دانشگاه کردستان
University of Kurdistan

# فصل چهارم

# جستجوی محلی و بهینه‌سازی

**مدرس: سعدون عزیزی**

گروه مهندسی کامپیوتر

پاییز ۹۶

# رئوس مطالب

- جستجوی تپه‌نوردی (Hill-climbing search)
- جستجوی تبرید شبیه‌سازی شده (Simulated Annealing search)
- جستجوی پرتو محلی (Local Beam search)
- الگوریتم‌های ژنتیک (Genetic Algorithms)
- جستجوی محلی در فضای پیوسته (Searching in continuous spaces)

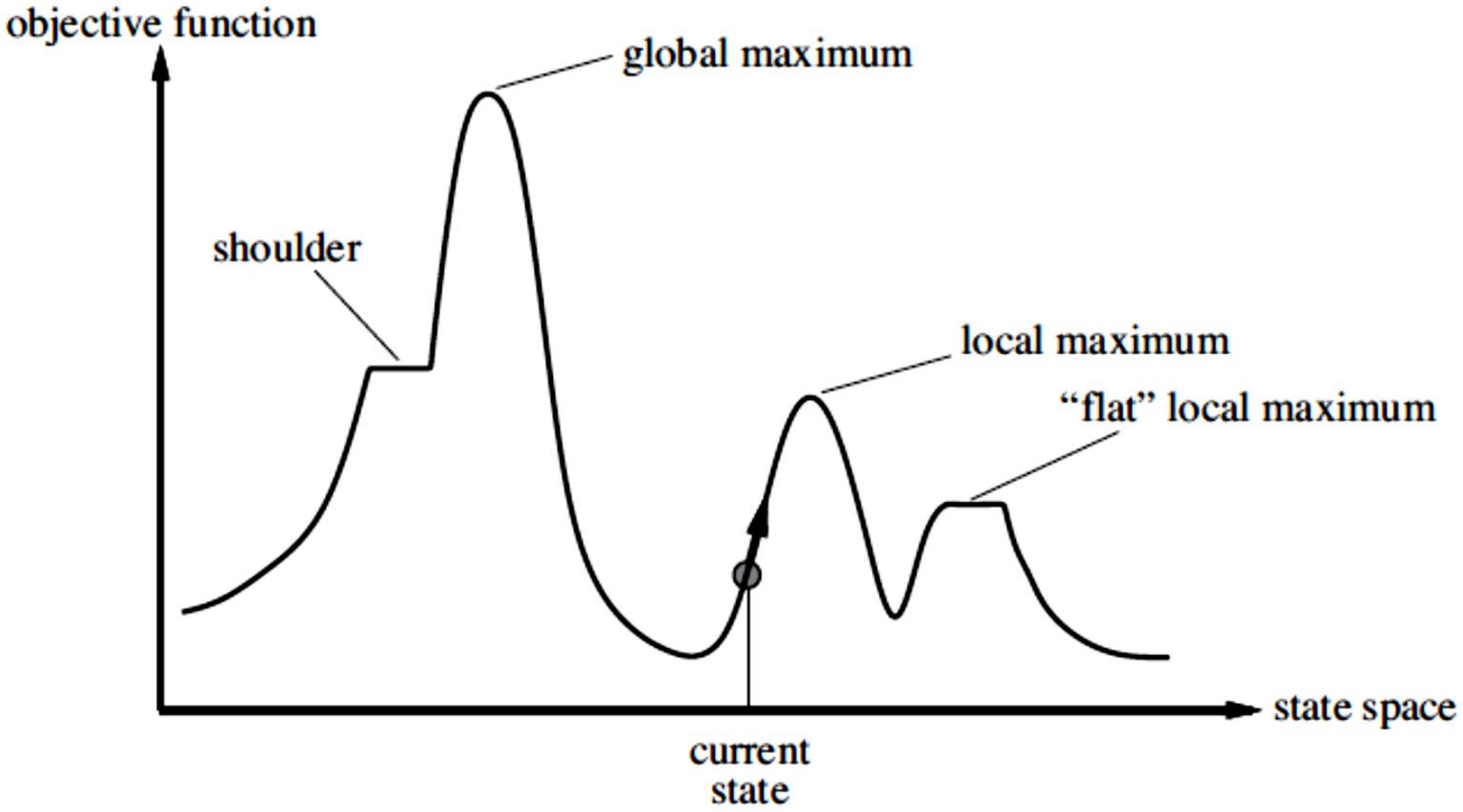
# جستجوی سیستماتیک در مقابل جستجوی محلی

- **جستجوی سیستماتیک:** مسیر رسیدن به هدف مهم است
  - اثبات قضیه
  - پیدا کردن مسیر
  - معمای ۸-تایی
  - شطرنج
- **جستجوی محلی:** مسیر تا هدف مهم نیست بلکه خود حالت هدف مهم است
  - ۸ وزیر
  - فروشنده دوره گرد (TSP)
  - طراحی مدارهای مجتمع (VLSI)
  - زمانبندی کارها
  - برنامه‌نویسی خودکار

# مزیت‌های الگوریتم‌های جستجوی محلی

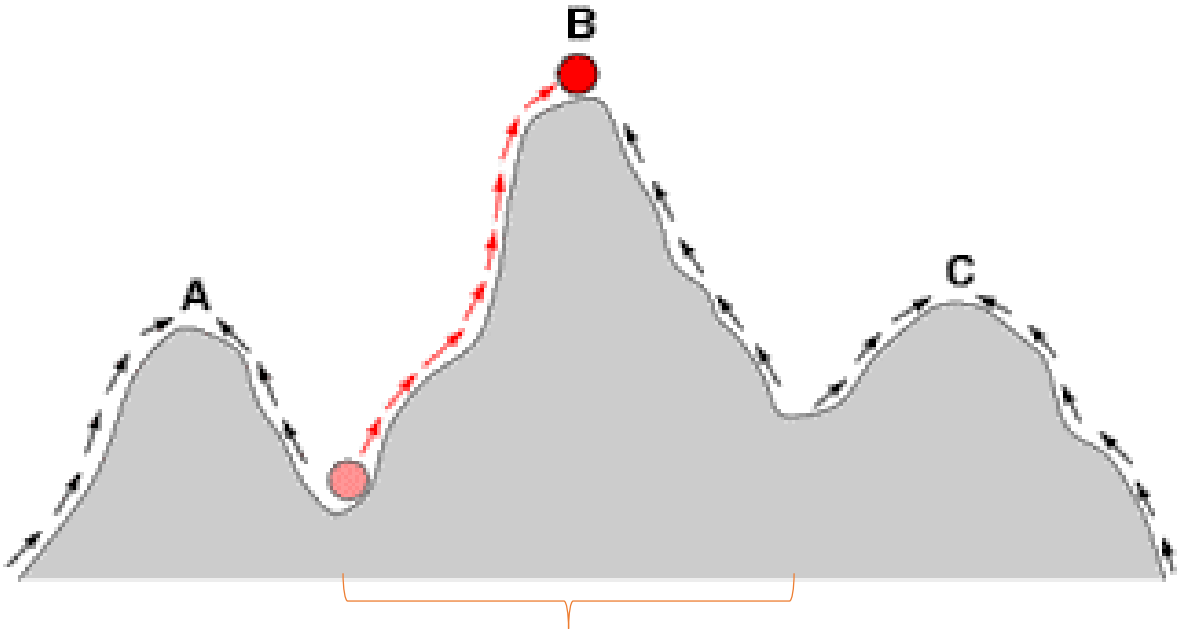
- از حافظه بسیار کمی استفاده می‌کنند (معمولا یک مقدار ثابت)
- اغلب می‌توانند در فضاهای حالت بزرگ یا نامتناهی (پیوسته) که برای آن الگوریتم‌های سیستماتیک مناسب نیستند، راه‌حل‌های معقولی بیابند
- این الگوریتم‌های علاوه بر یافتن هدف، برای حل مسائل بهینه‌سازی نیز بسیار سودمند هستند
  - در مسائل بهینه‌سازی، هدف پیدا کردن بهترین حالت بر اساس تابع هدف است

# دورنمایی از فضای حالت



# جستجوی تپه‌نوردی

- **ایده اصلی الگوریتم:** همیشه به سمت بالای تپه حرکت کن؛ تا زمانی که هیچ همسایه‌ای مقدار بیشتری از مقدار فعلی نداشته باشد.




**اگر حالت شروع در این محدوده قرار بگیرد، جواب بهینه بدست می‌آید**

# جستجوی تپه‌نوردی (الگوریتم)

```
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 تپه‌نوردی

- الگوریتم جستجوی تپه نوردی فقط از یک حلقه تشکیل می شود که مدام در جهت افزایش مقدار حرکت می‌کند (یعنی به سمت بالای تپه).
- الگوریتم زمانی خاتمه می‌یابد که به قلّه‌ای برسد که در آنجا هیچ همسایه‌ای مقدار بیشتری نداشته باشد.
- این الگوریتم، درخت جستجو را نگهداری نمی‌کند. بنابراین، در ساختمان داده‌ی گره فعلی تنها نیاز به ثبت حالت و مقدار تابع هدف آن دارد.
- جستجوی تپه نوردی، فراتر از همسایه‌های مجاور حالت فعلی را نگاه نمی‌کند.
- تپه‌نوردی را گاهی جستجوی حریصانه محلی می‌نامند.

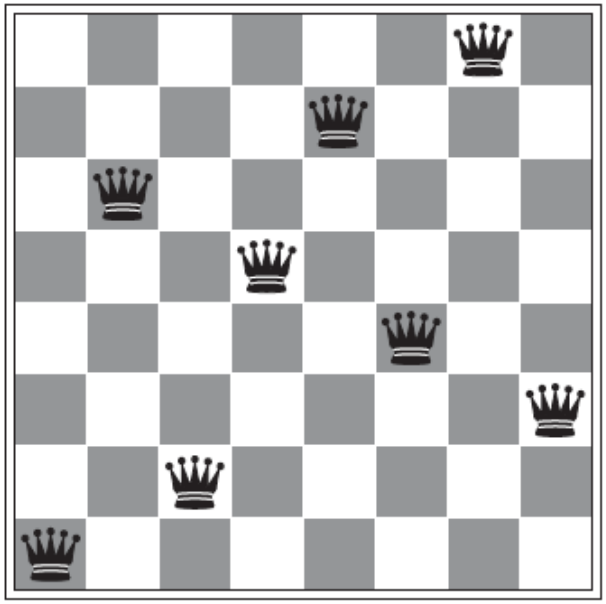
# جستجوی تپه‌نوردی (مسئله ۸ وزیر)

- حالات: ۸ وزیر روی صفحه، یک وزیر در هر ستون ($8^8 \approx 17$ million)
- تابع پسین: حرکت هر وزیر در همان ستون ($8 \times 7 = 56$)
- تابع هزینه h(s): تعداد زوج وزیرهایی که به صورت مستقیم یا غیرمستقیم یکدیگر را تهدید می‌کنند
- کمینه سراسری: h(s)=0

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 18 | 12 | 14 | 13 | 13 | 12 | 14 | 14 |
| 14 | 16 | 13 | 15 | 12 | 14 | 12 | 16 |
| 14 | 12 | 18 | 13 | 15 | 12 | 14 | 14 |
| 15 | 14 | 14 | ♛ | 13 | 16 | 13 | 16 |
| ♛ | 14 | 17 | 15 | ♛ | 14 | 16 | 16 |
| 17 | ♛ | 16 | 18 | 15 | ♛ | 15 | ♛ |
| 18 | 14 | ♛ | 15 | 15 | 14 | ♛ | 16 |
| 14 | 14 | 13 | 17 | 12 | 14 | 12 | 18 |

**h(s)=17**

**بعد از ۵ گام**



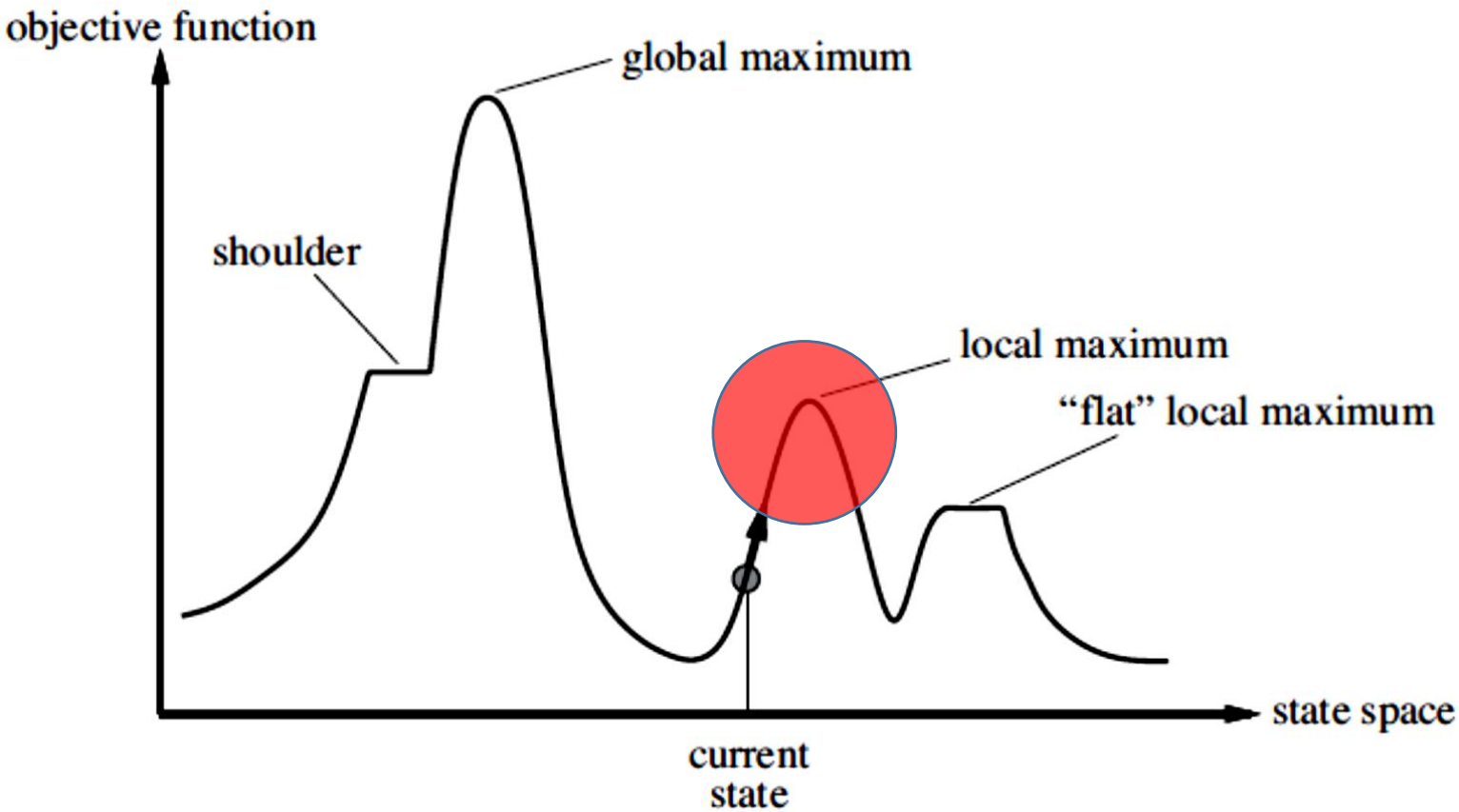
**h(s)=1**

# دلایل گیر کردن جستجوی تپه‌نوردی

- بیشینه یا کمینه محلی (Local Maxima)
- سطوح صاف یا فلات‌ها (Plateau-Flat)
- تیغه‌ها (Ridges)

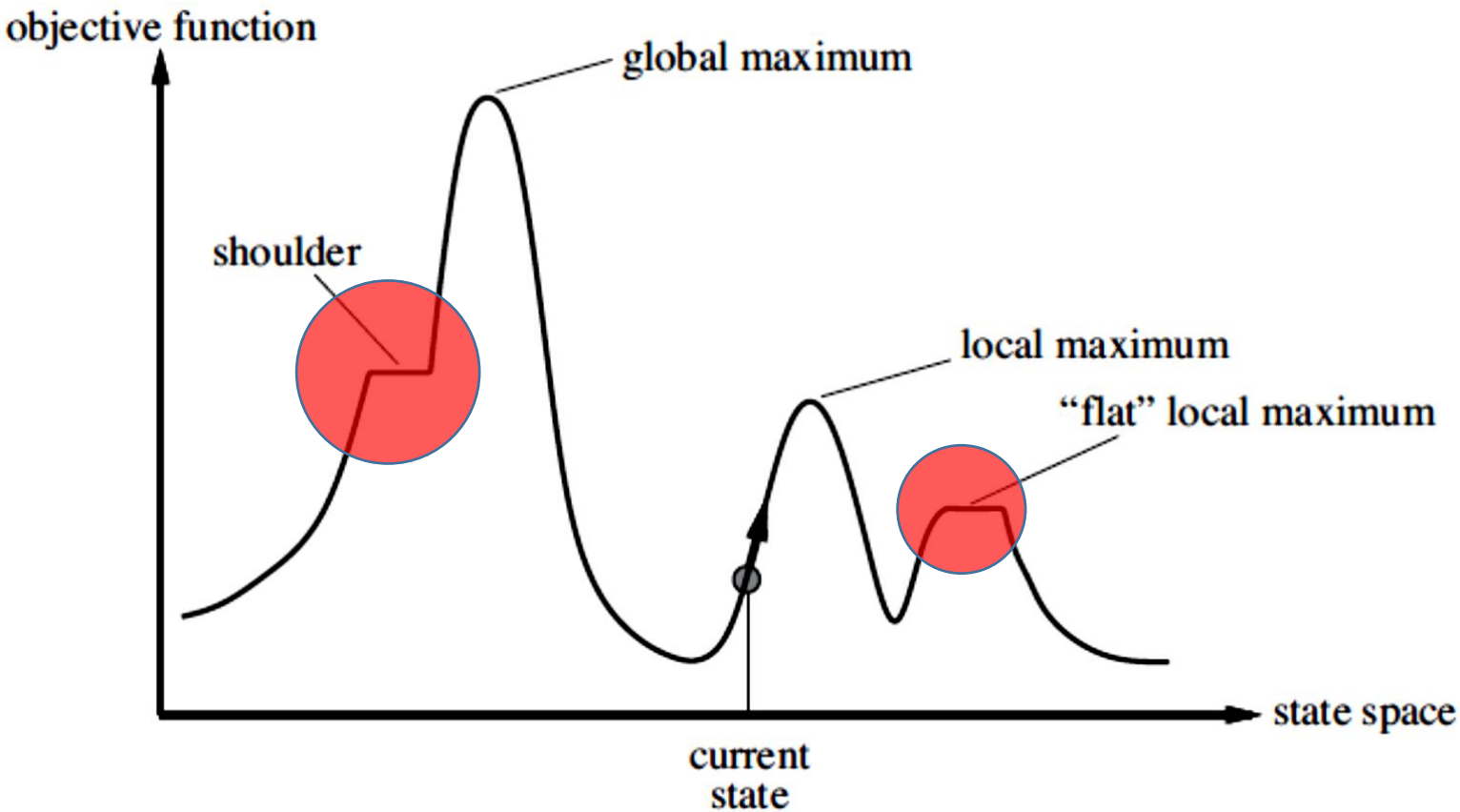
# بیشینه یا کمینه محلی (Local Maxima)

- یک بیشینه محلی، قله‌ای است که از همه حالت‌های همسایه‌اش بلندتر است ولی از بیشینه سراسری کوتاه‌تر است



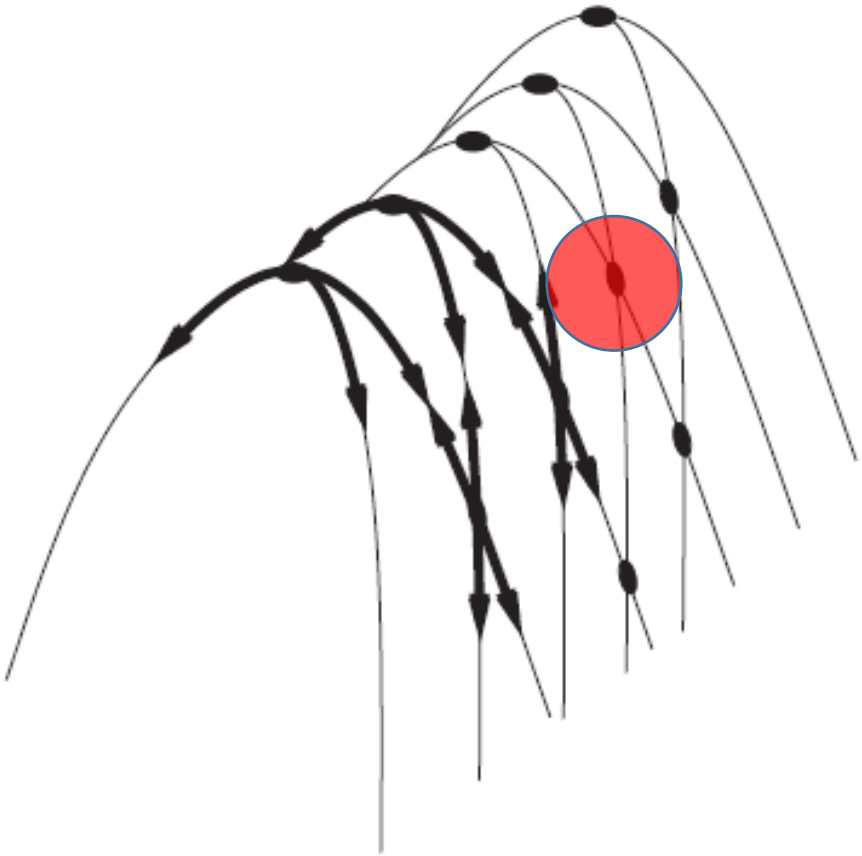
# سطوح صاف یا فلات‌ها (Plateau-Flat)

- ناحیه‌ای از دورنمای فضای حالت است که در آن تابع ارزیاب ثابت است
  - بیشینه محلی مسطح
  - شانه



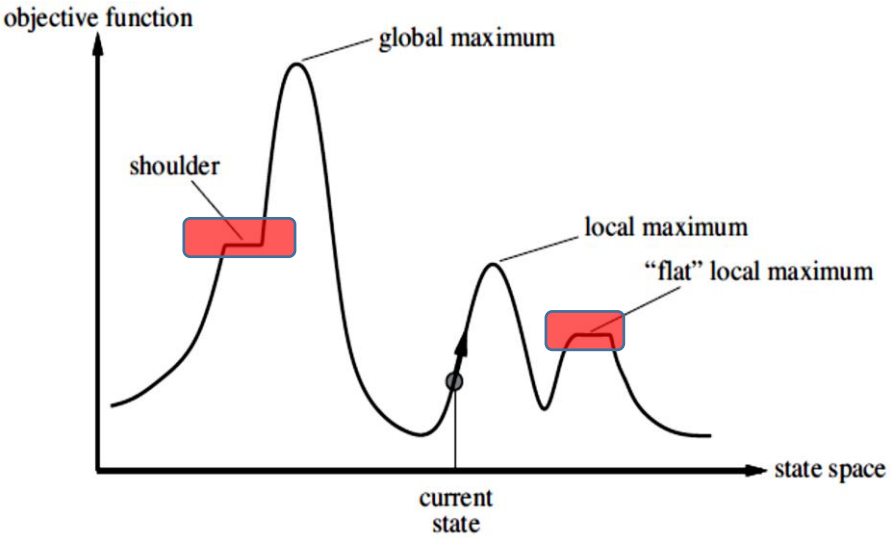
# تیغه‌ها (Ridges)

- دنباله‌ای از بیشینه‌های محلی که گذشتن از آنها برای الگوریتم‌های جستجوی محلی خریصانه بسیار دشوار است



# حل مشکل سطوح صاف

- حرکت‌های از کناره (sideway moves):
  - انجام چند حرکت به امید اینکه سطح صاف از نوع شانه باشد
  - مسئله: گیر کردن در حلقه بی‌نهایت اگر بیشینه محلی مسطح باشد
  - راه‌حل: محدود کردن تعداد حرکت‌های از کناره پی‌درپی
  - در مسئله ۸ وزیر با محدودیت ۱۰۰ حرکت از کناره پی‌درپی درصد مسائل حل شده را از ۱۹٪ به ۹۴٪ می‌رساند!



# انواع مختلف جستجوی تپه‌نوردی

- تپه‌نوردی تصادفی (stochastic hill-climbing)
- تپه‌نوردی اولین انتخاب (first-choice hill-climbing)
- تپه‌نوردی با شروع مجدد تصادفی (random restart hill-climbing)

# تپه‌نوردی تصادفی

- به صورت تصادفی از میان حرکت‌های رو به بالا یکی را انتخاب می کند.
- احتمال این انتخاب می‌تواند براساس شیب حرکت‌های رو به بالا تغییر کند.
- این روش معمولاً کندتر از تندترین شیب به نتیجه می‌رسد، اما در بعضی از دورنماهای حالت، بهترین راه حل را پیدا می کند.

# تپه‌نوردی اولین انتخاب

- در این روش به صورت تصادفی پسین‌های یک حالت تولید می‌شوند تا زمانی که پسینی تولید شود که از حالت فعلی بهتر باشد.
- این رویکرد وقتی مناسب است که یک حالت دارای تعداد زیادی (مثلاً هزار) پسین باشد.

# تپه‌نوردی با شروع مجدد تصادفی

- **ایده اصلی الگوریتم:** "در ناامیدی بسی امید است"
- در این روش یک مجموعه جستجوی تپه‌نوردی را با حالت‌های شروع تصادفی اجرا می کند و هنگامی که یک هدف پیدا شد متوقف می‌شود.
- این روش با احتمال نزدیک به یک، کامل می باشد؛
  - برای این موضوع دلیل ساده‌ای وجود دارد: بالاخره یک حالت هدف به عنوان حالت شروع تولید خواهد شد
- اگر موفقیت هر جستجوی تپه‌نوردی p باشد آنگاه مقدار متوسط تعداد شروع‌های مجدد لازم 1/p است (چرا؟)

# تپه‌نوردی با شروع مجدد تصادفی (مثال)

- در مسئله ۸ وزیر داریم:
  - $p \approx 0.14$
  - تعداد تکرارها (شروع مجدد تصادفی): $1/p \approx 7$
  - 6 شکست و 1 موفقیت
  - مقدار متوسط تعداد گام‌ها: 22 گام
  - با استفاده از حرکت‌های از کناره داریم:
    - تعداد تکرارها: $\frac{1}{0.94} \approx 1.06$
    - مقدار متوسط تعداد گام‌ها: حدود 26 گام
    - برای سه میلیون وزیر در کمتر از یک دقیقه راه‌حل پیدا می‌شود

# تاثیر دورنمای مسئله روی کارایی تپه‌نوردی

- اگر تعداد بیشینه محلی و فلات کم باشد، تپه‌نوردی با شروع مجدد تصادفی یک راه‌حل خوب را خیلی سریع پیدا می‌کند
- بسیاری از مسائل دنیای واقعی شبیه خانواده جوجه‌تیغی هستند!
- مسائل NP-hard معمولا دارای تعداد نمایی بیشنه محلی هستند
- یک بیشینه محلی نسبتاً خوب را اغلب می‌توان پس از تعداد کمی شروع مجدد پیدا کرد

# تبرید شبیه‌سازی شده (Simulated Annealing)

- **تپه‌نوری:** حرکت به سمت یک حالت بهتر
  - کارآمد است اما کامل نیست (گیر کردن در بیشینه محلی)
- **گشت تصادفی (Random Walk):** حرکت به سمت یک پسین تصادفی
  - به صورت مجانبی کامل است اما به شدت ناکارآمد است
- **ایده:** فرار از بیشینه محلی با اجازه دادن به انجام چند حرکت بد در ابتدا و سپس به تدریج از تعداد آنها کاسته شود
  - در شروع کار استفاده از گشت تصادفی (مشابه حرارت بسیار بالا در فلزات و شیشه‌ها) و سپس به تدریج استفاده از تپه نوردی (مشابه خنک‌سازی تدریجی فلزات و شیشه‌ها )

# تبرید شبیه‌سازی شده (الگوریتم)

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"

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

T: درجه حرارت

t: زمان

schedule: تابع کاهش دما نسبت به زمان

# تبرید شبیه‌سازی شده

- SA به جای انتخاب بهترین حرکت، یک حرکت تصادفی را انتخاب می‌کند. اگر این حرکت، شرایط را بهبود ببخشد، همیشه پذیرفته می شود. در غیر این صورت، الگوریتم این حرکت را با احتمالی کمتر از ۱ می پذیرد.
- این احتمال بر اساس میزان بدی حرکت به صورت نمایی کاهش می‌یابد
- این احتمال با کاهش دمای T نیز کاهش می یابد: احتمال مجاز شمردن حرکتهای بد در آغاز که مقدار T بالا است، زیاد خواهد بود؛ اما با کاهش T مقدار احتمال کاهش می‌یابد
- می‌توان ثابت کرد که اگر زمانبندی مقدار T را به اندازه کافی آهسته کاهش دهد، SA یک بهینه سراسری را با احتمال نزدیک به ۱ پیدا خواهد کرد

# جستجوی پرتو محلی (Local Beam search)

- نگهداری تنها یک گره در حافظه یک واکنش افراطی نسبت به محدودیت حافظه است (کاری که تپه نوردی و SA انجام می‌دهند)
- در الگوریتم **جسجوی پرتو محلی** به جای یک حالت، اطلاعات k حالت نگهداری می‌شود
- در هر گام، تمامی پسین‌های هر k حالت تولید می‌شوند
- هر کدام از آنها که هدف باشد، الگوریتم متوقف می‌شود
- در غیر این صورت، با انتخاب بهترین k پسین از کل لیست، این عملیات را تکرار می‌کند

# جستجوی پرتو محلی (الگوریتم)

Start with *k* randomly generated states

Loop:

All the successors of all k states are generated

**if** any one is a goal state **then** stop

**else** select the k best successors from the complete list of successors and repeat.

# جستجوی پرتو محلی

- این الگوریتم با اجرای موازی تپه نوردی با k شروع مجدد تصادفی متفاوت است (چرا؟)
- الگوریتم جستجوی محلی بعد از چند تکرار به سرعت جستجوهای خود را به جایی منتقل می‌کند که بیشترین پیشرفت صورت می‌گیرد
  - اگر یک حالت پسین خوبی تولید کند و k-1 حالت دیگر همگی پسینهای بدی تولید کنند، حالت اول به بقیه اعلام می‌کند: بیایید اینجا، بهترین‌ها اینجا هستند!
- **جستجوی پرتو تصادفی:** انتخاب k پسین به صورت تصادفی به طوری که احتمال انتخابشان یک تابع افزایشی از مقدار تابع هدفشان (objective function) باشد

# الگوریتم‌های ژنتیک (Genetic Algorithms)

- یک الگوریتم ژنتیک (GA) گونه‌ای از جستجوی پرتو تصادفی است که حالت‌های پسین به جای تغییر یک حالت، از ترکیب دو حالت والد به وجود می‌آید
- همانند جستجوی پرتو محلی، GA با یک مجموعه از k حالت تصادفی، که **جمعیت** نامیده می‌شود، شروع می‌شود

# الگوریتم‌های ژنتیک

- یک حالت (یا یک راه‌حل) به صورت یک رشته از الفبای محدود بیان می‌شود.
  - مانند یک کروموزوم که شامل مجموعه‌ای از ژن‌ها است
- k حالت که به صورت تصادفی تولید می‌شوند (جمعیت اولیه)
- تابع ارزیابی برای ارزیابی حالت‌ها (تابع برازش – fitness function)
- ترکیب (cross-over) دو حالت والد و تشکیل فرزندان
- تغییر یک حالت با انجام عمل جهش (mutation) روی آن
- نسل بعدی حالت‌ها با انتخاب (بر اساس تابع برازش)، ترکیب و جهش تولید می‌شوند

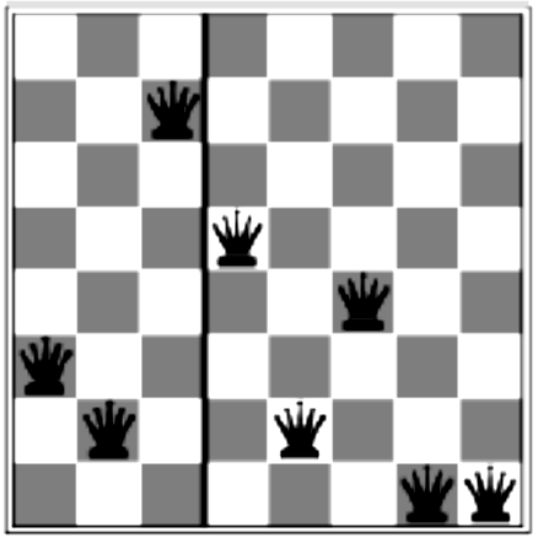
# کروموزوم و تابع برازش: مسئله ۸ وزیر

- کروموزوم:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ۲ | ۴ | ۷ | ۴ | ۸ | ۵ | ۵ | ۲ |

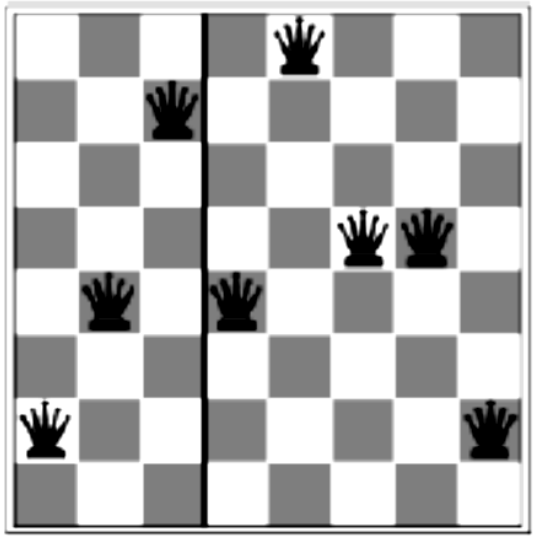
- **تابع برازش:** تعداد وزیرهایی که یکدیگر را تهدید نمی‌کنند
  - برای این مثال، تابع برازش مقدار ۲۴ را بر می‌گرداند

# ترکیب: مسئله ۸ وزیر



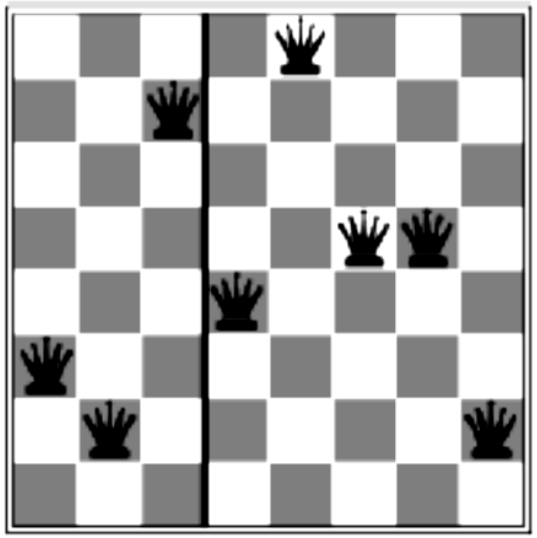
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 7 | 2 | 4 | 7 | 5 | 8 | 8 |

+



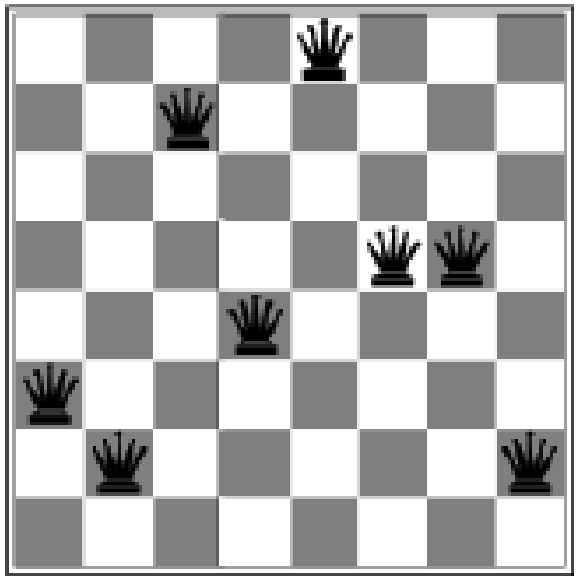
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 5 | 2 | 5 | 1 | 4 | 4 | 7 |

=



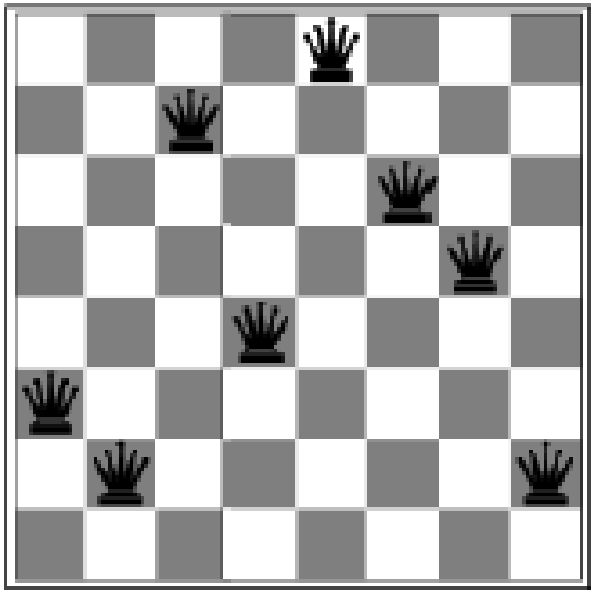
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 7 | 2 | 5 | 1 | 4 | 4 | 7 |

# جهش: مسئله ۸ وزیر



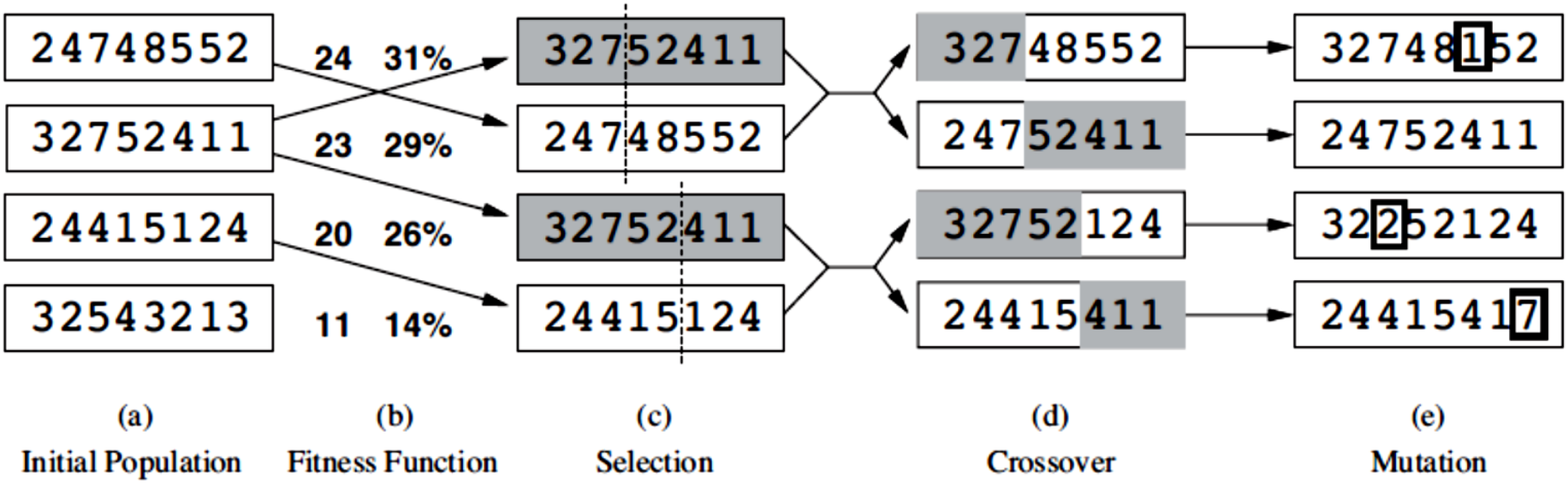
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 7 | 2 | 5 | 1 | 4 | 4 | 7 |

جهش



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 7 | 2 | 5 | 1 | 3 | 4 | 7 |

# مراحل الگوریتم ژنتیک: مسئله ۸ وزیر



# شبه کد الگوریتم 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
     for i = 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))
```

# جستجوی محلی در فضای پیوسته

- هیچکدام از الگوریتم‌هایی که تاکنون بیان شد، نمی‌توانند از پس فضاهای پیوسته برآیند
  - در اکثرموارد، تابع پسین تعداد نامتناهی حالت بر می‌گرداند
- فرض کنید بخواهیم سه فرودگاه جدید در رومانی بسازیم به طوری که مجموع مجذور فواصل تمام شهرهای روی نقشه از نزدیک‌ترین فرودگاه به آن کمینه شود
  - در این صورت فضای حالت توسط مختصات این فرودگاه‌ها بیان می‌شود: $(x_1 , y_1)$ و $(x_2 , y_2)$ و $(x_3 , y_3)$
- یک روش برای اجتناب از مسائل پیوسته، گسسته‌سازی همسایگی هر حالت است
  - برای مثال، می‌توان مکان یک فرودگاه را در یکی از جهات x یا y به اندازه مقدار ثابت $\pm \delta$ جابجا کرد (۱۲ پسین به ازای هر حالت)
- روش دوم استفاده از بهینه‌سازی در فضای پیوسته است (Gradient ascent)

# عامل‌های جستجوی برخط و محیط‌های ناشناخته

مطالعه بر عهده دانشجو...

(صفحه 147 تا 153 کتاب مرجع)