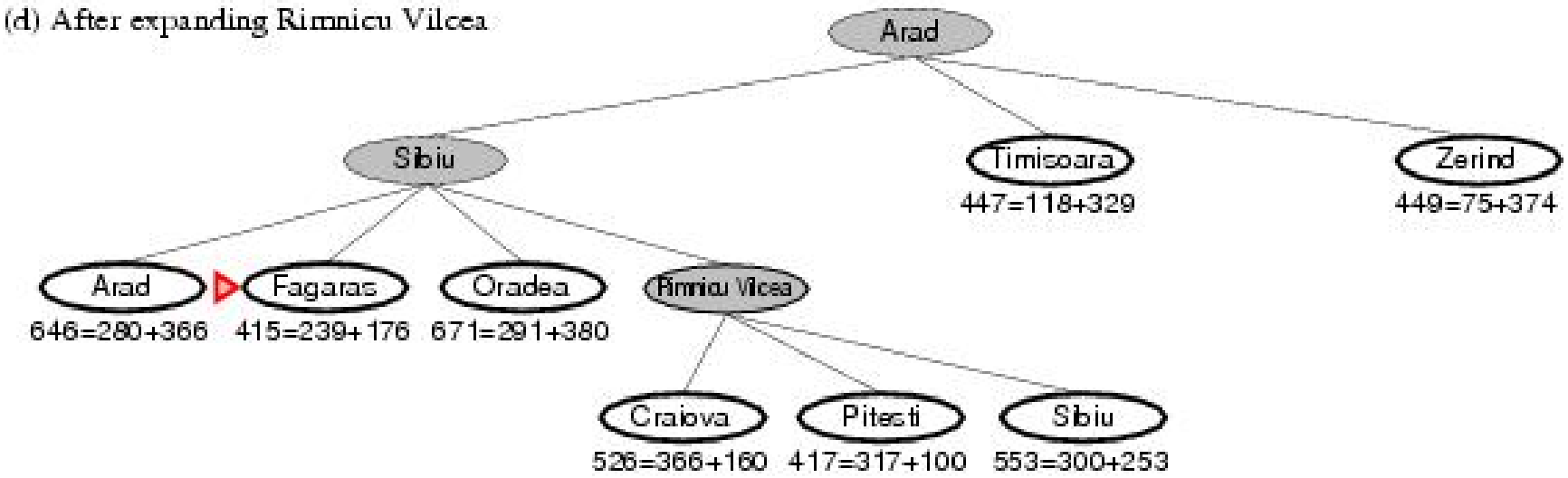
f(Fagaras)=c(Sibiu,Fagaras)+h(Fagaras)=239+179=415

f(Oradea)=c(Sibiu,Oradea)+h(Oradea)=291+380=671

f(Rimnicu Vilcea)=c(Sibiu,Rimnicu Vilcea)+ h(Rimnicu Vilcea)=220+192=413

بهترین انتخاب شهر Rimnicu Vilcea است

جستجوی A* در نقشه رومانی



Rimnicu Vilcea را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

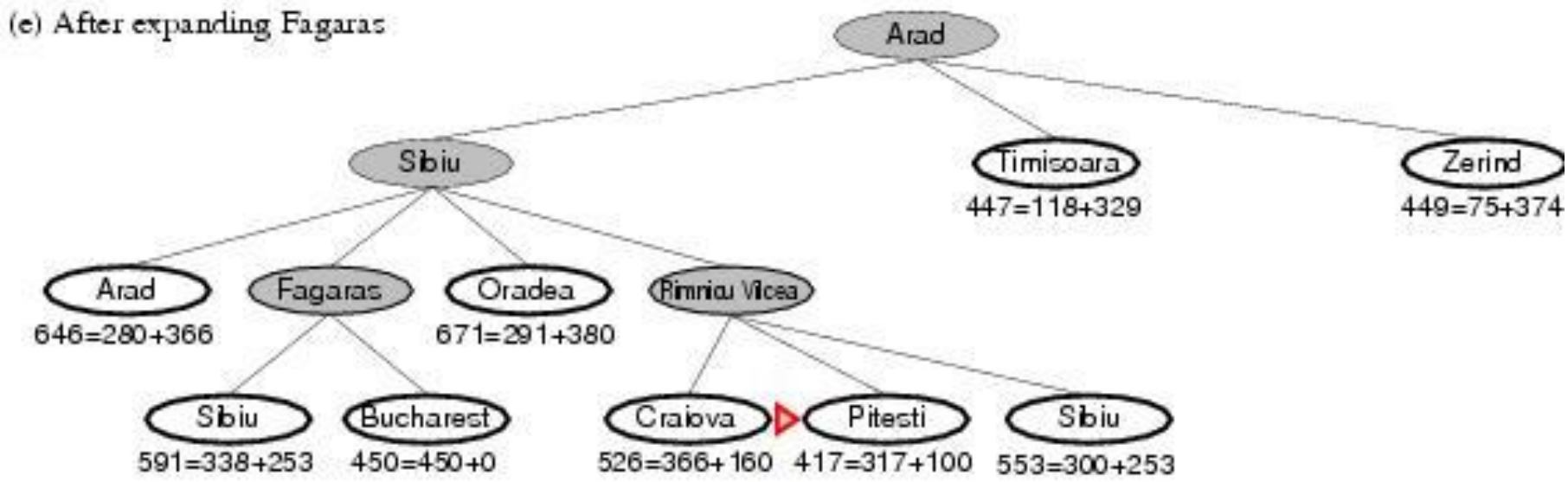
f(Craiova)=c(Rimnicu Vilcea, Craiova)+h(Craiova)=360+160=526

f(Pitesti)=c(Rimnicu Vilcea, Pitesti)+h(Pitesti)=317+100=417

f(Sibiu)=c(Rimnicu Vilcea,Sibiu)+h(Sibiu)=300+253=553

بهترین انتخاب شهر Fagaras است

جستجوی A* در نقشه رومانی



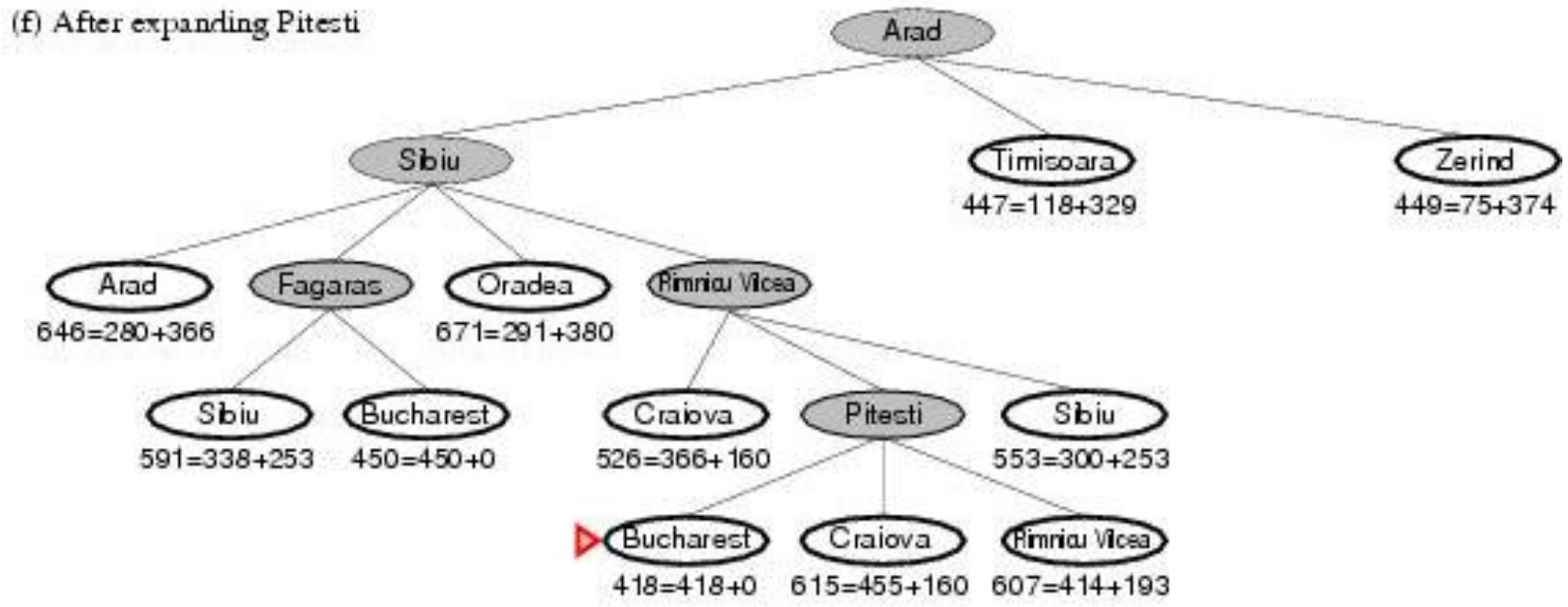
Fagaras را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

f(Sibiu)=c(Fagaras, Sibiu)+h(Sibiu)=338+253=591

f(Bucharest)=c(Fagaras,Bucharest)+h(Bucharest)=450+0=450

بهترین انتخاب شهر !!! Pitesti است

جستجوی A* در نقشه رومانی



Pitesti را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

f(Bucharest)=c(Pitesti,Bucharest)+h(Bucharest)=418+0=418

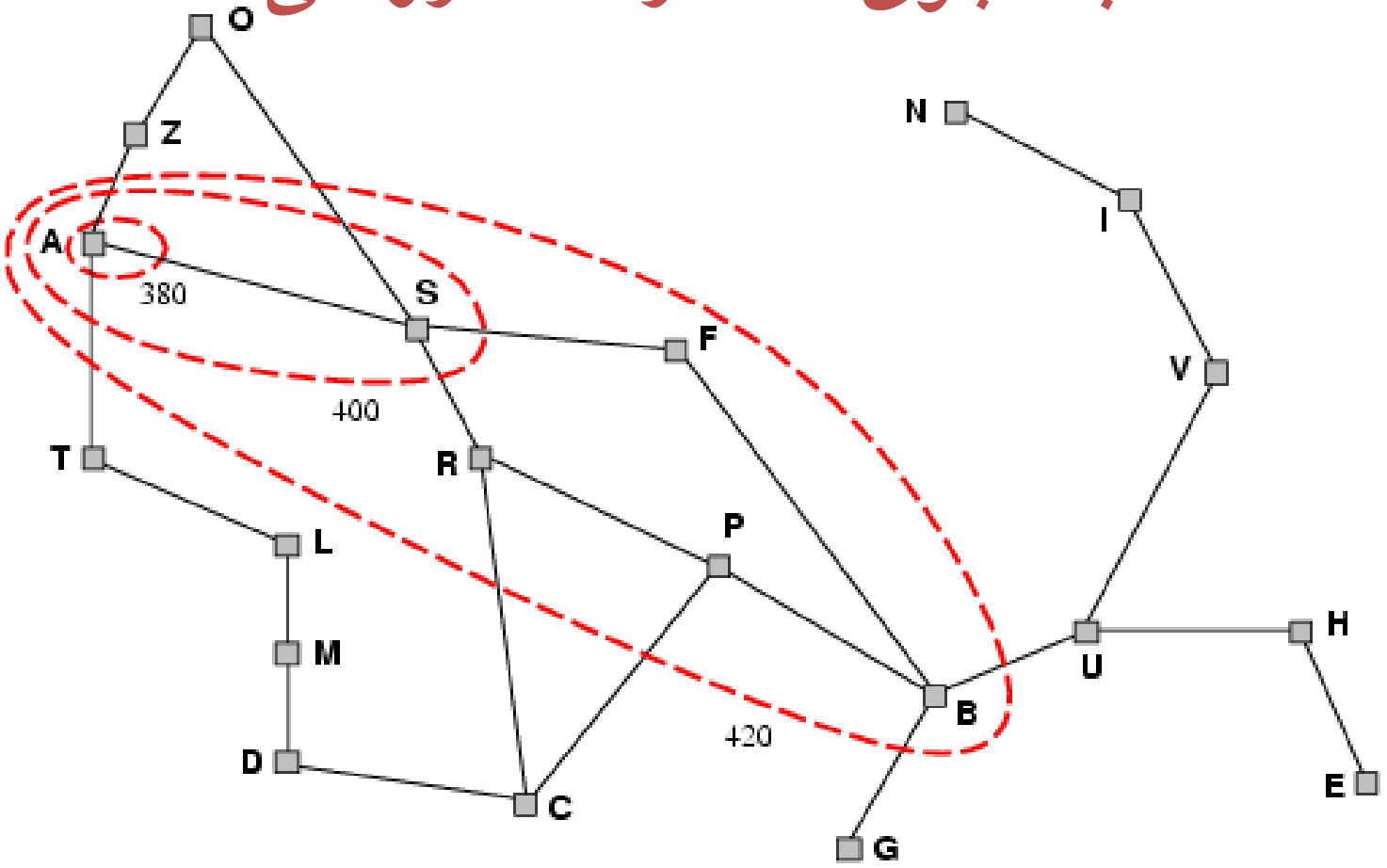
بهترین انتخاب شهر !!! Bucharest است

جستجوی A*

- کامل بودن: بله
- بهینگی: بله
- پیچیدگی زمانی: $O(b^m)$
- پیچیدگی فضا: $O(b^m)$

جست و جوی آگاهانه و اکتشاف

جستجوی A* در نقشه رومانی



جستجوی اکتشافی با حافظه محدود IDA*

- ساده ترین راه برای کاهش حافظه مورد نیاز A* استفاده از عمیق کننده تکرار در زمینه جست و جوی اکتشافی است.
- الگوریتم عمیق کننده تکرار A* ← IDA*
- در جستجوی IDA* مقدار برش مورد استفاده، عمق نیست بلکه هزینه f-cost(g+h) است.
- IDA* برای اغلب مسئله های با هزینه های مرحله ای، مناسب است و از سربار ناشی از نگهداری صف مرتبی از گره ها اجتناب میکند

بهترین جستجوی بازگشتی RBFS

- ساختار آن شبیه جست و جوی عمقی بازگشتی است، اما به جای اینکه دائما به طرف پایین مسیر حرکت کند، مقدار f مربوط به بهترین مسیر از هر جد گره فعلی را نگهداری میکند، اگر گره فعلی از این حد تجاوز کند، بازگشتی به عقب برمیگردد تا مسیر دیگری را انتخاب کند.
- این جستجو اگر تابع اکتشافی قابل قبولی داشته باشد، بهینه است.
- پیچیدگی فضایی آن O(bd) است
- تعیین پیچیدگی زمانی آن به دقت تابع اکتشافی و میزان تغییر بهترین مسیر در اثر بسط گره ها بستگی دارد.

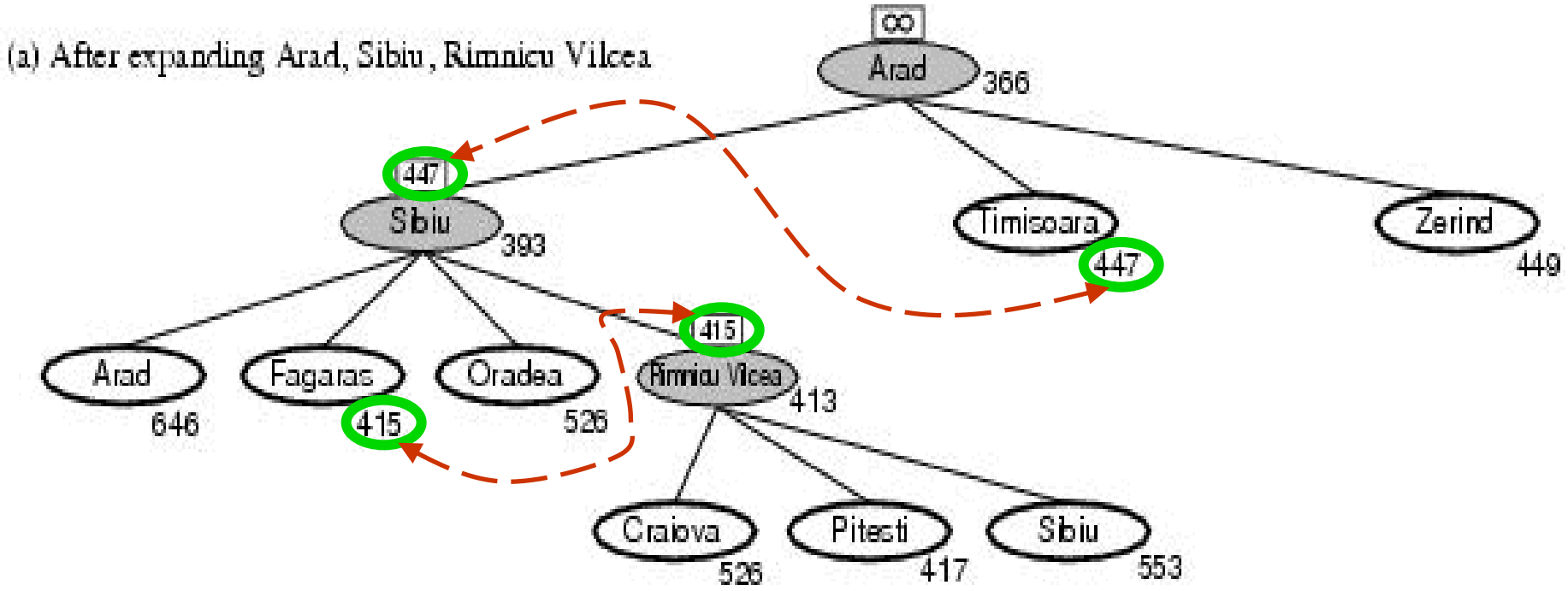
بهترین جستجوی بازگشتی RBFS

function RECURSIVE-BEST-FIRST-SEARCH(*problem*) **returns** a solution, or failure
    RBFS(*problem*, MAKE-NODE(INITIAL-STATE[*problem*]), ∞)

function RBFS(*problem*, *node*, *f_limit*) **returns** a solution, or failure and a new f-cost limit
    if GOAL-TEST[*problem*](STATE[*node*]) **then return** *node*
    successors $\leftarrow$ EXPAND(*node*, *problem*)
    if *successors* is empty **then return** *failure*, ∞
    for each s **in** *successors* **do**
        $f[s] \leftarrow \max(g(s) + h(s), f[node])$
    repeat
        best $\leftarrow$ the lowest f-value node in *successors*
        if $f[best] > f\_limit$ **then return** *failure*, $f[best]$
        alternative $\leftarrow$ the second-lowest f-value among *successors*
        result, $f[best] \leftarrow$ RBFS(*problem*, *best*, min(*f_limit*, *alternative*))
        if *result* $\neq$ *failure* **then return** *result*

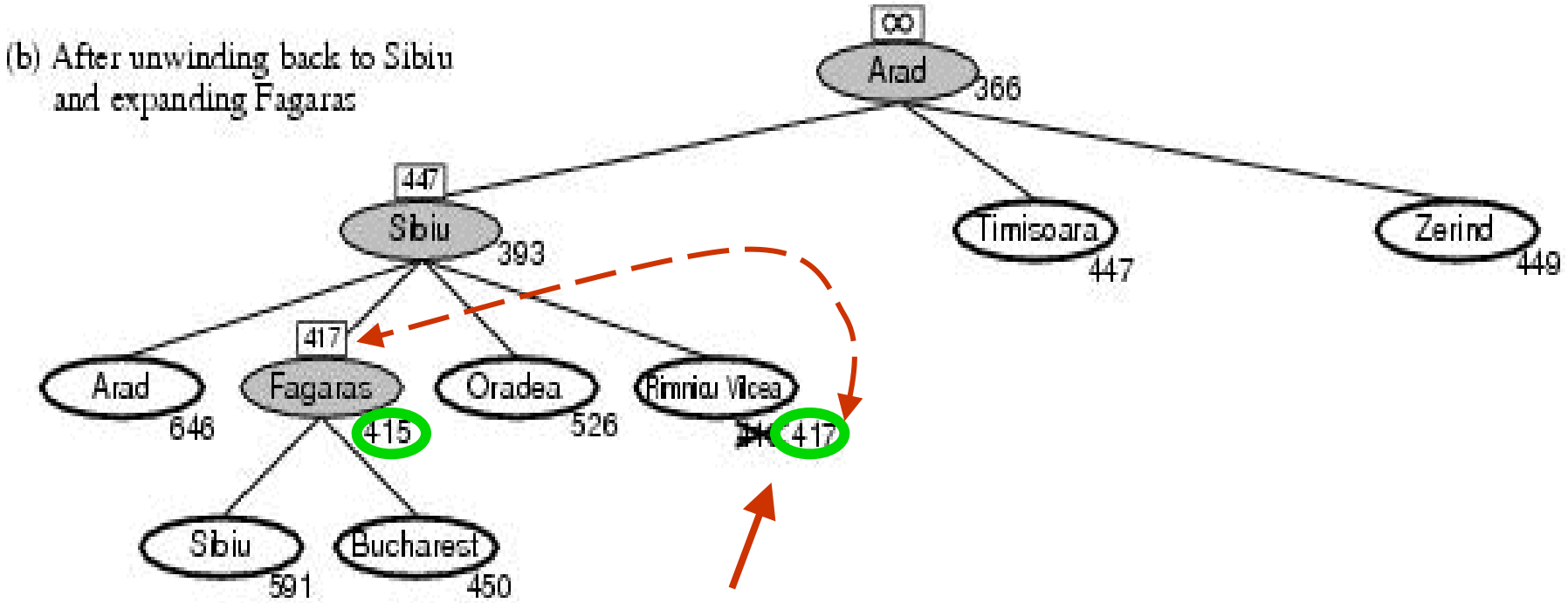
بهترین جستجوی بازگشتی RBFS

در نقشه رومانی



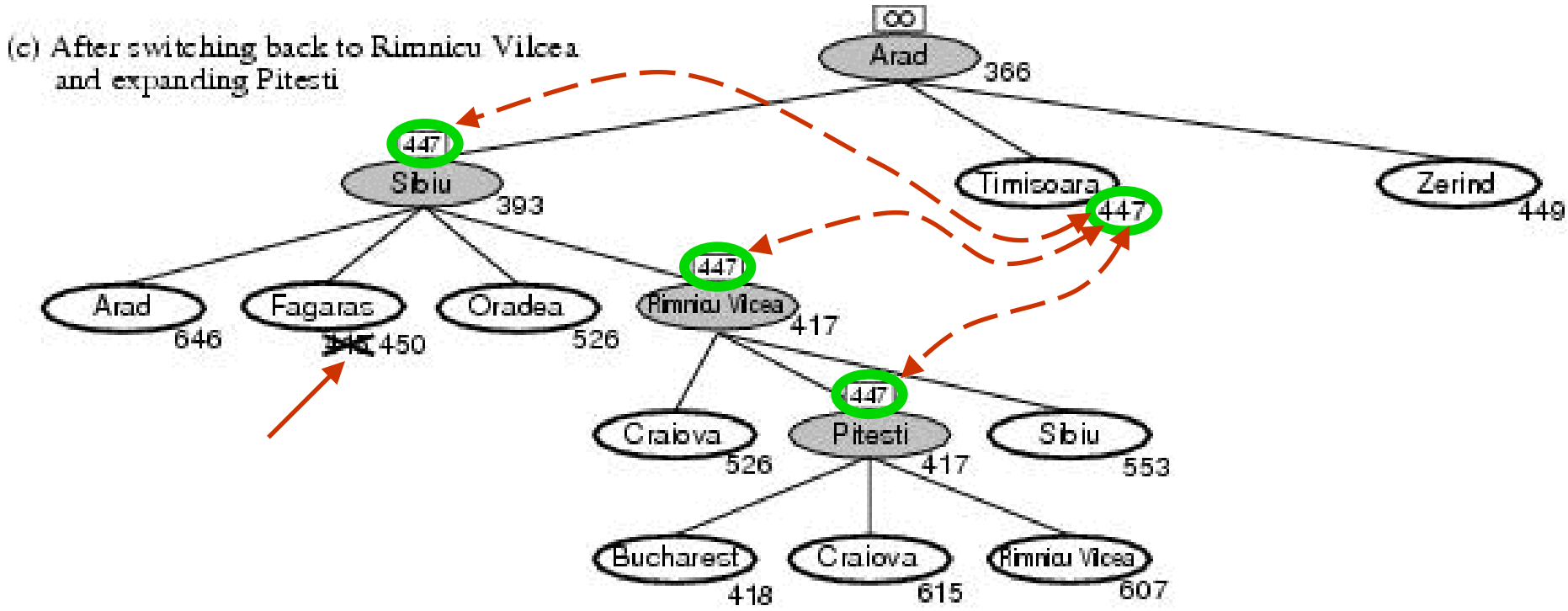
بهترین جستجوی بازگشتی RBFS

در نقشه رومانی



بهترین جستجوی بازگشتی RBFS

در نقشه رومانی



بهترین جستجوی بازگشتی RBFS

- RBFS تا حدی از IDA* کارآمدتر است، اما گره های زیادی تولید میکند.

- IDA* و RBFS در معرض افزایش توانی پیچیدگی قرار دارند که در جست و جوی گرافها مرسوم است، زیرا نمیتوانند حالتهای تکراری را در غیر از مسیر فعلی بررسی کنند. لذا، ممکن است یک حالت را چندین بار بررسی کنند.

- IDA* و RBFS از فضای اندکی استفاده میکنند که به آنها آسیب میرساند. IDA* بین هر تکرار فقط یک عدد را نگهداری میکند که مقدار فعلی هزینه f است. RBFS اطلاعات بیشتری در حافظه نگهداری میکند

جستجوی حافظه محدود ساده SMA*

- SMA* بهترین برگ را بسط میدهد تا حافظه پر شود. در این نقطه بدون از بین بردن گره های قبلی نمیتواند گره جدیدی اضافه کند

- SMA* همیشه بدترین گره برگ را حذف میکند و سپس از طریق گره فراموش شده به والد آن بر میگردد. پس جد زیر درخت فراموش شده، کیفیت بهترین مسیر را در آن زیر درخت میداند

- اگر عمق سطحی ترین گره هدف کمتر از حافظه باشد، کامل است.

- SMA* بهترین الگوریتم همه منظوره برای یافتن حلهای بهینه میباشد

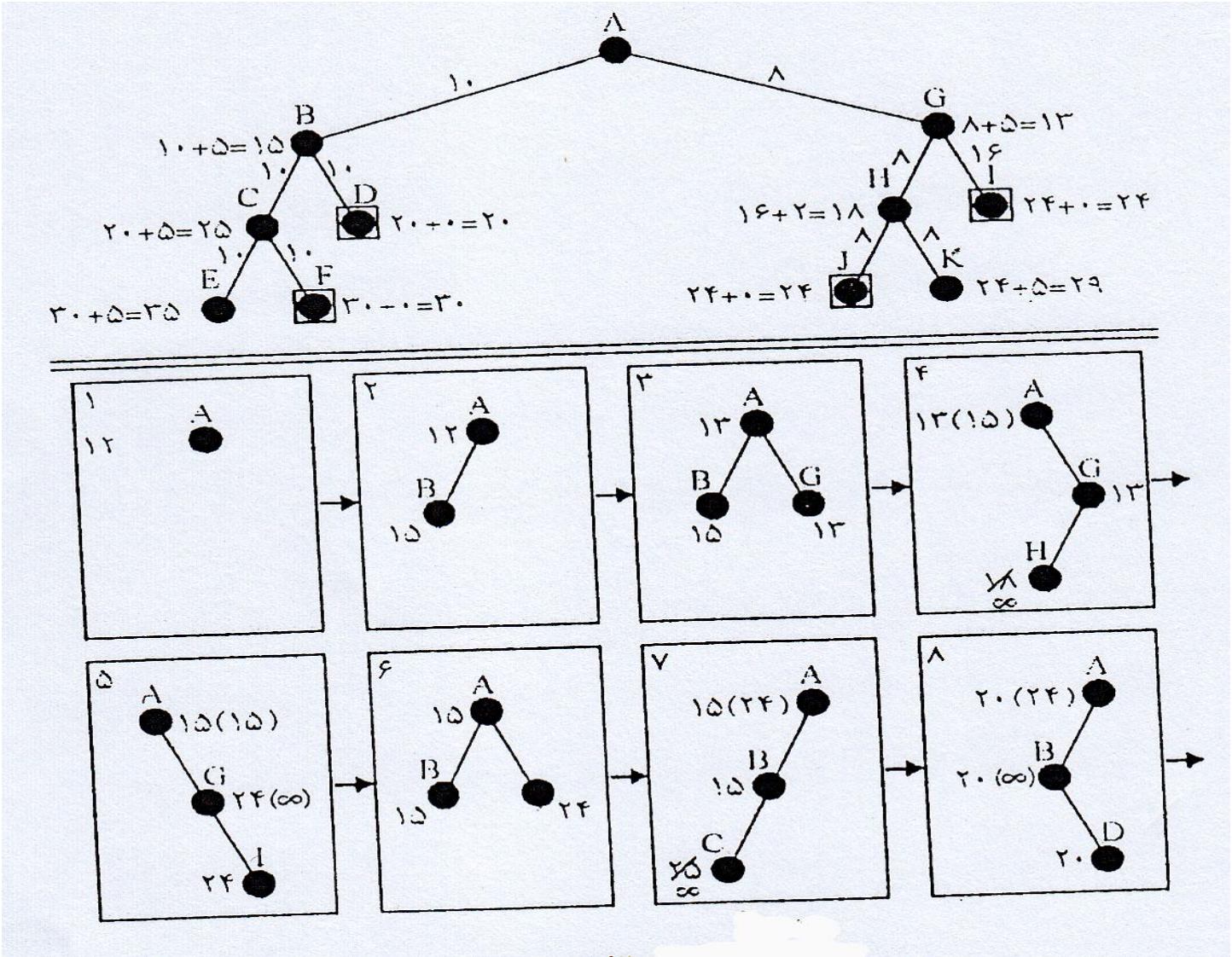
SMA*جستجوی حافظه محدود ساده

- اگر مقدار f تمام برگها یکسان باشد و الگوریتم یک گره را هم برای بسط و هم برای حذف انتخاب کند، SMA* این مسئله را با بسط بهترین برگ جدید و حذف بهترین برگ قدیمی حل میکند

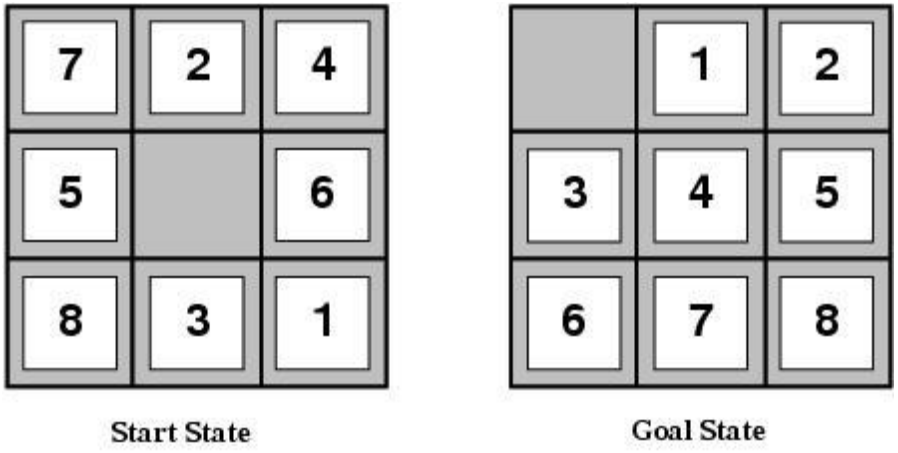
- ممکن است SMA* مجبور شود دائما بین مجموعه ای از مسیرهای حل کاندید تغییر موضع دهد، در حالی که بخش کوچکی از هر کدام در حافظه جا شود

- محدودیتهای حافظه ممکن است مسئله ها را از نظر زمان محاسباتی، غیر قابل حل کند.

جستجوی حافظه محدود ساده SMA*



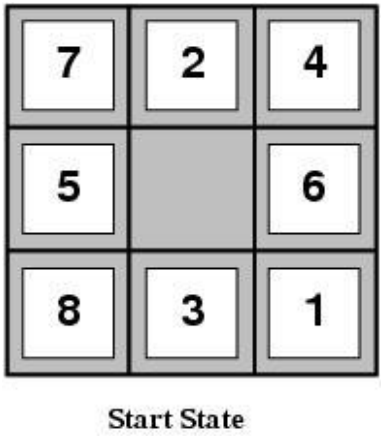
توابع اکتشافی



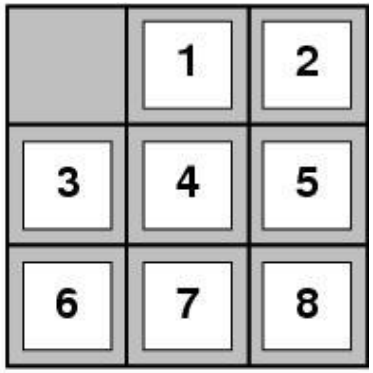
مثال برای معمای ۸

- میانگِین هزینه حل تقریبا ۲۲ مرحله و فاکتور انشعاب در حدود ۳ است.
- جست و جوی جامع تا عمق ۲۲ : $3^{22} \approx 3.1 \times 10^{10}$
- با انتخاب یک تابع اکتشافی مناسب میتوان مراحل جستجو را کاهش داد

دو روش اکتشافی متداول برای معمای ۸



Start State



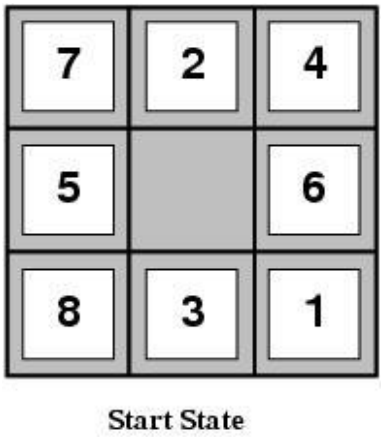
Goal State

$h_1 =$ تعداد کاشیها در مکانهای نادرست

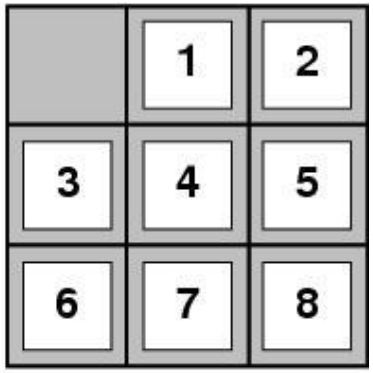
$h_1 = 8$ در حالت شروع

h_1 اکتشاف قابل قبولی است، زیرا هر کاشی که در جای نامناسبی قرار دارد، حداقل یکبار باید جابجا شود

دو روش اکتشافی متداول برای معمای ۸



Start State



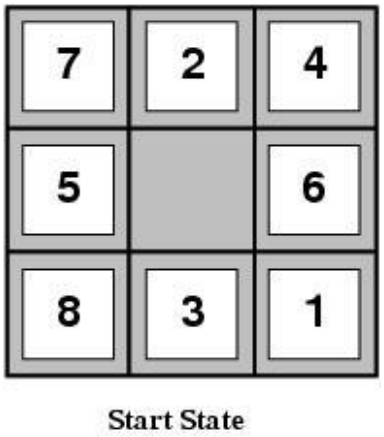
Goal State

h_2 = مجموعه فواصل کاشیها از موقعیتهای هدف آنها در حالت شروع

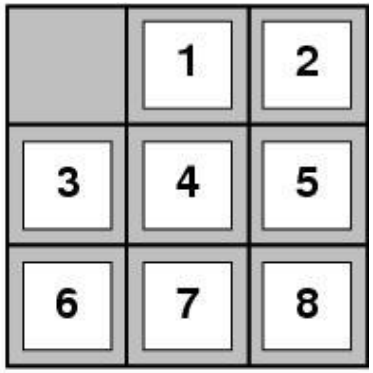
$$h_2 = 3+1+2+2+2+3+3+2 = 18$$

چون کاشیها نمیتوانند در امتداد قطر جا به جا شوند، فاصله ای که محاسبه میکنیم مجموع فواصل افقی و عمودی است. این فاصله را فاصله **بلوک شهر** یا فاصله **مانهاتان** مینامند.

دو روش اکتشافی متداول برای معمای ۸



Start State



Goal State

h_2 = مجموعه فواصل کاشیها از موقعیتهای هدف آنها

h_2 قابل قبول است، زیرا هر جابجایی که میتواند انجام گیرد، به اندازه یک مرحله به هدف نزدیک میشود.

هیچ کدام از این برآوردها، هزینه واقعی راه حل نیست

هزینه واقعی ۳۶ است

جست و جوی آگاهانه و اکتشاف

اثر دقت اکتشاف بر کارایی

- اگر برای هر گره n داشته باشیم: h2(n) >= h1(n)
  - h2 بر h1 غالب است
  - غالب بودن مستقیما به کارایی ترجمه میشود
  - تعداد گره هایی که با بکارگیری h2 بسط داده میشود، هرگز بیش از بکارگیری h1 نیست

همیشه بهتر است از تابع اکتشافی با مقادیر بزرگ استفاده کرد، به شرطی که زمان محاسبه اکتشاف، خیلی بزرگ نباشد

مسئله راحت (relax)

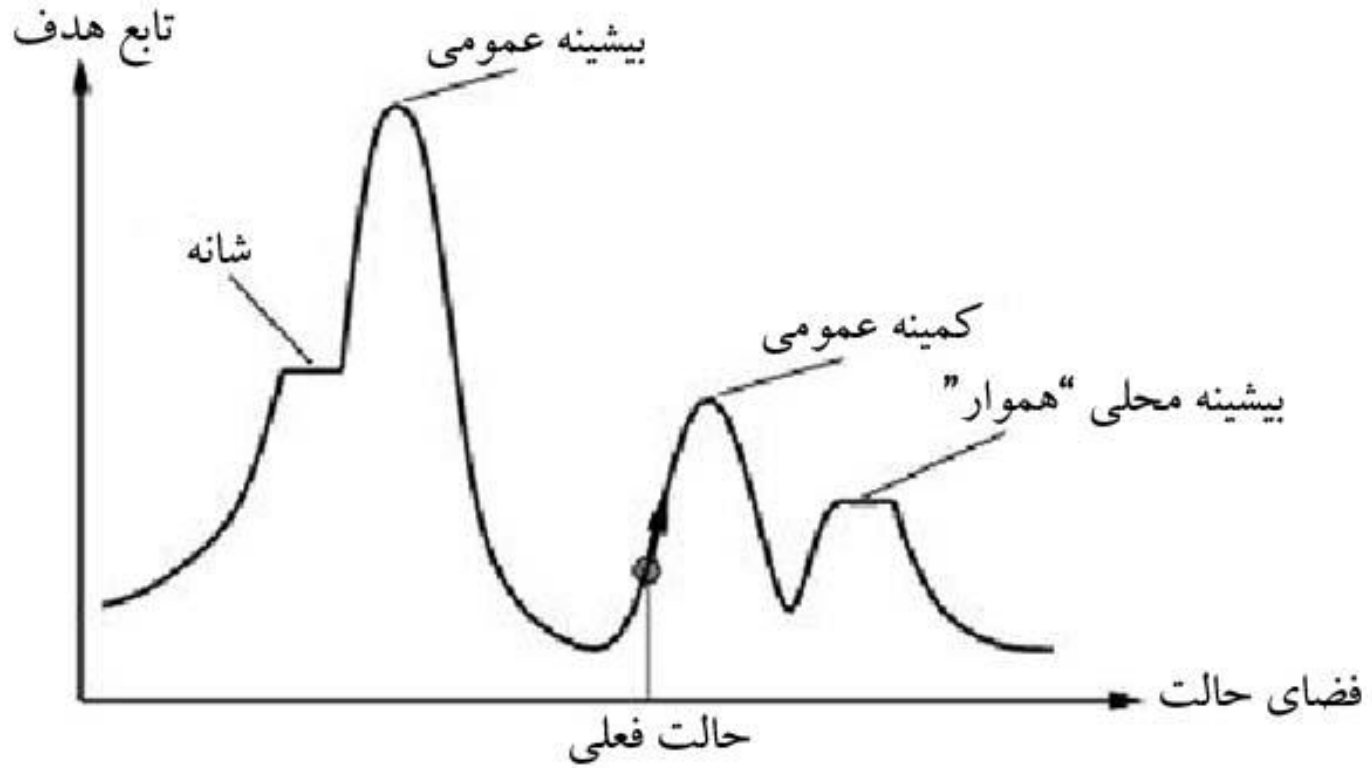
- مسئله راحت مسئله ای است که تعدادی از محدودیت های عملگرهای آن حذف شده باشند.
- جواب دقیق یک مسئله راحت معمولاً یک تخمین برای مسئله اصلی است.
- مثال معمای ۸ : یک چهار خانه می تواند از خانهٔ A به خانهٔ Bحرکت کند، اگر A همسایهٔ B باشد و B یک خانهٔ خالی باشد.
  - یک خانه مربع می تواند از خانهٔ A به خانهٔ B برود اگر B خالی باشد.
  - یک خانه مربع می تواند از خانهٔ A به خانهٔ B برود.
  - یک خانه مربع می تواند از خانهٔ A به خانهٔ B برود اگر مجاور همدیگر باشند.

الگوریتم های جست و جوی محلی و بهینه سازی

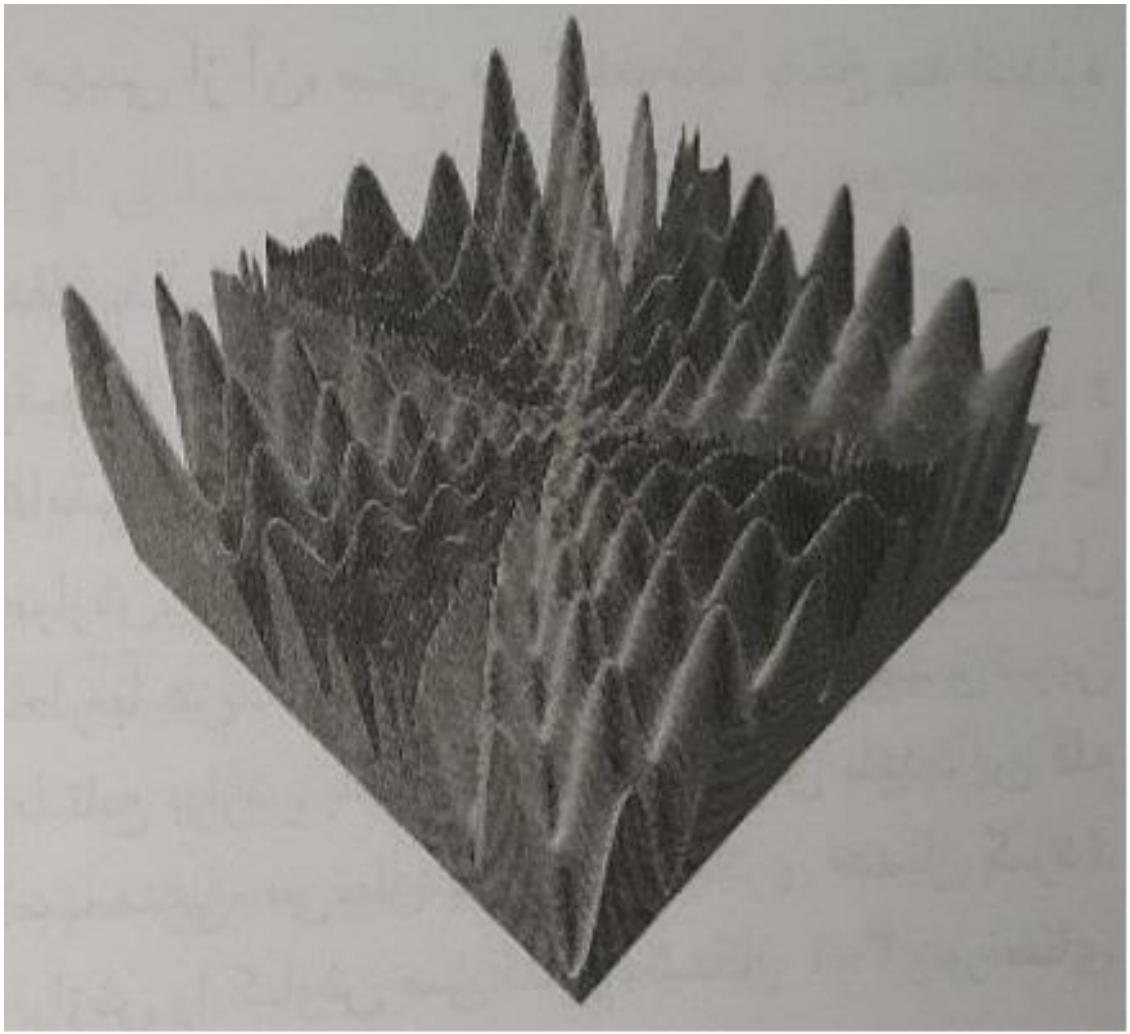
- الگوریتم های قبلی، فضای جست و جو را به طور سیستماتیک بررسی میکنند
  - تا رسیدن به هدف یک یا چند مسیر نگهداری میشوند
  - مسیر رسیدن به هدف، راه حل مسئله را تشکیل میدهد
- در الگوریتم های محلی مسیر رسیدن به هدف مهم نیست
  - مثال: مسئله ۸ وزیر
- دو امتیاز عمده جست و جوهای محلی
  - استفاده از حافظه کمکی
  - ارائه راه حلهای منطقی در فضاهای بزرگ و نامتناهی
- این الگوریتمها برای حل مسائل بهینه سازی نیز مفیدند
  - یافتن بهترین حالت بر اساس تابع هدف

جست و جوی آگاهانه و اکتشاف

الگوریتم های جست و جوی محلی و بهینه سازی



فضای جستجوی دوبعدی



جستجوی تپه نوردی (Hill-Climbing)

- در جهت افزایش مقدار پیوسته حرکت می کند
- زمانی متوقف می شود که به نقطهٔ اوجی برسد که همسایه بالاتری نداشته باشد.
- الگوریتم درخت جستجو نمی سازد،
  - بنابراین ساختار دادهٔ گره جاری تنها نیاز به ثبت حالت و مقدار تابع هدف دارد.
- تپه نوردی ماوراء همسایگان بلافصل گره جاری را نگاه نمی کند.
  - این همانند آن است که بخواهیم بالای قلهٔ اورست را در حالی پیدا کنیم که قله در مه غلیظی فرو رفته و ما دچار فراموشی هستیم.

جستجوی تپه نوردی (Hill-Climbing)

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
```

جستجوی تپه نوردی (Hill-Climbing)

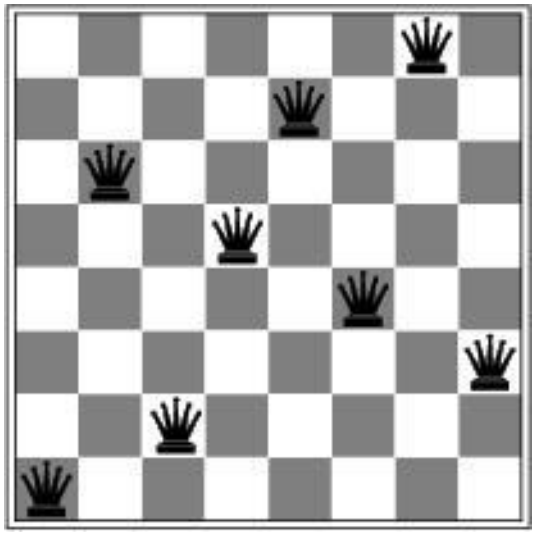
مثال: مسئله ۸ وزیر

- مسئله ۸ وزیر با استفاده از فرمولبندی حالت کامل
  - در هر حالت ۸ وزیر در صفحه قرار دارند
- تابع جانشین: انتقال یک وزیر به مربع دیگر در همان ستون
- تابع اکتشاف: جفت وزیرهایی که نسبت به هم گارد دارند

جستجوی تپه نوردی (Hill-Climbing)

الف

ب

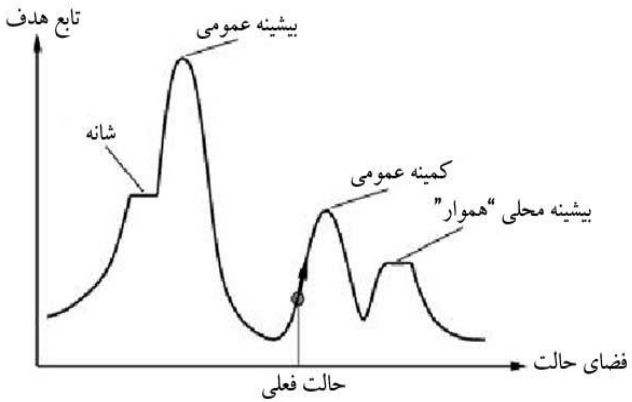


الف- حالت با هزینه h=17 که مقدار h را برای هر جانشین نشان میدهد

ب- کمینه محلی در فضای حالت ۸ وزیر؛ h=1

جستجوی تپه نوردی (Hill-Climbing)

- حلقه ای که در جهت افزایش مقدار حرکت میکند(بطرف بالای تپه)
  - رسیدن به بلندترین قله در همسایگی حالت فعلی، شرط خاتمه است.
- ساختمان داده گره فعلی، فقط حالت و مقدار تابع هدف را نگه میدارد
- جست و جوی محلی حریصانه نیز نام دارد
  - بدون فکر قبلی حالت همسایه خوبی را انتخاب میکند
- تپه نوردی به دلایل زیر میتواند متوقف شود:
  - بیشینه محلی
  - شانه
  - فلات



Iterative Improvement 3

Simple Iterative Improvement or Hill Climbing:

- Candidate is always and only accepted if cost is lower (or fitness is higher) than current configuration
- Stop when no neighbor with lower cost (higher fitness) can be found

Disadvantages:

- Local optimum as best result
- Local optimum depends on initial configuration
- Generally, no upper bound can be established on the number of iterations

How to cope with disadvantages

1. **Repeat** algorithm many times with **different initial** configurations

2. Use information **gathered in previous runs**

3. Use a more complex Generation Function to jump out of local optimum

4. Use a more complex **Evaluation Criterion** that accepts sometimes (randomly) also solutions away from the (local) optimum

جستجوی بازپخت شبیه سازی شده (Simulated Annealing)

- تپه نوردی مرکب با حرکت تصادفی
- شبیه سازی حرارت: حرارت با درجه بالا و به تدریج سرد کردن
- مقایسه با حرکت توپ
  - توپ در فرود از تپه به عمیق ترین شکاف میرود
  - با تکان دادن سطح توپ از بیشینه محلی خارج میشود
  - با تکان شدید شروع(دمای زیاد)
  - بتدریج تکان کاهش(به دمای پایین تر)
- با کاهش زمانبندی دما به تدریج، الگوریتم یک بهینه عمومی را می یابد

جستجوی بازپخت شبیه سازی شده (Simulated Annealing)

- به جای انتخاب بهترین حرکت، حرکتی تصادفی انتخاب خواهد شد.
- اگر حرکت، وضعیت را بهبود دهد همواره پذیرفته خواهد شد.
- در غیر این صورت، الگوریتم حرکت را با احتمالی کمتر از ۱ قبول خواهد کرد.
- این احتمال به صورت نمایی با "بد بودن حرکت" کاهش می یابد
- مقدار ΔE که با آن ارزیابی بدتر شده است.

$$\Delta E = Value[next] - Value[current]$$

$$P = \frac{۱}{-\frac{\Delta E}{T}}$$

Simulation of cooling (Metropolis 1953)

- At a fixed temperature T :
- Perturb (randomly) the current state to a new state
- ΔE is the difference in energy between current and new state
- If $\Delta E < 0$ (new state is lower), accept new state as current state
- If $\Delta E \geq 0$, accept new state with probability

$$Pr(accepted) = \exp(-\Delta E / k_B.T)$$

- Eventually the systems evolves into thermal equilibrium at temperature T ; then the formula mentioned before holds

- When equilibrium is reached, temperature T can be lowered and the process can be repeated

Control Parameters

1. How to define equilibrium?
2. How to calculate new temperature for next step?

1. Definition of equilibrium
    1. Definition is reached when we cannot yield any significant improvement after certain number of loops
    2. A constant number of loops is assumed to reach the equilibrium

2. Annealing **schedule** (i.e. How to reduce the temperature)
    1. A constant value is subtracted to get new temperature, $T' = T - T_d$
    2. A constant scale factor is used to get new temperature, $T' = T * R_d$
        - A scale factor usually can achieve better performance

Example of Simulated Annealing

- Traveling Salesman Problem (TSP)
  - Given 6 cities and the traveling cost between any two cities
  - A salesman need to start from city 1 and travel all other cities then back to city 1
  - Minimize the total traveling cost

Example: SA for traveling salesman

- Solution representation
  - An integer list, i.e., (1,4,2,3,6,5)
- Search mechanism
  - Swap any two integers (except for the first one)
    - (1,4,2,3,6,5) → (1,4,3,2,6,5)
- Cost function

Example of Not traveling salesman

- Temperature
  1. Initial temperature determination
     1. Initial temperature is set at such value that there is around 80% acceptation rate for “bad move”
     2. Determine acceptable value for ($C_{new} – C_{old}$)
  2. Final temperature determination
     - Stop criteria
     - Solution space coverage rate

- Annealing **schedule** (i.e. How to reduce the temperature)
  - A constant value is subtracted to get new temperature, $T' = T – T_d$
    - For instance new value is 90% of previous value.
    - Depending on solution space coverage rate

Homogeneous Algorithm of Simulated Annealing

```
initialize;
REPEAT
  REPEAT
    perturb ( config.i → config.j, ΔC_ij);
    IF ΔC_ij < 0 THEN accept
    ELSE IF exp(-ΔC_ij/c) > random[0,1) THEN accept;
    IF accept THEN update(config.j);
  UNTIL equilibrium is approached sufficient closely;
  c := next_lower(c);
UNTIL system is frozen or stop criterion is reached
```

In homogeneous algorithm the value of c is kept constant in the inner loop and is only decreased in the outer loop

جست و جوی پرتو محلی

- به جای یک حالت، k حالت را نگهداری میکند
  - حالت اولیه: k حالت تصادفی
  - گام بعد: جانشین همه k حالت تولید میشود
  - اگر یکی از جانشین ها هدف بود، تمام میشود
  - وگر نه بهترین جانشین را انتخاب کرده، تکرار میکند
- تفاوت عمده با جستجوی شروع مجدد تصادفی
  - در جست و جوی شروع مجدد تصادفی، هر فرایند مستقل از بقیه اجرا میشود
  - در جست و جوی پرتو محلی، اطلاعات مفیدی بین k فرایند موازی مبادله میشود
- جست و جوی پرتو غیرقطعی
  - به جای انتخاب بهترین k از جانشینها، k جانشین تصادفی را بطوریکه احتمال انتخاب یکی تابعی صعودی از مقدارش باشد، انتخاب میکند

الگوریتم ژنتیک

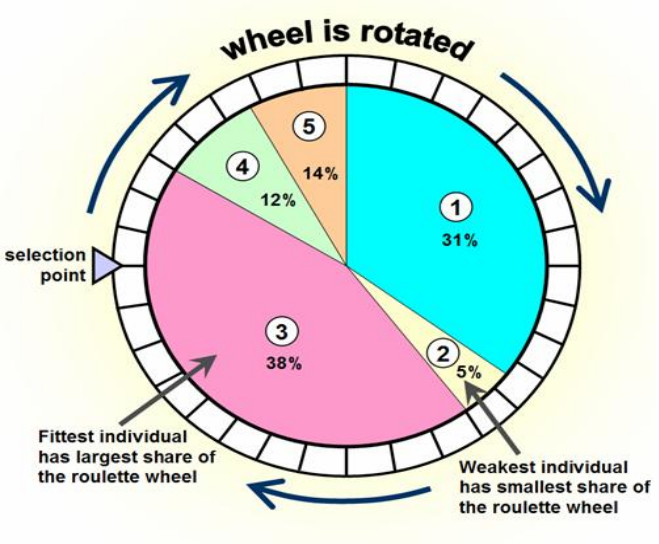
- الگوریتم ژنتیک یک تکنیک برنامه‌نویسی است که از تکامل ژنتیکی به عنوان یک الگوی حل مسئله استفاده می‌کند. در علوم کامپیوتر این الگوریتم یک تکنیک جستجو برای یافتن راه‌حل تقریبی برای مسائل بهینه‌سازی و جستجو است.
- در الگوریتم ژنتیک، مسئله‌ای که باید حل شود ورودی الگوریتم است و راه‌حلها طبق یک الگوی تصادفی ایجاد می‌شوند.
- الگوریتم ژنتیک از بخش های اصلی زیر تشکیل می‌شود:
  - نمایش
  - تابع برازندگی
  - انتخاب
  - ترکیب
  - جهش

الگوریتم ژنتیک

- **نمایش :** یک راه‌حل برای مسئله مورد نظر با لیستی از پارامترها نمایش داده می‌شود که به مجموعه آنها یک فرد گفته می شود.
  - عموماً اگر نمایش یک فرد به صورت یک رشته ساده از صفر و یک باشد به آن کروموزوم یا ژنوم می‌گویند.
- **تابع برازندگی :** هر کروموزوم یا فرد با استفاده از تابع برازندگی مورد آزمایش قرار می‌گیرد.
  - از آنجا که پارامترهای موردنظر الگوریتم جستجو در تابع برازندگی لحاظ می-شوند، مقدار برگشتی این تابع برای هر راه حل، میزان رضایت الگوریتم جستجو از راه حل پیشنهادی را نشان می-دهد.
  - با استفاده از مقدار برگشتی تابع برازندگی، می-توان مناسب ترین راه حل ها را برای تولید نسل بعدی انتخاب نمود.

الگوریتم ژنتیک

- **انتخاب:** در این مرحله تعدادی از راه حل های یک نسل، برای ایجاد نسل بعدی، انتخاب میشود.
  - انتخاب‌ها به گونه‌ای هستند که مناسبترین راه حل ها انتخاب شوند و معمولا به گونه ای انجام میشود که حتی ضعیفترین عناصر هم شانس انتخاب داشته باشند.
  - این کار باعث می شود که از گیر افتادن در نقاط بیشینه/کمینه محلی جلوگیری شود.
  - چندین الگوی انتخاب وجود دارد که مهمترین آنها عبارتند از :
    - انتخاب مسابقه‌ای
    - انتخاب چرخ گردان .

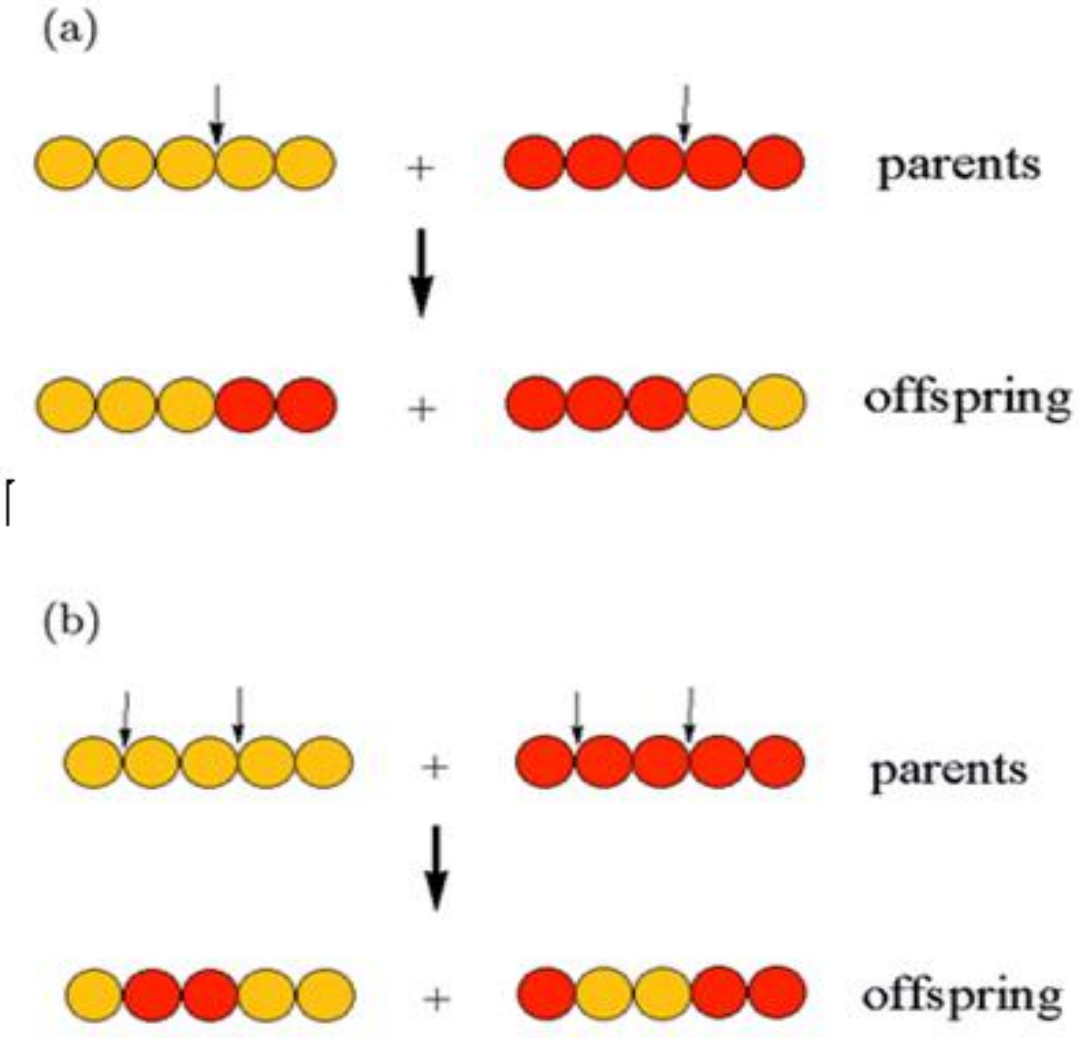


الگوریتم ژنتیک

- **ترکیب :** در مرحله بعد عناصر انتخاب شده با یکدیگر ترکیب شده و نسل جدیدی ایجاد میکنند.
- روشهای مختلفی برای ترکیب افراد و کروموزومها وجود دارد که مهمترین آنها عبارتنداز : عملگر یک نقطه‌ای و عملگر دو نقطه ای.
  - در عملگر یک نقطه ای یک عدد تصادفی بین یک و طول رشته تولید می‌شود. پس از مشخص شدن این عدد صحیح که نشان دهنده مکان تبادل روی رشته‌ها است, هر دو رشته از محلی که این عدد مشخص می‌کند شکسته می‌شوند و قسمت‌های انتهایی آنها با یکدیگر معاوضه می‌شوند، تا دو رشته جدید حاصل شود.
  - در روش دو نقطه ای، دو عدد تصادفی بین یک و طول رشته برای هر رشته انتخاب شده و زیر رشته بین این دو عدد در رشته های اصلی جابجا می شوند

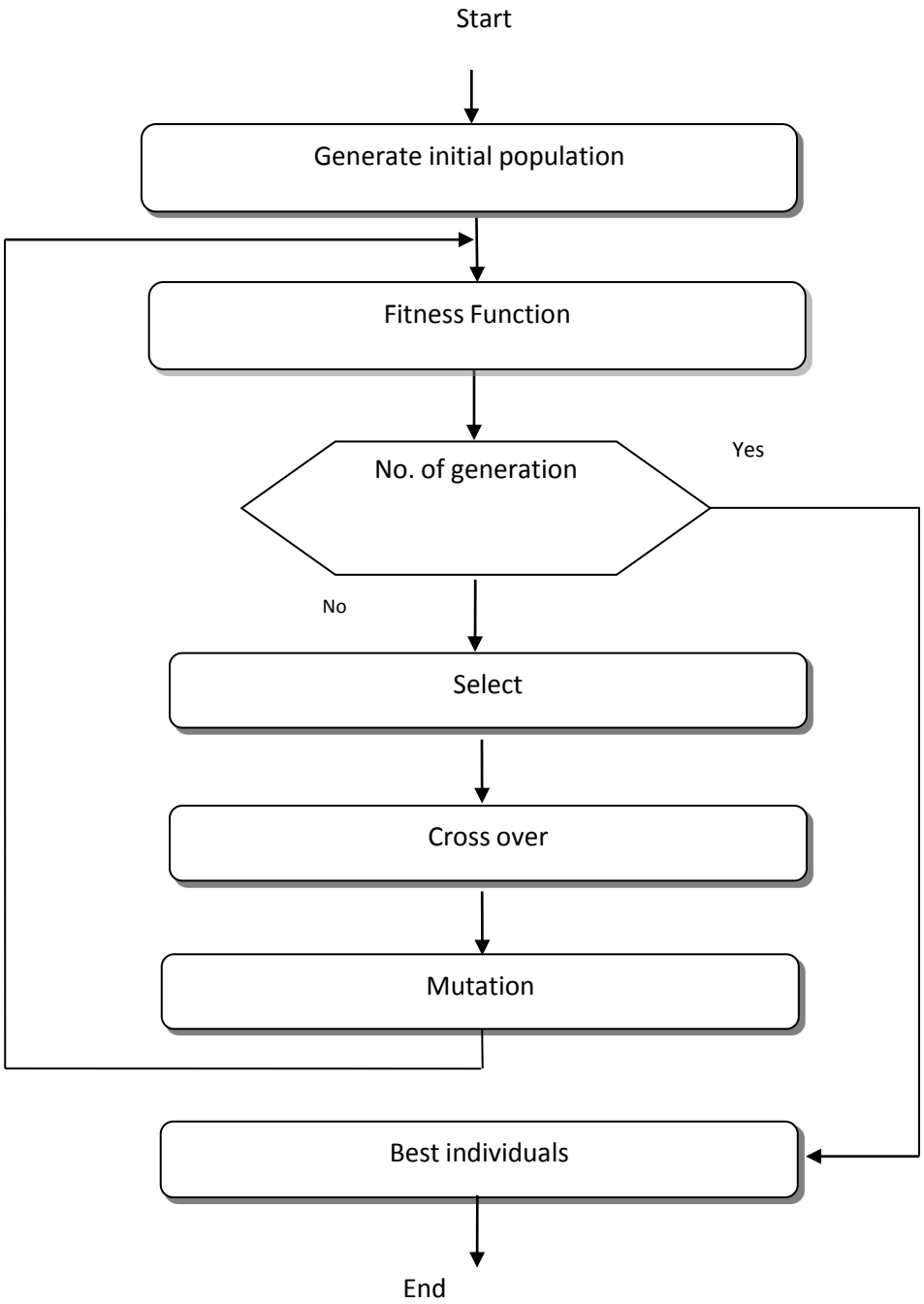
(a) ترکیب یک نقطه ای

(b) ترکیب دو نقطه ای

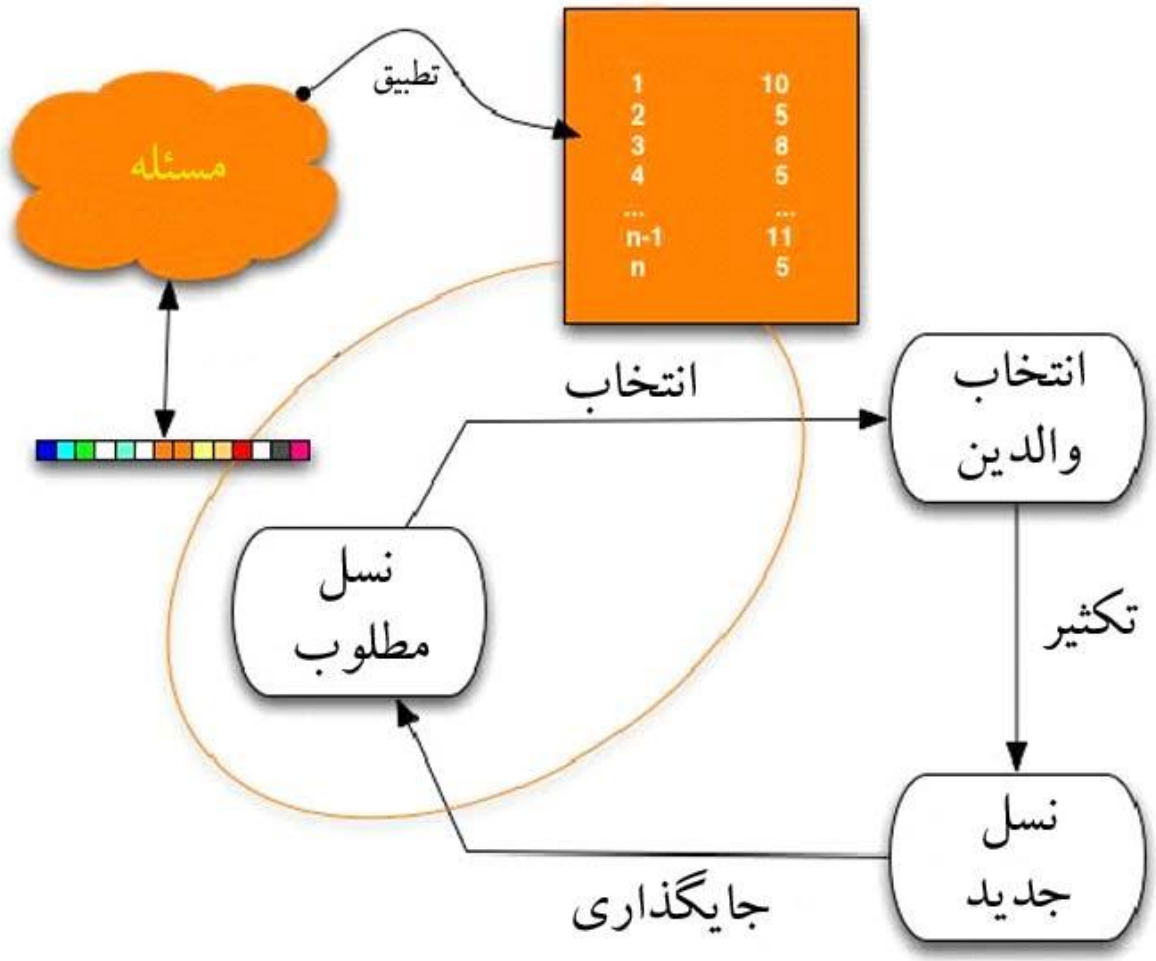


الگوریتم ژنتیک

- **جهش :** مرحله بعدی تغییر دادن فرزندان جدید است. الگوریتم‌های ژنتیک یک احتمال تغییر کوچک و ثابت دارند که معمولاً درجه‌ای در حدود ۰,۰۱ یا کمتر دارد. بر اساس این احتمال، کروموزوم‌های فرزند به طور تصادفی تغییر می‌کنند یا جهش می‌یابند. این تغییر باعث افزایش شانس انتخاب عناصری می شود که احتمال انتخاب کمتری دارند.
- آزمایش ها نشان می-دهند که با گذر از تعداد زیادی از نسلها، الگوریتم ژنتیک به سمت راه حلهای دقیقتر میل می‌کند. این فرآیند تکرار می‌شود تا این که به آخرین مرحله برسیم
- شرایط خاتمه الگوریتم‌های ژنتیک عبارتند از:
  - به تعداد ثابتی از نسل‌ها برسیم.
  - بودجه اختصاص داده‌شده تمام شود(زمان محاسبه/پول).
  - یک فرد (فرزند تولید شده) پیدا شود که مینیمم (کمترین) ملاک را برآورده کند.
  - بیشترین درجه برازندگی فرزندان حاصل شود یا دیگر نتایج بهتری حاصل نشود.
  - بازرسی دستی.
  - ترکیبهای بالا.

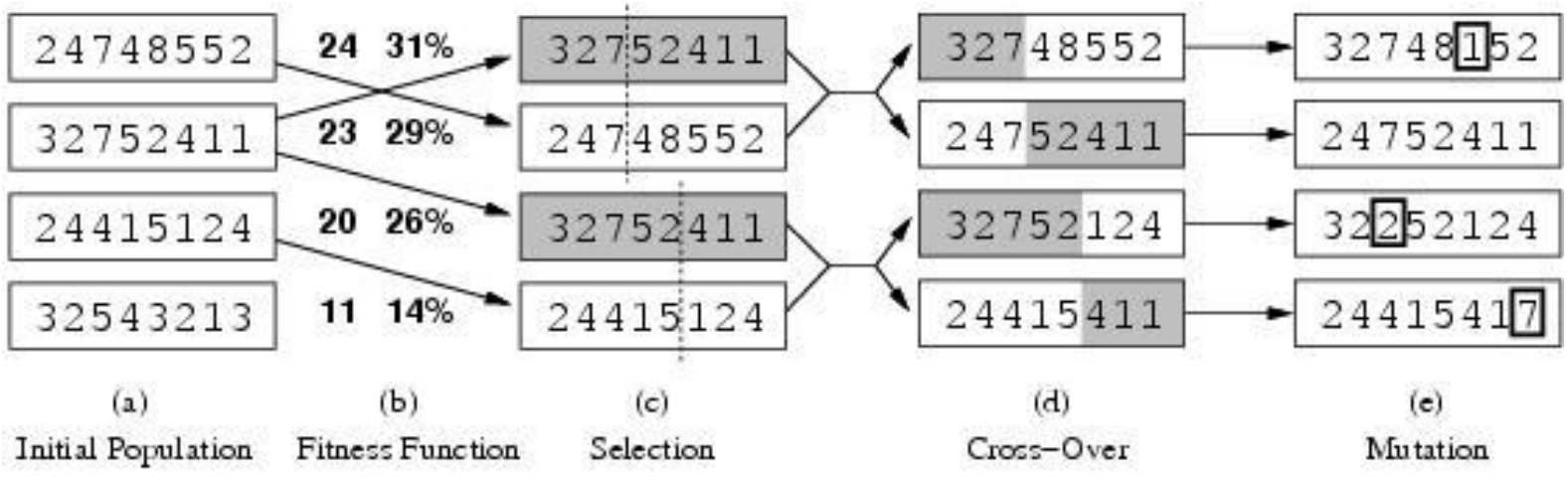

Start
Generate initial population
Fitness Function
No. of generation
Yes
No
Select
Cross over
Mutation
Best individuals
End

الگوریتم های ژنتیک

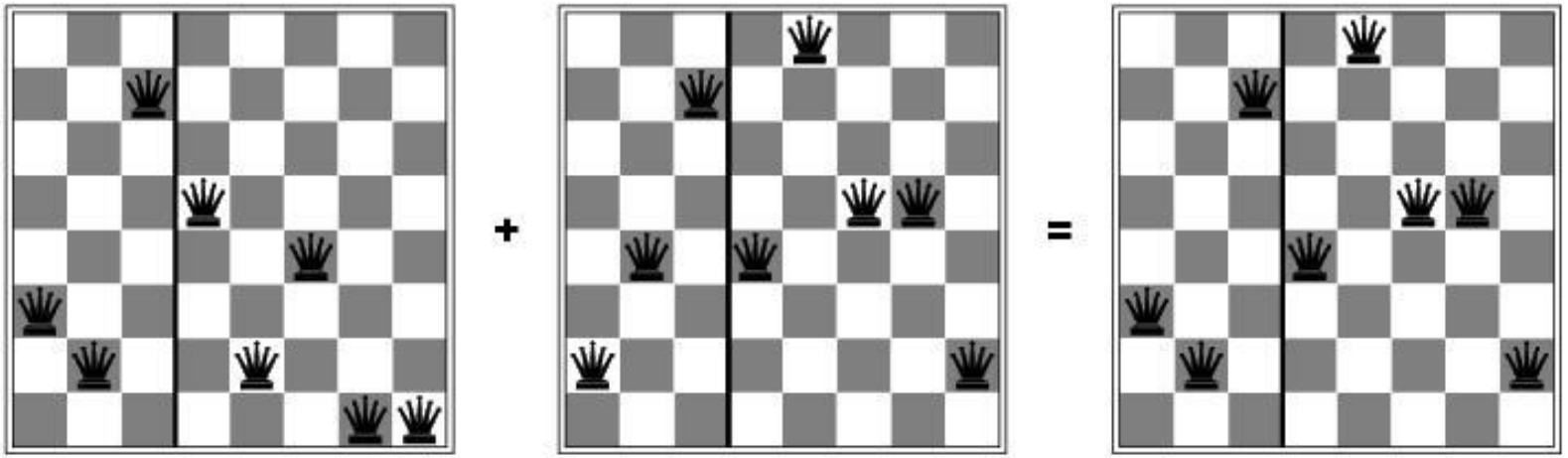


شکلی از جست و جوی پرتو غیر قطعی که حالتهای جانشین از طریق ترکیب دو حالت والد تولید میشود

الگوریتم های ژنتیک



جهت اولیه تابع برازش انتخاب تقاطع جهش



الگوریتم های ژنتیک

function GENETIC-ALGORITHM(*population*, FITNESS-FN) **returns** an individual
  inputs: *population*, a set of individuals
    FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new_population ← empty set
    loop for i **from** 1 **to** SIZE(*population*) **do**
      x ← RANDOM-SELECTION(*population*, FITNESS-FN)
      y ← RANDOM-SELECTION(*population*, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *new_population*
    population ← *new_population*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to FITNESS-FN

---

function REPRODUCE(x, y) **returns** an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, $c+1$, n))