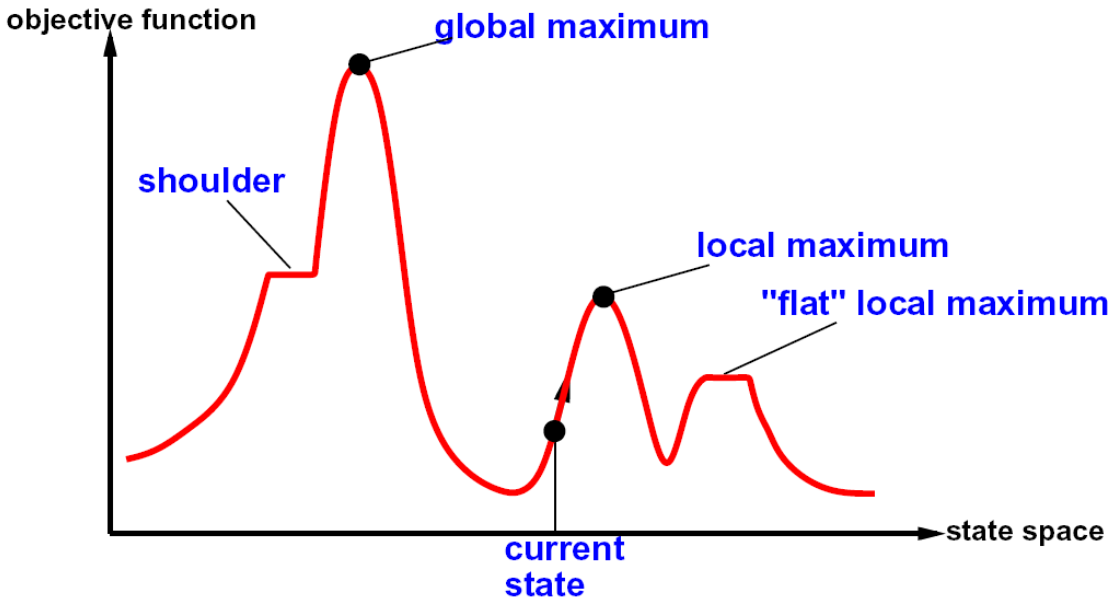
جستجوی محلی
(Local Search)

1. الگوریتم تپه نوردی (Hill Climbing)
2. الگوریتم سردسازی شبیه‌سازی شده (Simulated Annealing)
3. الگوریتم گرادیان کاهشی (Gradient Descent)
4. الگوریتم ژنتیک (Genetic Algorithm)

الگوریتم‌های جستجوی محلی

- مناسب برای مسائلی با فضای جستجوی بسیار بزرگ که نمی‌توان همه‌ی حالت‌ها را بررسی کرد.
- این مسائل با استفاده از تابع هزینه فرموله می‌شوند؛ هدف، ماکسیمم یا مینیمم کردن این تابع است.
- در این روش‌ها، هر state یک جواب کامل ممکن است.
- الگوریتم از یک جواب شروع می‌کند و با تغییرات محلی (local changes) به جواب‌های بهتر می‌رسد.
- نیازی به نگه داشتن مسیر طی‌شده نیست؛ فقط جواب فعلی و بهبودش مهم است.

الگوریتم تپه‌نوردی (Hill Climbing)



❑الگوریتم تپه‌نوردی:

- به طور دلخواه از یک نقطه اولیه شروع کن.
- تا زمانی که وضعیت بهتری در همسایه‌ها وجود دارد، جلو برو.
- اگر هیچ همسایه‌ای بهتر از حالت فعلی نبود، متوقف شو

❑مشکلات این روش:

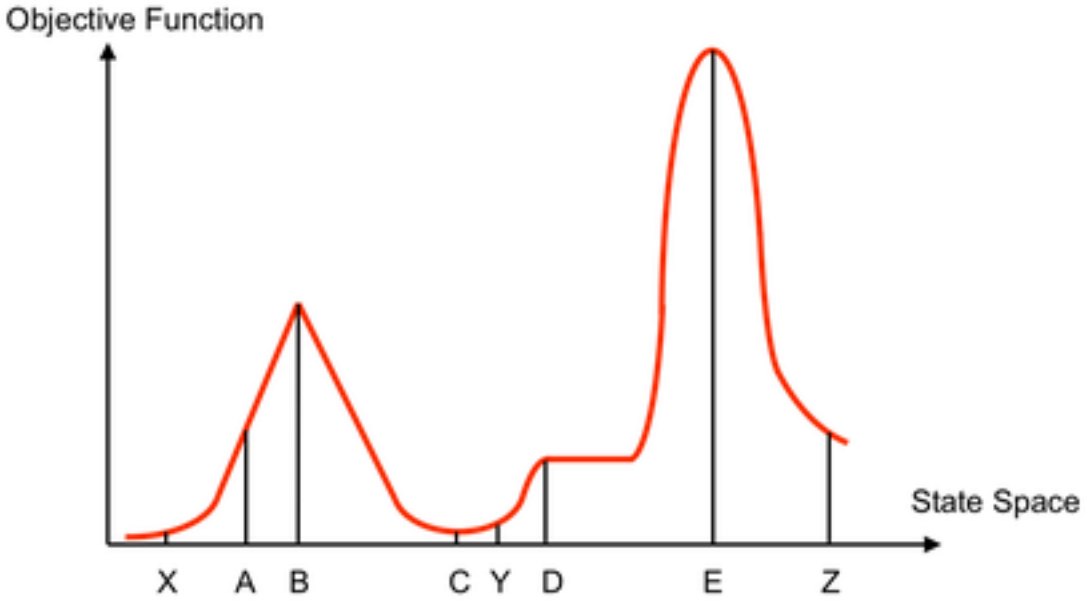
- ممکن است در بهینه‌های محلی (local optima) گیر کند
- تضمینی برای یافتن بهترین جواب (بهینه‌ی جهانی) وجود ندارد

❑مزایای این روش:

- ساده و سریع است.
- حافظه‌ی کمی مصرف می‌کند.

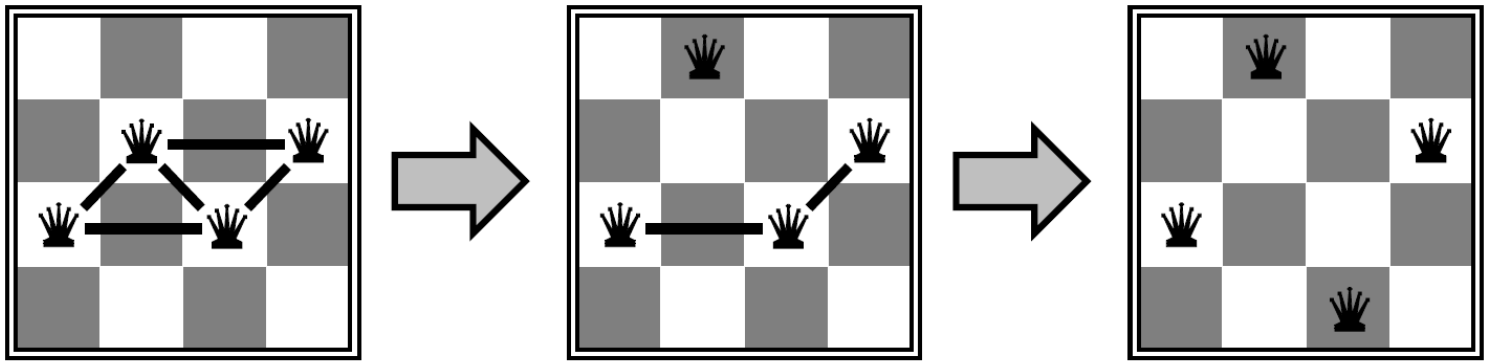
✓ اگر الگوریتم در نقطه‌ای گیر کند، می‌توان با Random Restart دوباره از یک وضعیت جدید شروع کرد

الگوریتم تپه‌نوردی



- اگر از X شروع شود، به کجا می‌رسد؟ B
- اگر از Y شروع شود، به کجا می‌رسد؟ D
- اگر از Z شروع شود، به کجا می‌رسد؟ E

مثال: N- وزیر



- **هدف:** هیچ دو وزیری نباید همدیگر را تهدید کنند.
- **حالت‌ها:** ۴ وزیر در ۴ ستون ($۴^۴$ = ۲۵۶ حالت)
- **عملگرها:** جابجایی یک وزیر در ستون خودش
- **تابع هزینه** : تعداد تهدیدها

الگوریتم تپه‌نوردی :

به طور دلخواه از یک حالت شروع کن (مثلا شروع با h=5).

یک وزیر را به‌صورت تصادفی در ستون خودش جابجا کن. (حرکت به حالت بعدی-همسایه)

اگر هیچ همسایه‌ای بهتر از حالت فعلی نبود، متوقف شو

الگوریتم تپه‌نوردی

```
function HILL-CLIMBING(problem) returns a state
    current ← make-node(problem.initial-state)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.value ≤ current.value then
            return current.state
        current ← neighbor
```

الگوریتم سرد سازی شبیه‌سازی شده (Simulated Annealing)

❑ **ایده:** برای فرار از بهینه‌های محلی، گاهی به سمت جواب‌های بدتر حرکت کن.

- اما با گذشت زمان این حرکت‌ها را کمتر کن.

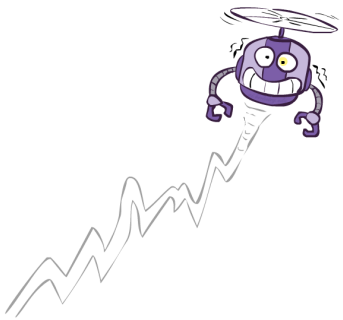
➢ اگر حالت بعدی انتخاب شده از حالت فعلی بهتر باشد ($\Delta E > 0$)، به آن حالت می‌رویم.

➢ در غیر این صورت($\Delta E < 0$)، تنها با احتمال $e^{\Delta E/T}$ به آن حالت خواهیم رفت.

- هر چقدر حالت بعدی بدتر باشد، این احتمال به صورت نمایی کاهش می‌یابد.
- همچنین با کاهش دما این احتمال کاهش می‌یابد.

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. 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^(Δ E/T)
```

❑ **پارامتر دما (T)**



➢ **دمای بالاتر:** الگوریتم تمایل بیشتری به انجام حرکات بدتر دارد (احتمال $e^{-\Delta E/T}$ بیشتره).[مانند جستجوی تصادفی]

- جستجو تصادفی‌تر و گسترده‌تر می‌شود.

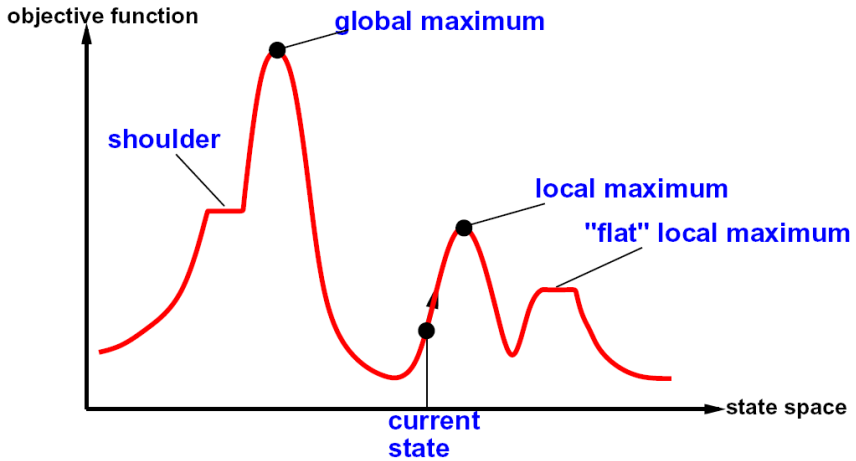
➢ **دمای پایین‌تر:** فقط جواب‌های بهتر پذیرفته می‌شود، یا با احتمال خیلی کم جواب‌های بدتر[مانند جستجوی تپه نوردی]

- جستجو متمرکز و محتاط‌تر می‌شود.

✓ هر چقدر دما آهسته‌تر کاهش یابد، تعداد مراحل جستجو و در نتیجه احتمال یافتن بهینه سراسری بیشتر است

«گرادیان نزولی» به جای «سردسازی شبیه‌سازی‌شده»
Gradient Descent as Simulated Annealing

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. 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^(Δ E/T)
```



❑ آیا می‌توان بهتر از حدس تصادفی عمل کرد؟

- بله! اگر تابع ما پیوسته و مشتق‌پذیر باشد.
  - وقتی تابع پیوسته و مشتق‌پذیر باشد، می‌توان از گرادیان (مشتق) برای هدایت حرکت به سمت نقاط بهتر استفاده کرد.

➢ در Simulated Annealing به طور تصادفی همسایه‌ها امتحان می‌شوند

➢ در Gradient Descent از مشتق تابع استفاده می‌شود تا جهت حرکت بهینه دقیق‌تر انتخاب شود.

✓ Annealing برای مسائل گسسته مناسب است چون مشتق نداریم

✓ Gradient Descent برای مسائل پیوسته و قابل مشتق‌گیری مناسب است چون دقیق‌تر به سمت جواب بهتر حرکت می‌کند.

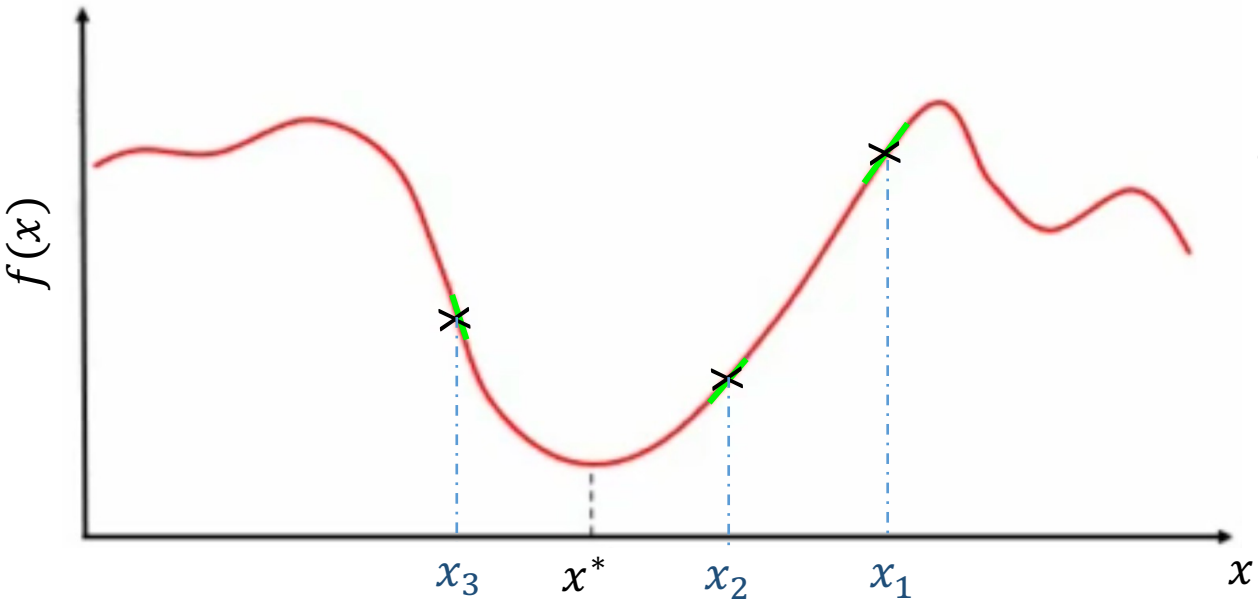
گرادیان نزولی (Gradient Descent)

❑الگوریتم گرادیان نزولی :

- از یک نقطه دلخواه روی نمودار تابع شروع کن.
- در هر مرحله، مقدار شیب (گرادیان) تابع را در آن نقطه حساب کن.
- در جهت شیب منفی حرکت کن (چون می‌خوایم مقدار تابع رو کم کنیم).
- این کار را تکرار کن تا به نقطه‌ای برسی که شیب تقریباً صفر شود (یعنی مینیمم رسیدی).

$$x_{\text{new}} = x_{\text{current}} - \eta \cdot \nabla f(x_{\text{current}})$$

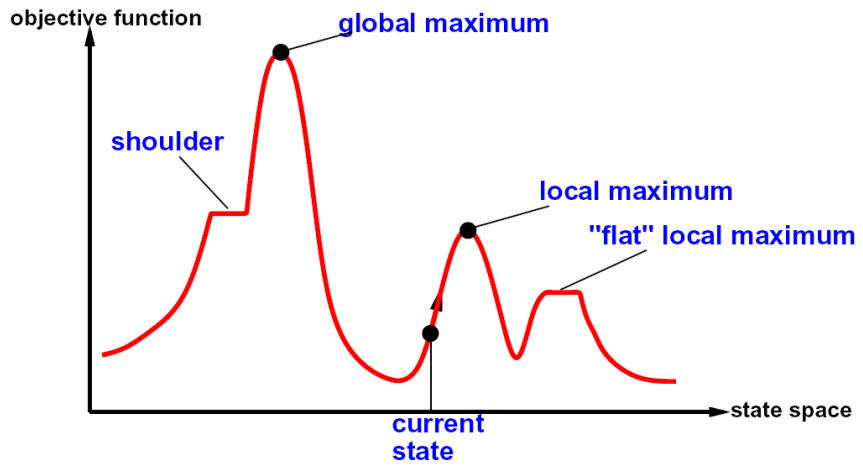
η: نرخ یادگیری (learning rate) – یعنی چقدر جلو بریم



✓ بسیاری از الگوریتم‌های یادگیری ماشین، از جست‌وجوهای محلی (مانند گرادیان نزولی) برای بهینه سازی استفاده می‌کنند.

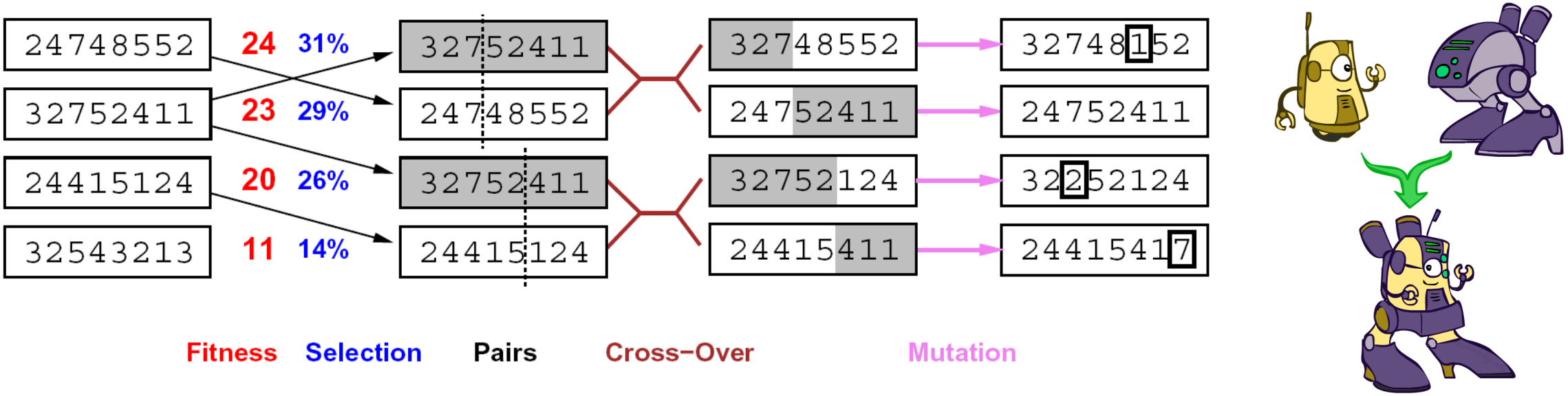
الگوریتم‌های ژنتیک (Genetic Algorithms)

```
{
   initialize population;
   evaluate population;
   while TerminationCriteriaNotSatisfied
   {
      select parents for reproduction;
      perform crossover and mutation;
      evaluate population;
   }
}
```



- الگوریتم ژنتیک به جای اینکه فقط یک راه‌حل را اپدیت کند (مثل تپه‌نوردی)، یک جمعیت از راه‌حل‌ها را نگه می‌دارد.
- هر راه‌حل ممکن (که "فرد" یا فردیت / کروموزوم نامیده می‌شود) در هر نسل مورد ارزیابی قرار می‌گیرد و افراد بهتر (بر اساس یک تابع برازندگی یا fitness function) انتخاب می‌شوند
- این افراد ترکیب (crossover) و جهش (mutation) پیدا می‌کند تا نسل بعدی ساخته شود.

الگوریتم‌های ژنتیک (Genetic Algorithms)



- الگوریتم‌های ژنتیک از ایده انتخاب طبیعی استفاده می‌کنند.
  - در هر مرحله بر اساس تابع برازندگی (Fitness Function) فقط N فرضیه بهتر را نگهدار. (گزینش)
  - همچنین از عملگرهای ژنتیکی ترکیب (Crossover) و جهش (Mutation) برای ایجاد گوناگونی (تنوع) استفاده می‌کنند.