سازمان سما

وابسته به دانشگاه آزاد اسلامی

دانشگاه سما واحد حاجی آباد

هوش مصنوعی

منبع : هوش مصنوعی دکتر محمد صبری

WWW.HREZAPOUR.IR

حمیدرضا رضاپور

هوش مصنوعی، رهیافتی نوین

فصل چهارم: «جستجوهای آگاهانه»

فهرست مطالب

- استراتژی‌های آگاهانه
  - جستجوی حریصانه
  - جستجوی A*
- کیفیت توابع هیوریستیک
- ابداع توابع هیوریستیک
- جستجوی محلی
  - تپه نوردی
  - SA
  - پرتوی محلی
  - ژنتیک

فهرست مطالب

- استراتژی‌های آگاهانه
  - جستجوی حریصانه
  - جستجوی A*
- کیفیت توابع هیوریستیک
- ابداع توابع هیوریستیک
- جستجوی محلی
  - تپه نوردی
  - SA
  - پرتوی محلی
  - ژنتیک

جستجوی درختی

```
function TREE-SEARCH(problem , fringe) return a solution or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if EMPTY?(fringe) then return failure
        node ← REMOVE-FIRST(fringe)
        if GOAL-TEST[problem] applied to STATE[node] succeeds
                then return SOLUTION(node)
        fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```

جستجوی سطحی

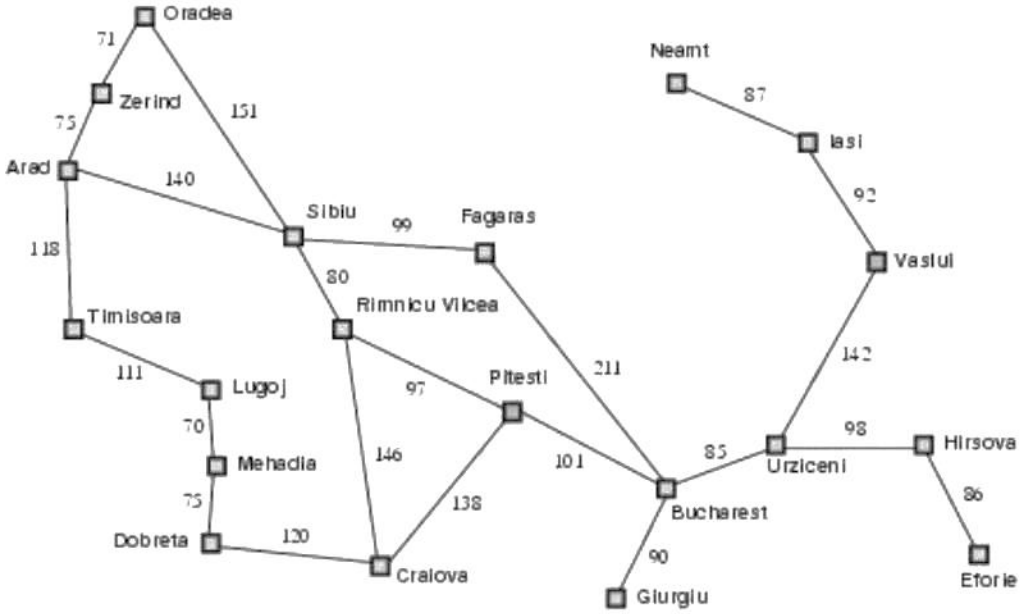
- روش کلی برای جستجوی آگاهانه
- جستجوی سطحی: گره برای بسط دادن بر اساس تابع ارزیابی *f(n)* انتخاب می‌شود.
- ایده: تابع ارزیابی فاصله تا هدف را اندازه‌گیری می‌کند. یعنی گره‌ایی را برای بسط دادن انتخاب می‌کند که به نظر بهترین باشد.
- پیاده سازی: *fringe* یک صف مرتب بر اساس *f(n)*

تابع هیورستیک

- [dictionary] *"A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood."*
  - $h(n)$ برابر با هزینه تخمینی راه حل بهینه از گره n تا هدف
  - اگر n برابر با هدف باشد، آنگاه $h(n)=0$ خواهد بود.

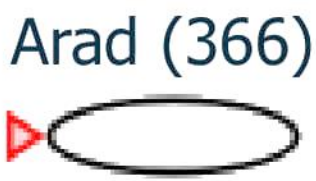
مساله کشور رمانی

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



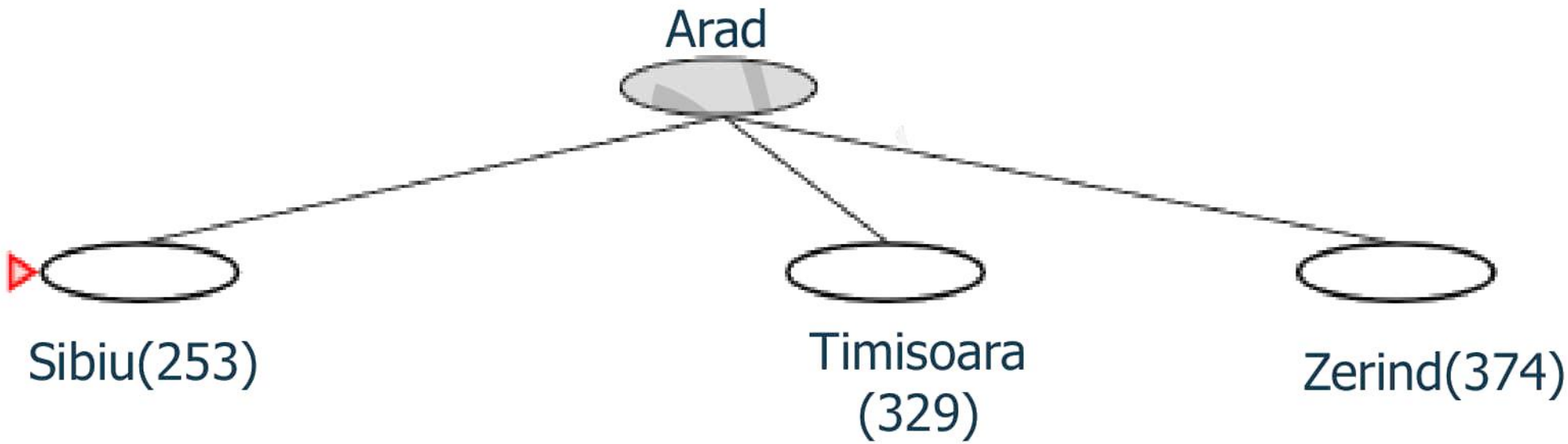
- h_{SLD} فاصله مستقیم هوایی هر گره تا هدف
- h_{SLD} نمی تواند از صورت مساله محاسبه شود.
- جستجوی حریصانه : $f(n)=h(n)$
- گره‌ایی را بسط بده که به هدف نزدیک‌تر باشد.

جستجوی حریصانه

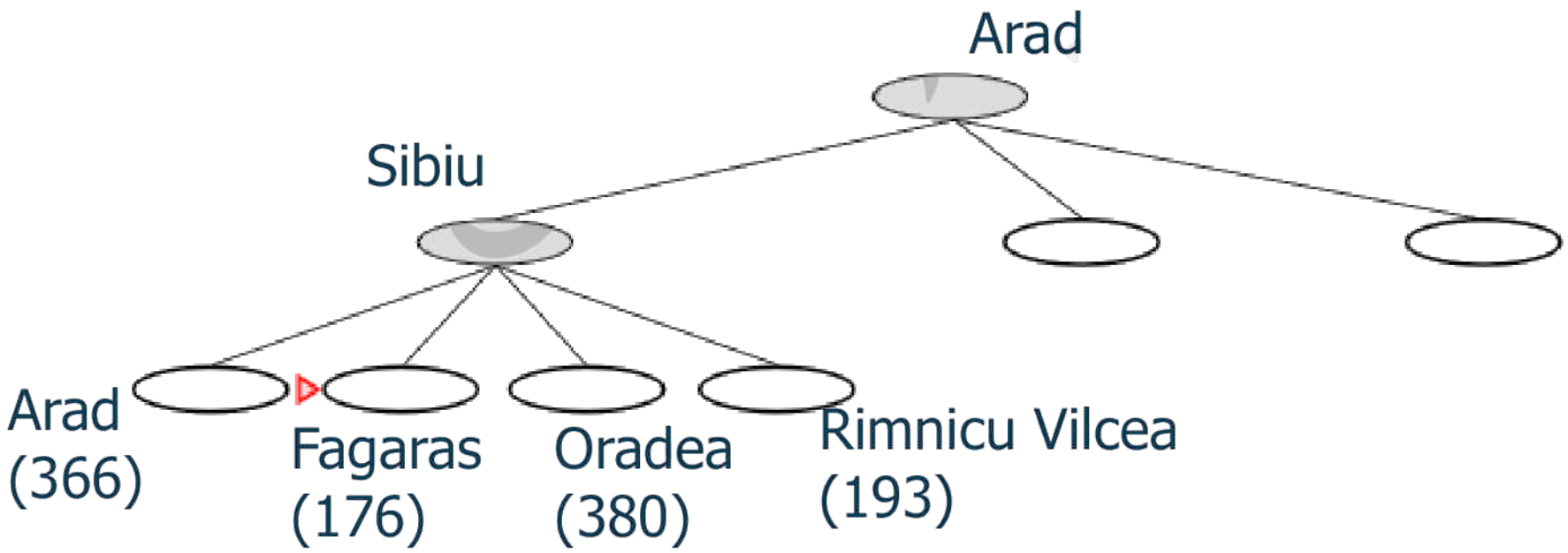


- ریشه = حالت شروع

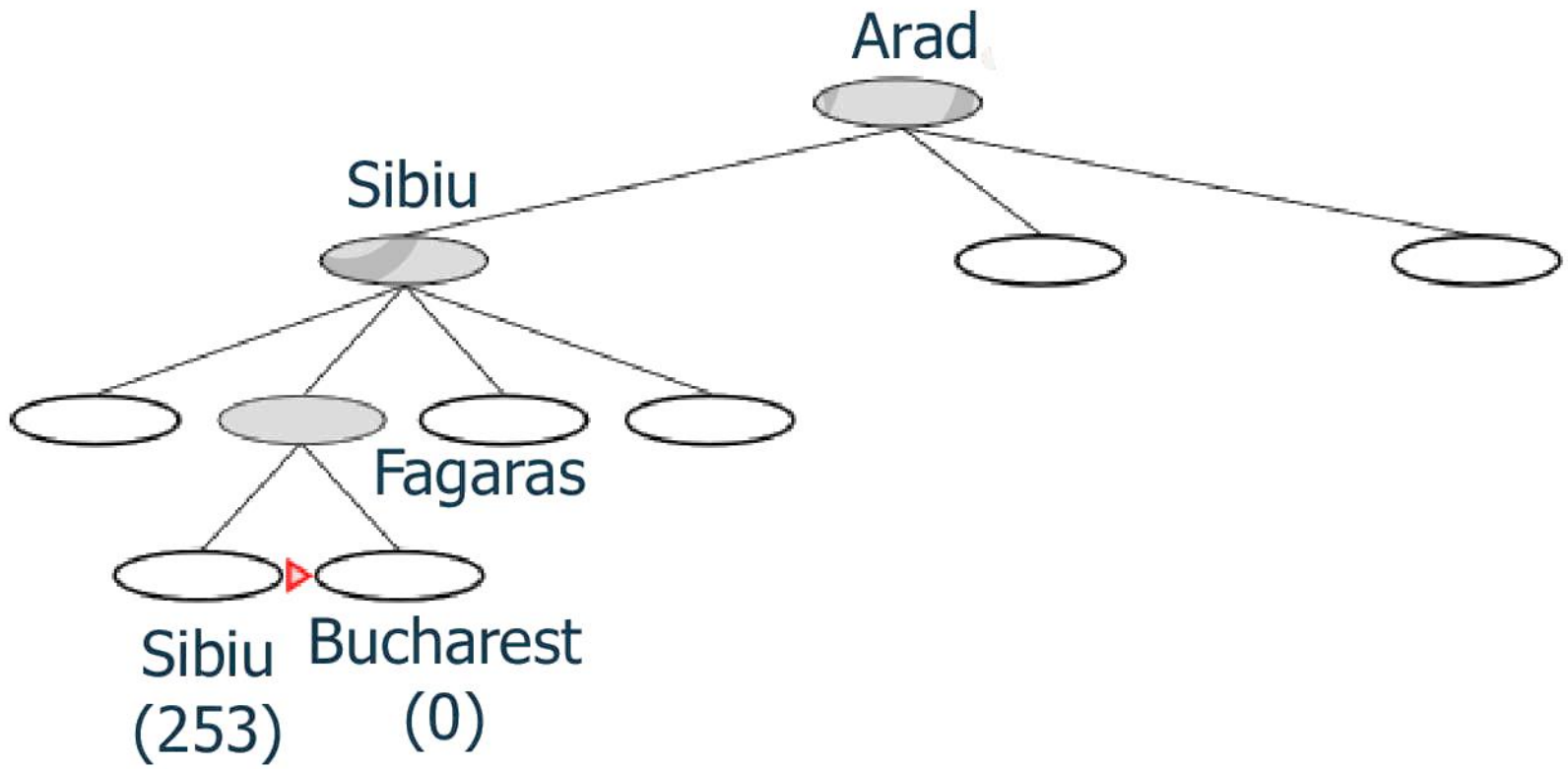
جستجوی حریصانه



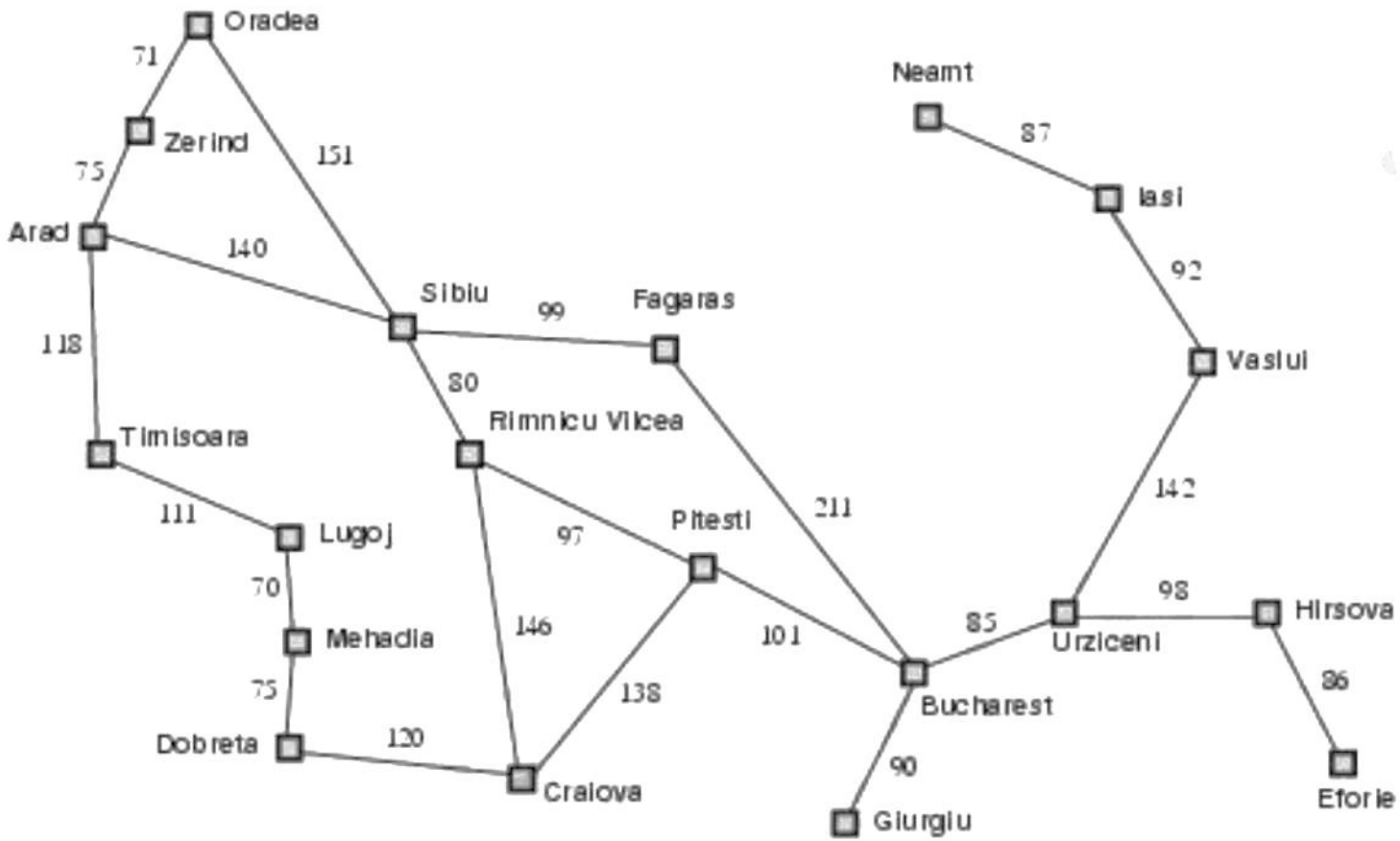
جستجوی حریصانه



جستجوی حریصانه



جستجوی حریصانه



- کامل بودن؟
- خیر (حلقه تکرار)
- بهینه بودن؟
- خیر (کامل نبودن)
- مرتبه زمانی؟ $O(b^m)$
- مرتبه حافظه؟ $O(b^m)$

جستجوی A*

- شناخته شده ترین جستجوی آگاهانه
- ایده: از بسط گره‌هایی که به صرفه به نظر نمی‌رسند، اجتناب می‌کند.
- تابع ارزیابی: $f(n)=g(n) + h(n)$
- $g(n)$ هزینه واقعی از گره شروع تا گره n
- $h(n)$ هزینه تخمینی از گره n تا هدف
- $f(n)$ هزینه تخمینی از گره شروع تا هدف با عبور از گره n

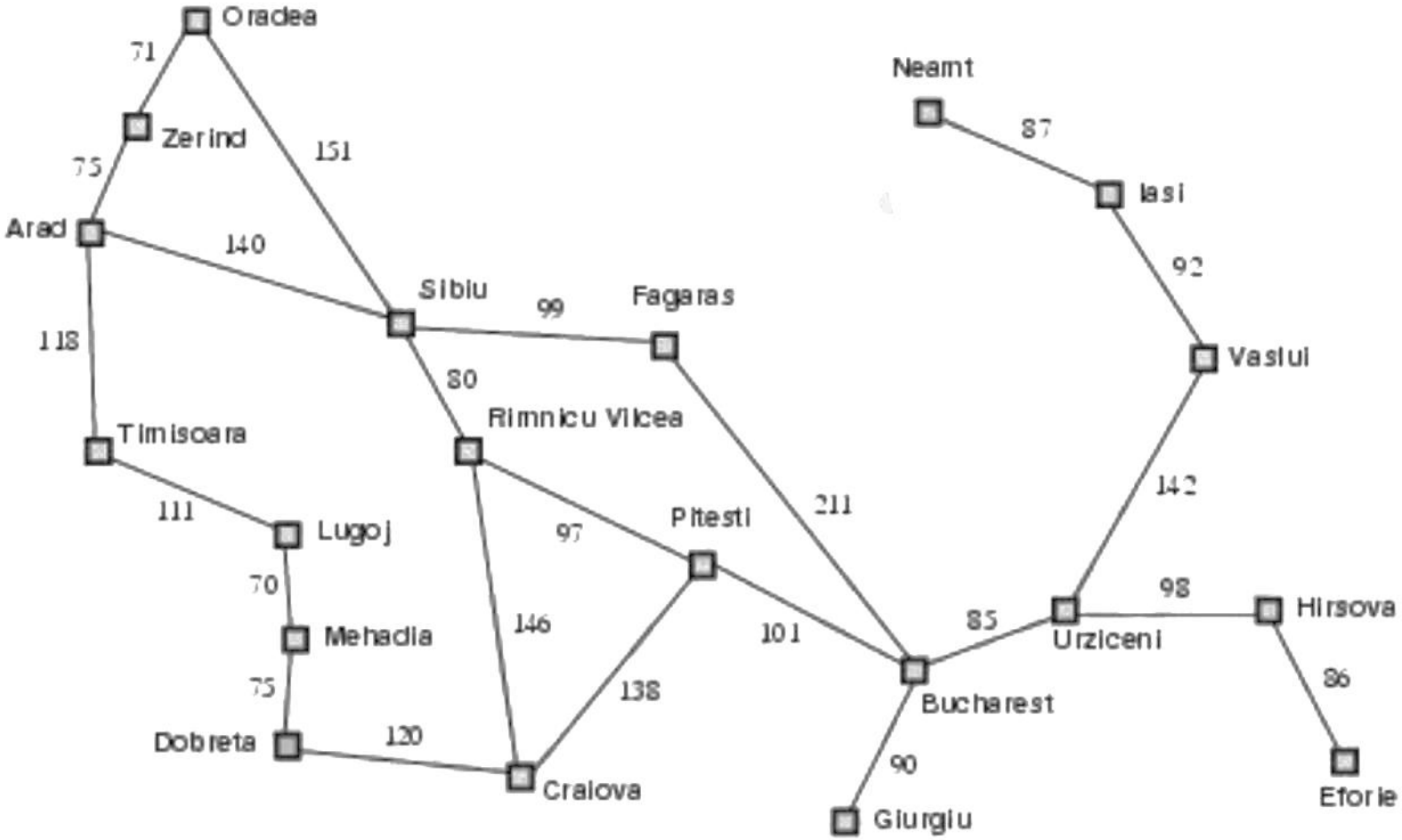
تابع هیوریستیک قابل پذیرش

- یک تابع هیوریستیک قابل پذیرش است:
  1. هیچگاه تخمین اضافی از هزینه تا هدف ارایه ندهد.
  2. خوش بینانه باشد.

Formally:

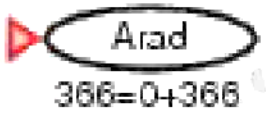
1. $h(n) <= h^*(n)$ where $h^*(n)$ is the true cost from n
2. $h(n) >= 0$ so $h(G)=0$ for any goal G.

- آیا $h_{SLD}(n)$ قابل پذیرش است؟


Oradea
71
Zerind
151
75
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Dobreta
120
Craiova
Neamt
87
Iasi
92
Vaslui
142
98
Hirsova
85
Urziceni
86
Bucharest
90
Giurgiu
Eforie

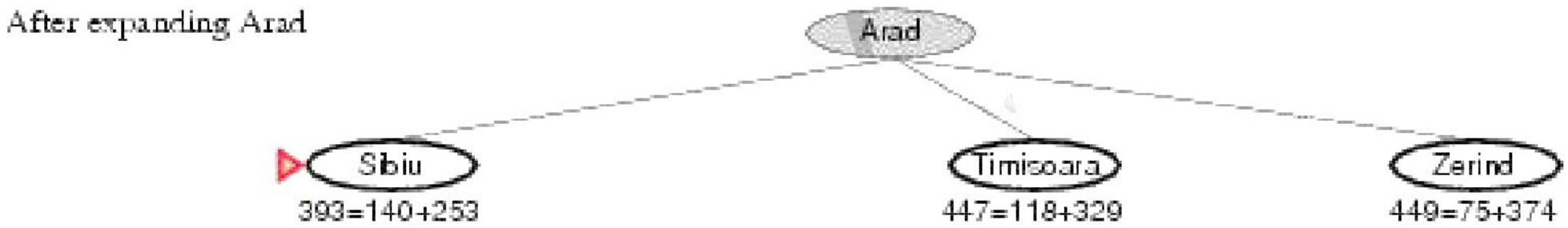
جستجوی A*

(a) The initial state



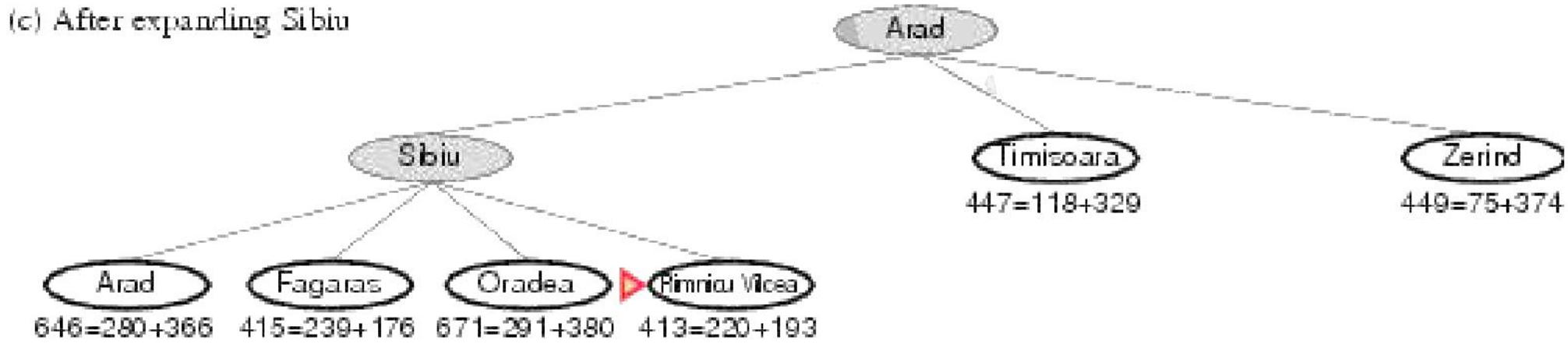
f(Arad) = c(??,Arad)+h(Arad)=0+366=366

جستجوی A*



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

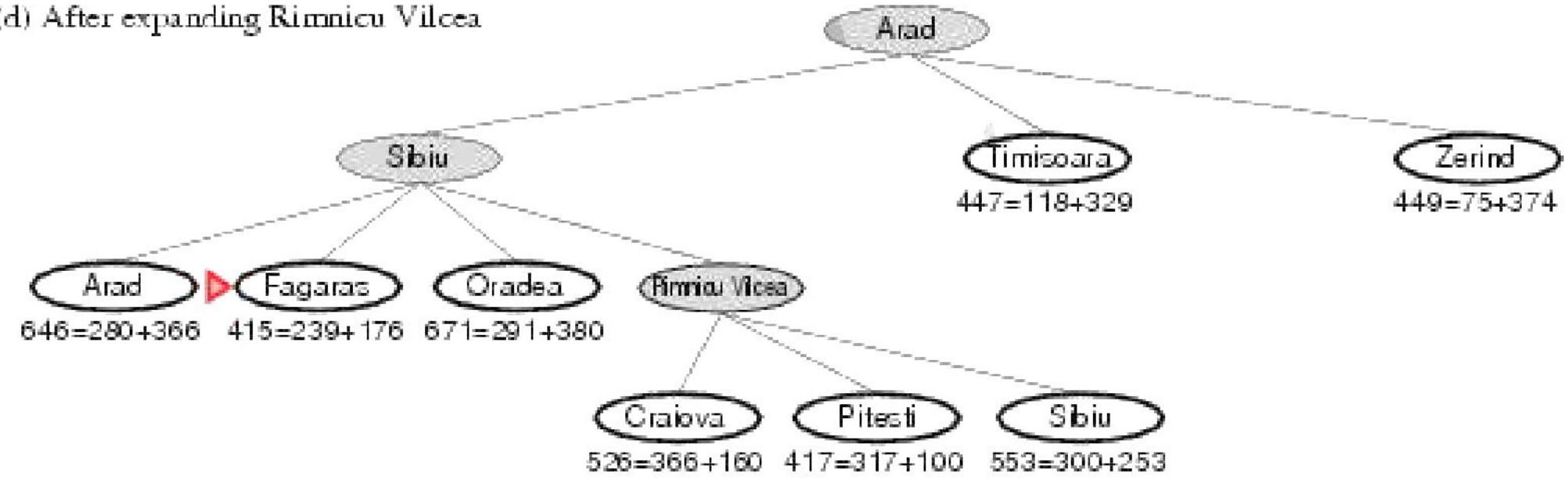
جستجوی A*



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

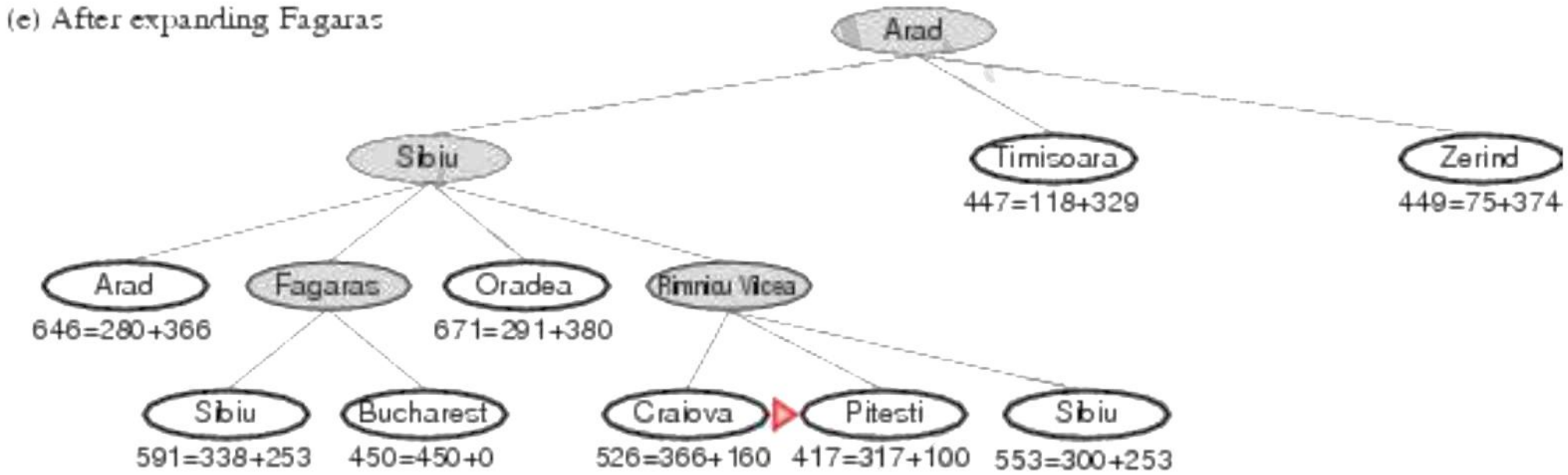
جستجوی A*

(d) After expanding Rimnicu Vilcea



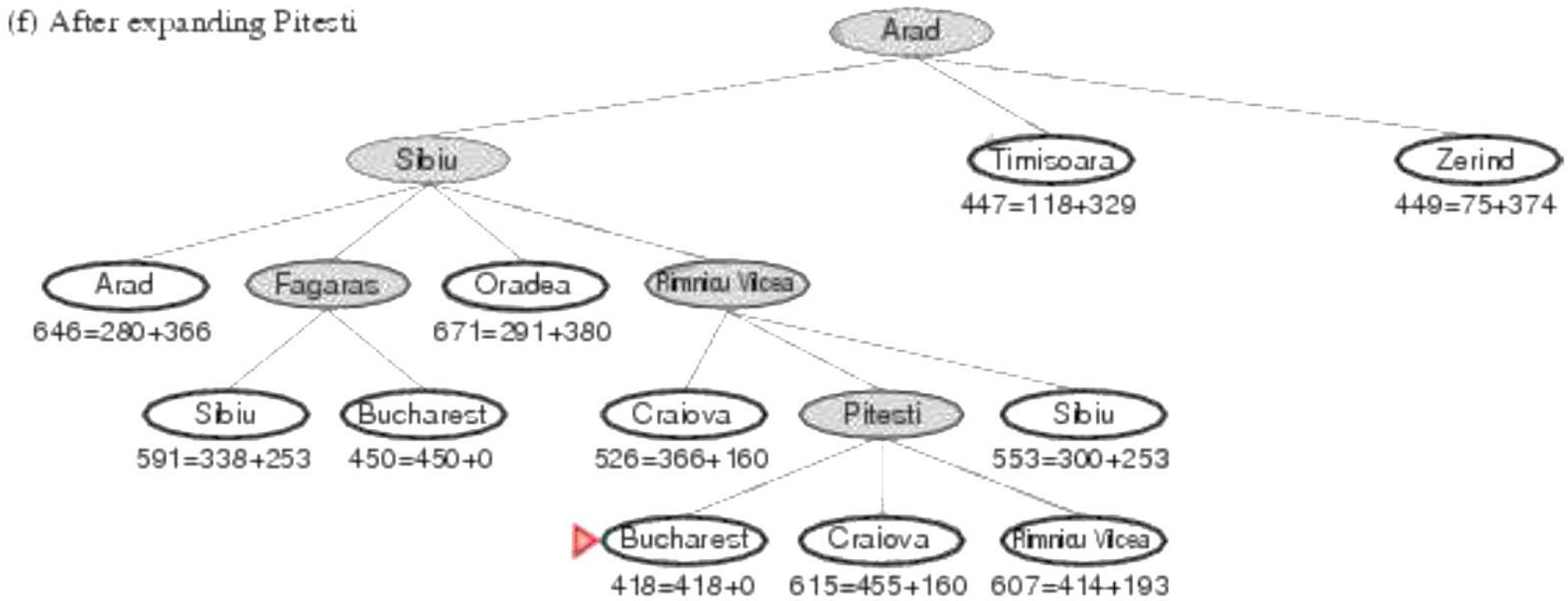
f(Craiova)=c(Rimnicu Vilcea, Craiova)+h(Craiova)=360+160=526
f(Pitesti)=c(Rimnicu Vilcea, Pitesti)+h(Pitesti)=317+100=417
f(Sibiu)=c(Rimnicu Vilcea,Sibiu)+h(Sibiu)=300+253=553

جستجوی A*



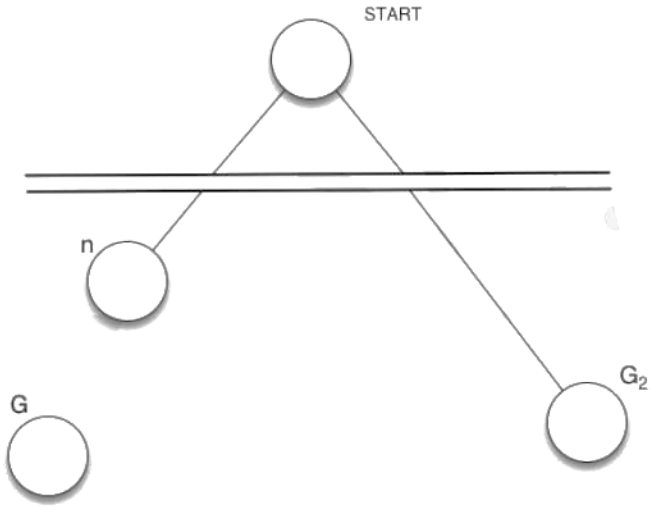
f(Sibiu)=c(Fagaras, Sibiu)+h(Sibiu)=338+253=591
f(Bucharest)=c(Fagaras,Bucharest)+h(Bucharest)=450+0=450

جستجوی A*



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

A* جستجوی



Suppose suboptimal goal G2 in the queue.
Let n be an unexpanded node on a shortest to optimal goal G.

f(G2) = g(G2) since h(G2)=0
> g(G) since G2 is suboptimal
>= f(n) since h is admissible

Since f(G2) > f(n), A* will never select G2 for expansion

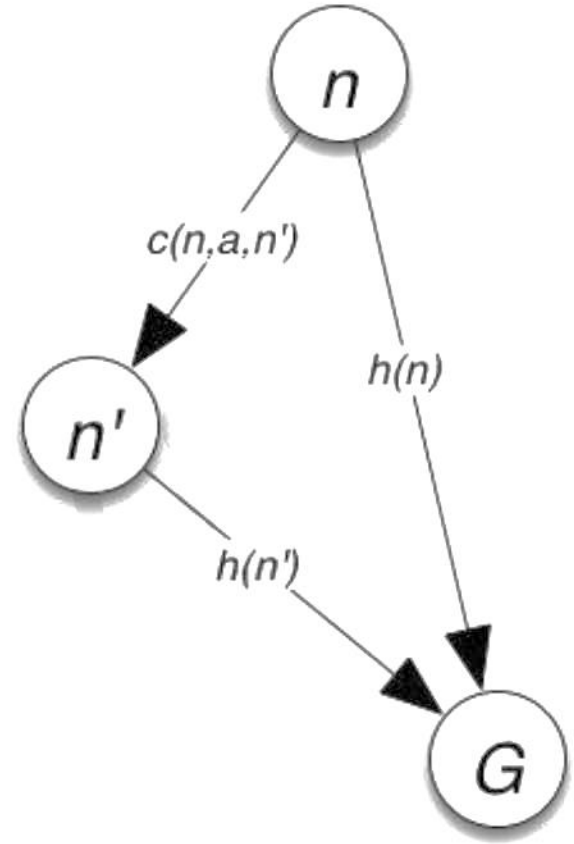
A* جستجوی

- A heuristic is consistent if

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

- If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\ge g(n) + h(n) \\ &\ge f(n) \end{aligned}$$



جستجوی A*

- کامل بودن؟
- بلی
- بهینه بودن؟
- بلی
  - A* expands all nodes with $f(n) < C^*$
  - A* expands some nodes with $f(n) = C^*$
  - A* expands no nodes with $f(n) > C^*$
- بنابراین A* بهینه موثر نیز می‌باشد.

جستجوی A*

- مرتبه زمانی؟
- کماکان از مرتبه نمایی می‌باشد.
- مرتبه حافظه؟
- هم مرتبه با پیچیدگی زمانی است.

! حافظه به مراتب مشکل بزرگتری است.

جستجوی A* با حافظه محدود

- بهبود مشکل حافظه A* با حفظ ویژگی کامل و بهینه بودن

1. ID A* : تعیین یک هزینه محدود *f*-cost (*g*+*h*) به جای عمق محدود
2. RBFS : الگوریتم بازگشتی برای داشتن حافظه خطی
3. (S)MA* : بسط بهترین گره جدید و حذف بدترین گره‌ی قدیمی در زمان پر شدن حافظه

Recursive best-first search

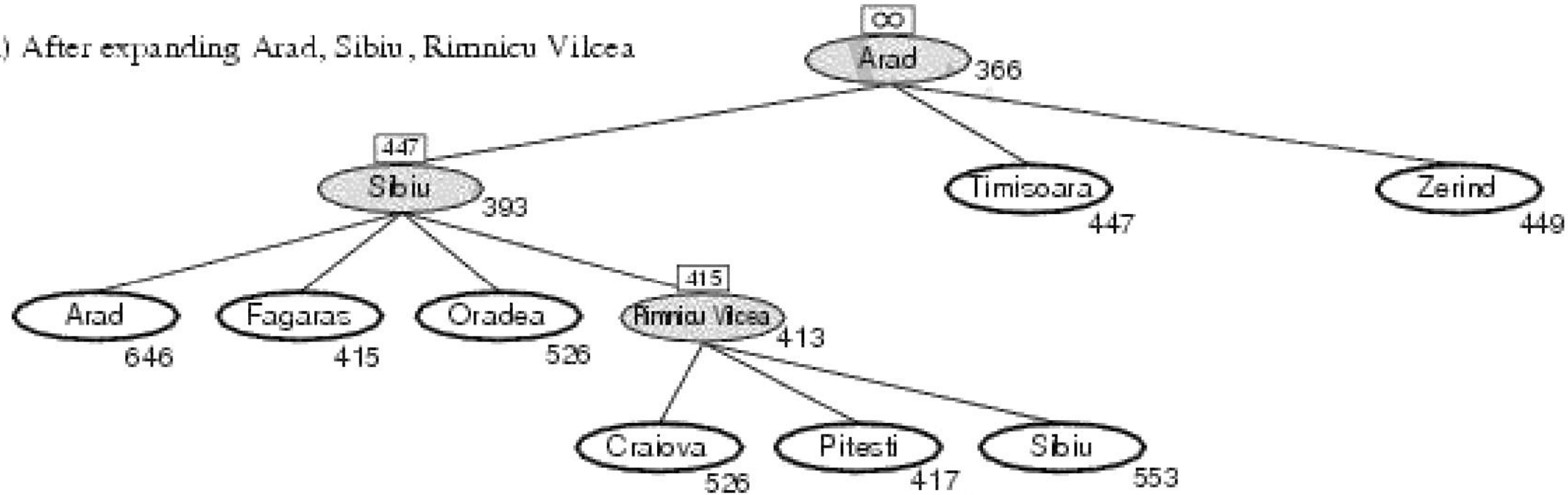
```
function RECURSIVE-BEST-FIRST-SEARCH(problem) return a solution or failure
    return RFBS(problem,MAKE-NODE(INITIAL-STATE[problem]),∞)

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

الگوریتم RBFS

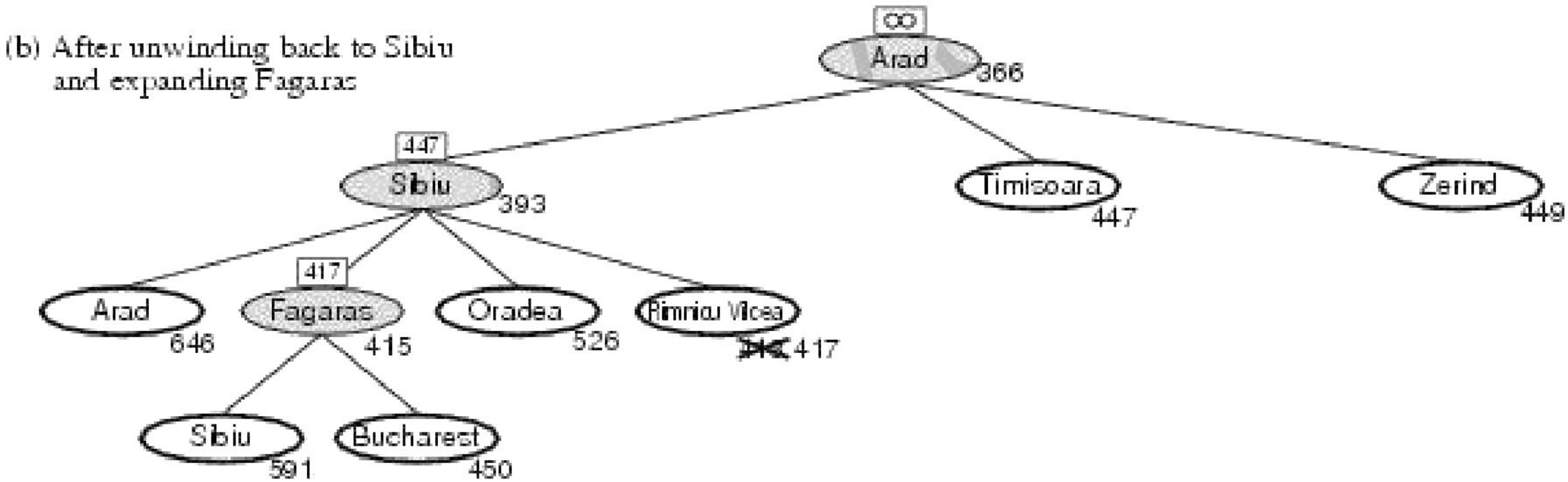
1. بهترین گره برگ و بهترین جانشین برای آن انتخاب شود.
2. اگر مقدار بهترین گره برگ از جانشین آن بیشتر شد، آنگاه به مسیر جانشین عقبگرد شود.
3. در حین عقبگرد، مقدار f(n) بروزرسانی شود.
4. گره جانشین بسط داده شود.

RBFS

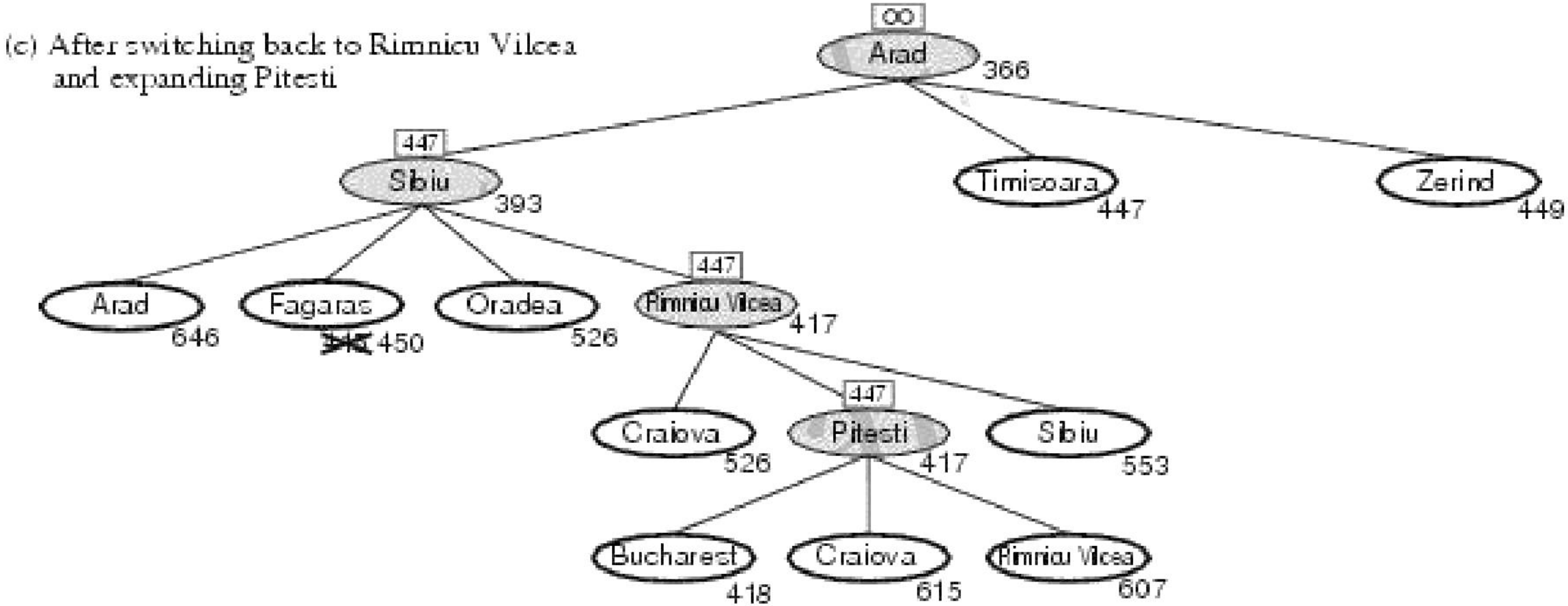
(a) After expanding Arad, Sibiu, Rimnicu Vilcea



RBFS


(b) After unwinding back to Sibiu
and expanding Fagaras
∞
Arad
366
447
Sibiu
393
Timisoara
447
Zerind
449
417
Arad
646
Fagaras
415
Oradea
526
Rimnicu Vilcea
417
Sibiu
591
Bucharest
450

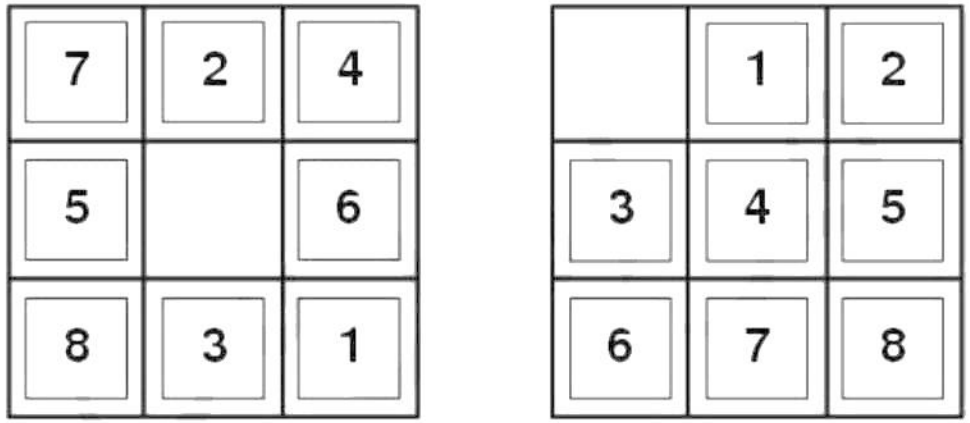
RBFS


(c) After switching back to Rimnicu Vilcea
and expanding Pitesti
∞
Arad
366
447
Sibiu
393
Timisoara
447
Zerind
449
Arad
646
Fagaras
450
Oradea
526
447
Rimnicu Vilcea
417
Craiova
526
447
Pitesti
417
Sibiu
553
Bucharest
418
Craiova
615
Rimnicu Vilcea
607

RBFS

- RBFS جستجوی به مراتب موثرتری از ID A* است.
- کماکان از تولید تعداد بسیار زیادی گره به دلیل **تغییر عقیده** رنج می برد.
- مانند A* اگر h(n) قابل پذیرش باشد، بهینه است.
- پیچیدگی حافظه o(bd) است.
- پیچیدگی زمانی به کیفیت تابع هیوریستیک و میزان تغییر عقیده بستگی دارد.

تابع هیوریستیک



Start State Goal State

- عمق جواب بهینه به طور متوسط ۲۲ است.
- فاکتور انشعاب به طور متوسط ۳ است.
- تعداد فضای حالت 3.1×10^{10} می‌باشد.
- تابع هیوریستیک خوب می‌تواند فضای حالت را به طرز چشمگیری کاهش دهد.

فهرست مطالب

- استراتژی‌های آگاهانه
  - جستجوی حریصانه
  - جستجوی A*
- کیفیت توابع هیوریستیک
- ابداع توابع هیوریستیک
- جستجوی محلی
  - تپه نوردی
  - SA
  - پرتوی محلی
  - ژنتیک

فهرست مطالب

- استراتژی‌های آگاهانه
  - جستجوی حریصانه
  - جستجوی A*
- کیفیت توابع هیوریستیک
- ابداع توابع هیوریستیک
- جستجوی محلی
  - تپه نوردی
  - SA
  - پرتوی محلی
  - ژنتیک

تابع هیوریستیک

- h_1 تعداد کاشی هایی که سرجای خود نمی‌باشند.

$h_1(s)=8$

- h_2 مجموع فاصله افقی – عمودی (منهتن) هر کاشی تا جای واقعی

$h_2(s)=3+1+2+2+2+3+3+2=18$

- h_3 مجموع فاصله افقی – عمودی – قطری هر کاشی تا جای واقعی

$h_3(s)=2+1+1+1+2+2+2+2=13$

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

کیفیت تابع هیوریستیک

- فاکتور انشعاب موثر (b*) : فاکتور انشعاب در درخت یکنواخت

$$N+1=1+b^*+(b^*)^2+\ldots+(b^*)^d$$

- بهترین مقدار برای b* چه مقداری است؟

کیفیت تابع هیوریستیک

	Search Cost			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

فهرست مطالب

- استراتژی‌های آگاهانه
  - جستجوی حریصانه
  - جستجوی A*
- کیفیت توابع هیوریستیک
- ابداع توابع هیوریستیک
- جستجوی محلی
  - تپه نوردی
  - SA
  - پرتوی محلی
  - ژنتیک

فهرست مطالب

- استراتژی‌های آگاهانه
  - جستجوی حریصانه
  - جستجوی A*
- کیفیت توابع هیوریستیک
- ابداع توابع هیوریستیک
- جستجوی محلی
  - تپه نوردی
  - SA
  - پرتوی محلی
  - ژنتیک

ابداع تابع هیوریستیک قابل پذیرش (۱)

- از طریق نسخه ساده شده از مساله (relax version)

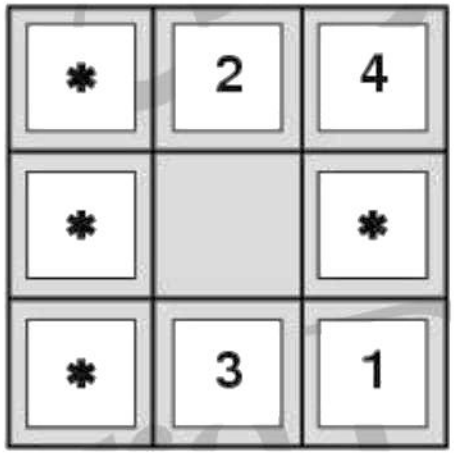
✓ h_1 هر کاشی می تواند به هرجایی منتقل شود.

✓ h_2 هر کاشی می تواند به هر خانه همسایه منتقل شود.

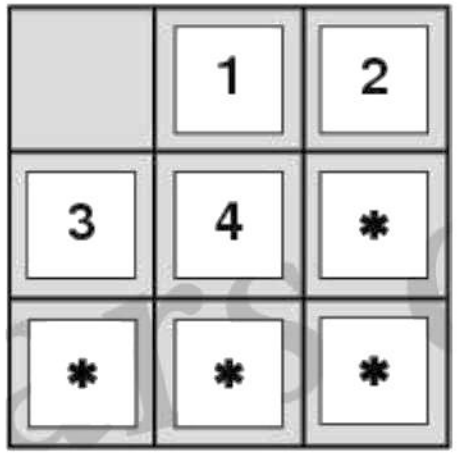
✓ `ABSolver` هزینه راه حل برای مکعب روبیک را تخمین می‌زند.

ابداع تابع هیوریستیک قابل پذیرش (۲)

- از طریق نسخه کوچتر از مساله (subproblem)



Start State



Goal State

ابداع تابع هیوریستیک قابل پذیرش (۳)

- از طریق یادگیری از تجربه (learning from experience)

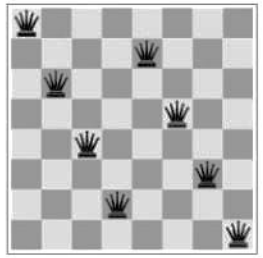
✓ تجربه : حل تعداد بسیار زیادی از مساله

فهرست مطالب

- استراتژی‌های آگاهانه
  - جستجوی حریصانه
  - جستجوی A*
- کیفیت توابع هیوریستیک
- ابداع توابع هیوریستیک
- جستجوی محلی
  - تپه نوردی
  - SA
  - پرتوی محلی
  - ژنتیک

فهرست مطالب

- استراتژی‌های آگاهانه
  - جستجوی حریصانه
  - جستجوی A*
- کیفیت توابع هیوریستیک
- ابداع توابع هیوریستیک
- جستجوی محلی
  - تپه نوردی
  - SA
  - پرتوی محلی
  - ژنتیک



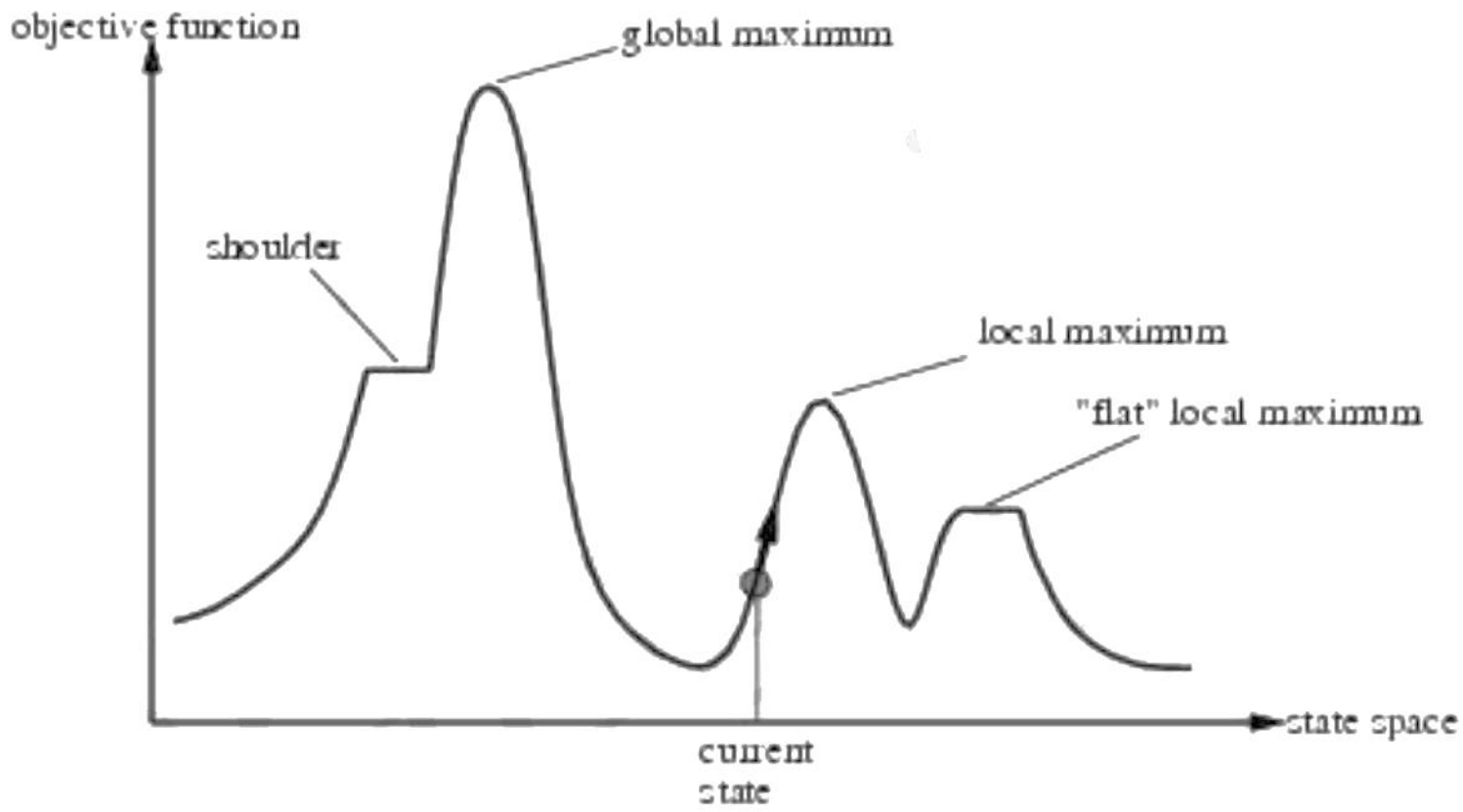
جستجوی محلی

- برای برخی از مسایل راه حل رسیدن تا هدف مهم نمی‌باشد بلکه خود حالت هدف مهم می‌باشد.
- برای اینگونه مسایل از جستجوی محلی استفاده می‌شود.
- جستجوی محلی: فقط حالت فعلی را ذخیره نموده و به بهترین همسایه حرکت می‌کند.
- حافظه بسیار کم و امکان پیدا کردن راه حل در مسایل با تعداد حالات بسیار زیاد، از مزایای جستجوهای محلی است.

الگوریتم تپه نوردی

- به طور متناوب در جهت بهبود حرکت می‌کند. زمانی که به قله برسد متوقف می‌شود.
- تپه نوردی به آینده گره‌های برگ توجه نمی‌کند، به همین دلیل به الگوریتم جستجوی محلی **حریصانه** هم مشهور است.
- تپه نوردی در صورتیکه بیشتر از یک گره بهترین وجود داشته باشد، بهترین گره را به صورت تصادفی انتخاب می‌کند.

الگوریتم تپه نوردی

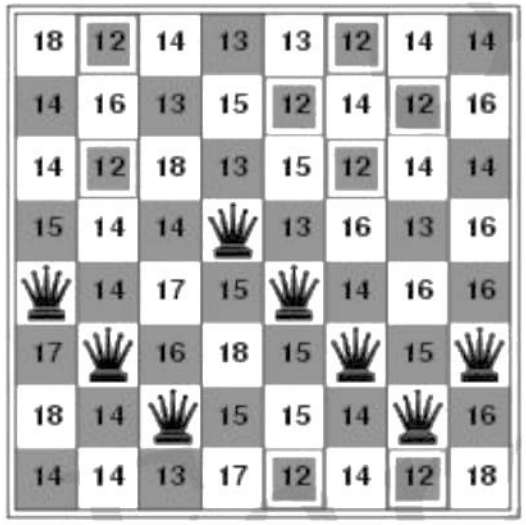


الگوریتم تپه نوردی

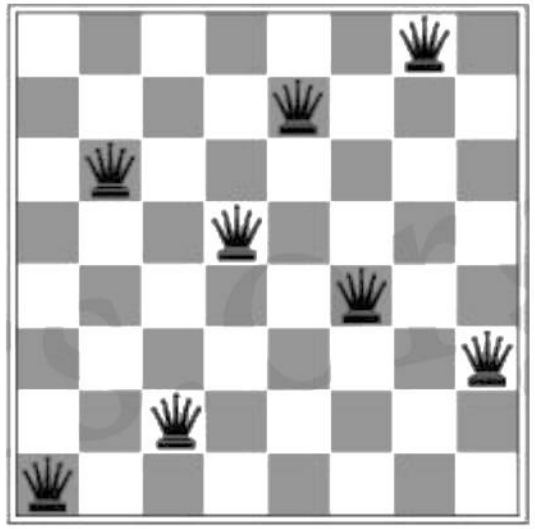
```
function HILL-CLIMBING( problem) return a state that is a local maximum
    input: problem, a problem
    local variables: current, a node.
                     neighbor, a node.

    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
```

الگوریتم تپه نوردی

- h(n) : تعداد جفت وزیرهایی که همدیگر را به صورت مستقیم یا غیر مستقیم تهدید می‌کنند.

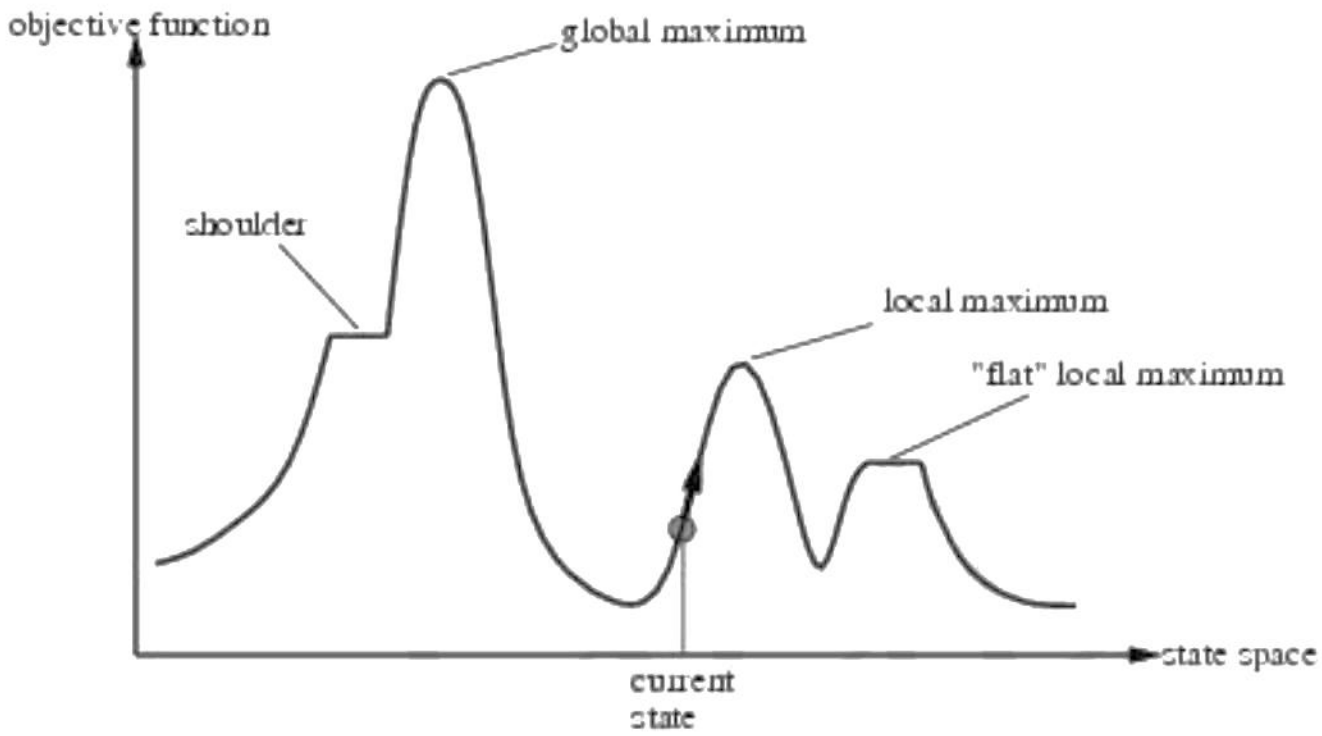


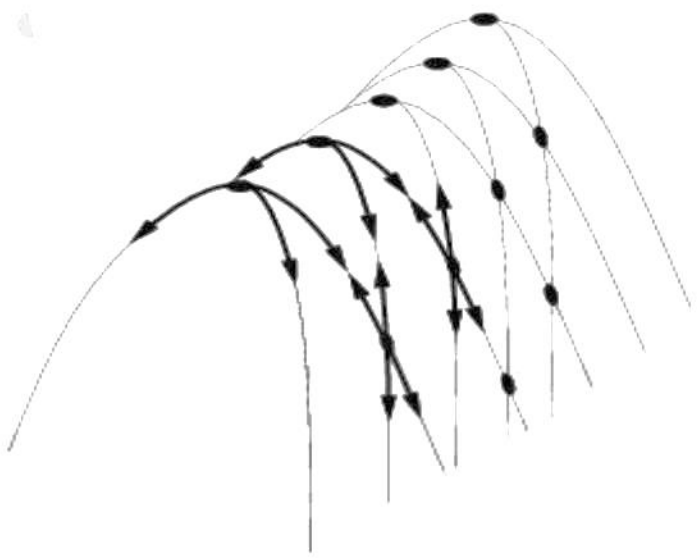
a)h=3+4+2+3+2+2+1+0=17



b)h=1

الگوریتم تپه نوردی





انواع الگوریتم تپه نوردی

- تپه نوردی غیر قطعی (Stochastic hill-climbing):

✓ در بین حرکت های رو به بالا یکی به صورت تصادفی انتخاب شود. البته احتمال انتخاب با شیب متناسب است.

- تپه نوردی با انتخاب اولین گزینه (First-choice hill-climbing):

✓ گره‌ها تا حصول یک گره بهتر بسط داده می‌شوند.

- تپه نوردی تصادفی (Random-restart hill-climbing):

✓ از حالت شروع مجدد تصادفی تا حصول جواب مجدداً شروع خواهد نمود.

Simulated annealing

- اجتناب از گیر کردن در بیشینه‌های محلی با اجازه دادن به انجام حرکت های فرعی(نا مناسب)، که در حین گذشت زمان احتمال و تعداد آن کاهش می‌یابد.
- پیشینه الگویتم به علوم متالورژی بر می‌گردد.
- مثال: Bouncing ball analogy

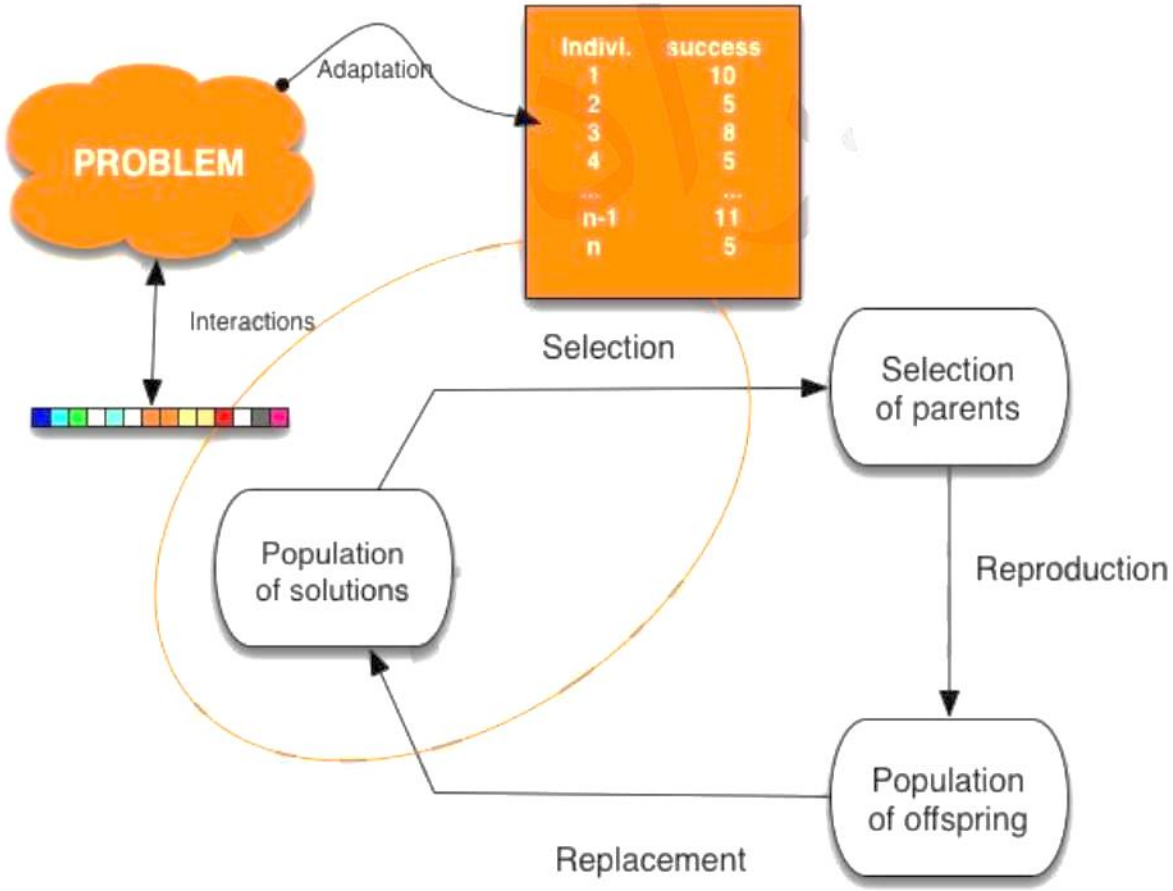
Simulated annealing

```
function SIMULATED-ANNEALING( problem, schedule) return a solution state
    input: problem, a problem
           schedule, a mapping from time to temperature
    local variables: current, a node.
                     next, a node.
                     T, a "temperature" controlling the probability of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] - VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^(ΔE/T)
```

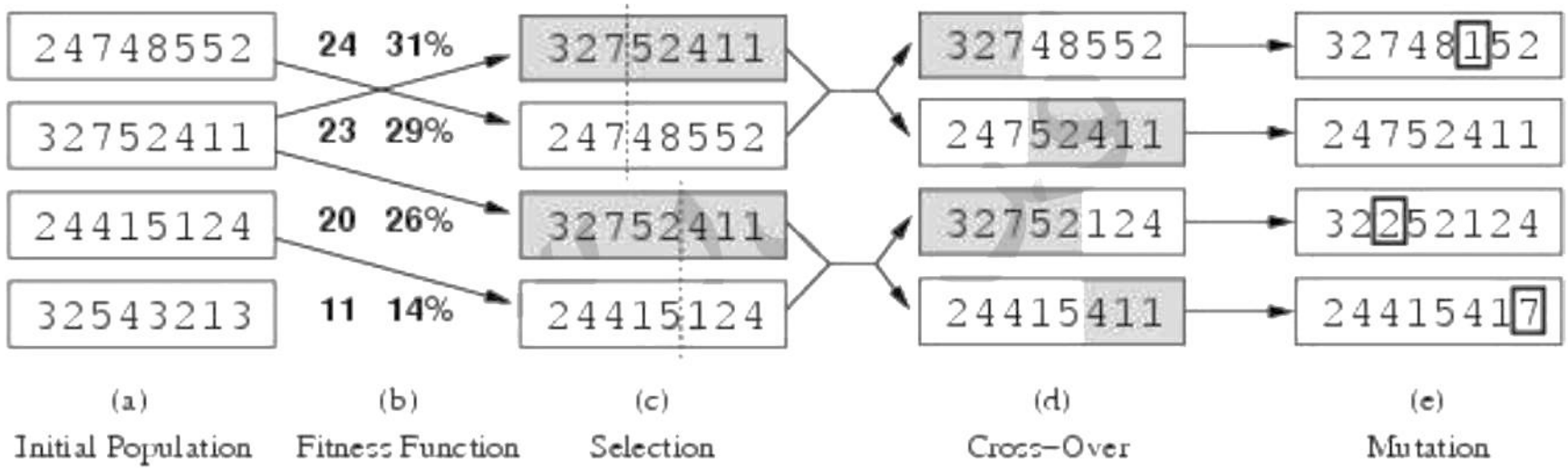
Local beam search

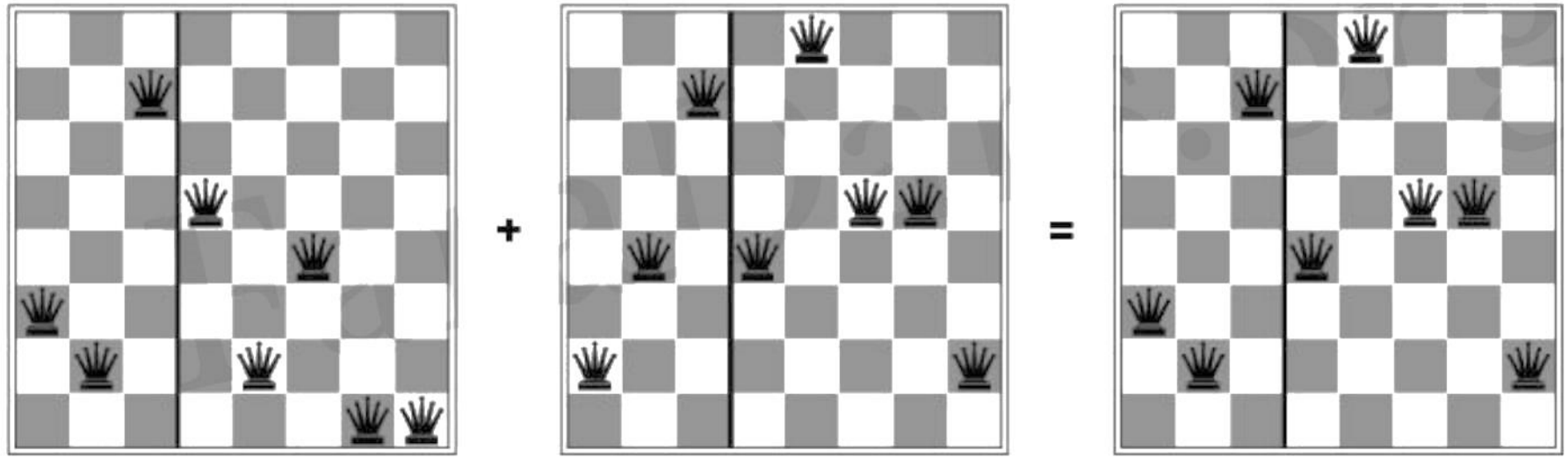
- از k حالت شروع به جای یک حالت شروع بهره می‌برد.
  - حالت شروع: k حالت تصادفی
  - حالت بعدی: انتخاب k تا بهترین حالت از بین تمام برگ‌ها
  - حالت خاتمه: پیدا شدن هدف یا بررسی تمام حالات
- **تفاوت** با تپه نوردی با شروع مجدد تصادفی این است که اطلاعات به اشتراک گذاشته می‌شود.
- ممکن است الگوریتم از **عدم تنوع** کافی برخوردار باشد.

Genetic algorithms



Genetic algorithms





فهرست مطالب

- استراتژی‌های آگاهانه
  - جستجوی حریصانه
  - جستجوی A*
- کیفیت توابع هیوریستیک
- ابداع توابع هیوریستیک
- جستجوی محلی
  - تپه نوردی
  - SA
  - پرتوی محلی
  - ژنتیک

فهرست مطالب

- استراتژی‌های آگاهانه
  - جستجوی حریصانه
  - جستجوی A*
- کیفیت توابع هیوریستیک
- ابداع توابع هیوریستیک
- جستجوی محلی
  - تپه نوردی
  - SA
  - پرتوی محلی
  - ژنتیک