جستجوی محلی

سید ناصر رضوی n.razavi@tabrizu.ac.ir
۱۳۹۷

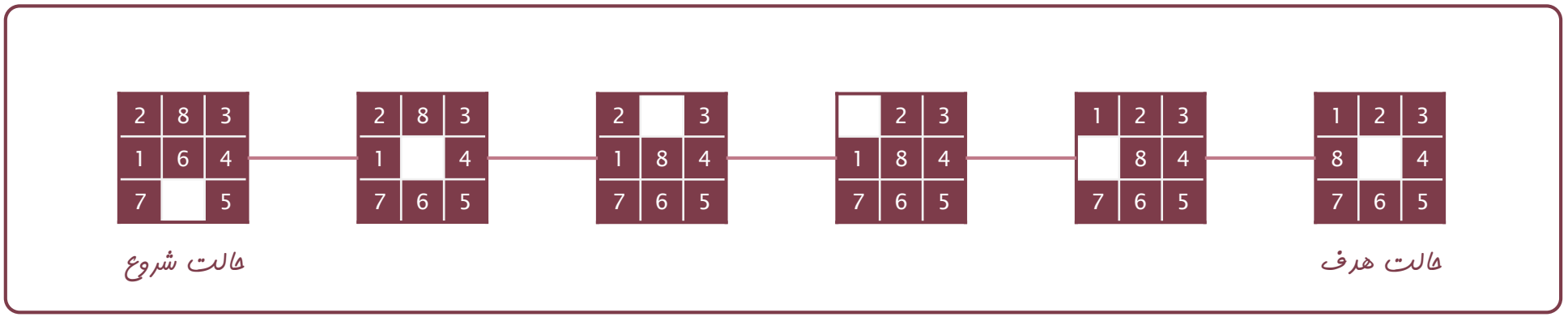
فهرست مطالب

- جستجوی محلی.
- الگوریتم تپه‌نوردی.
- الگوریتم سرد سازی شبیه‌سازی شده.
- الگوریتم ژنتیک.

یادآوری: جستجو

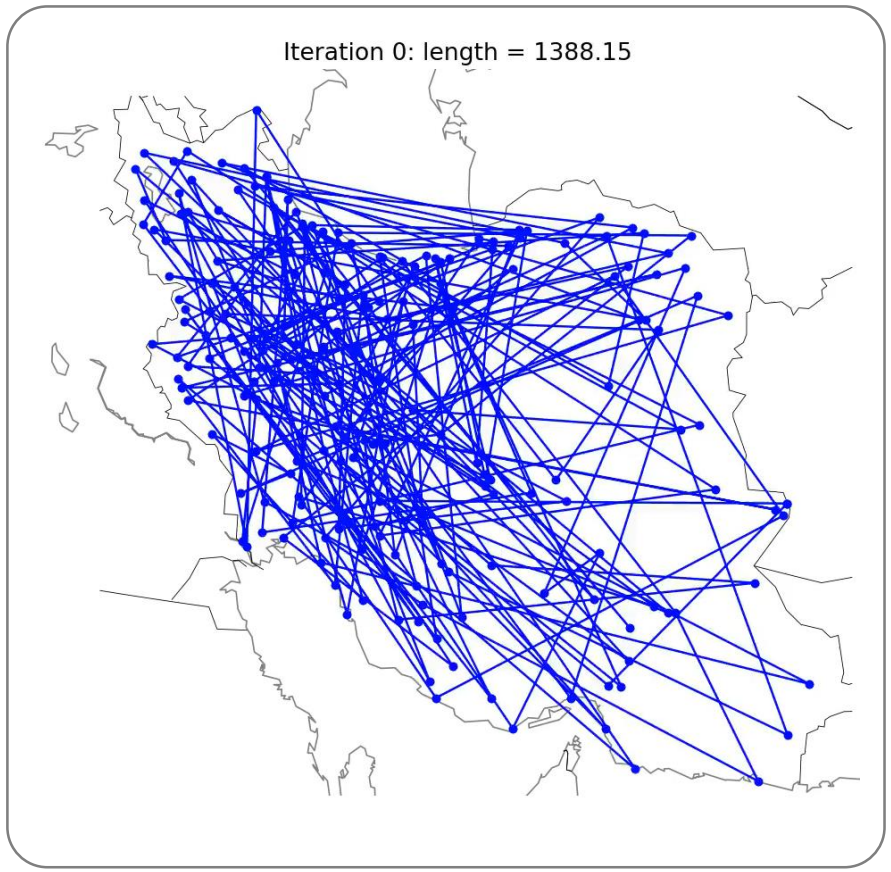
- هدف. یافتن مسیری با کم‌ترین هزینه از حالت شروع به حالت هدف در گراف فضای حالت.

- مثال. معمای هشت



جستجوی محلی

- ایده. در بسیاری از مسائل بهینه‌سازی مسیر راه‌حل اهمیت ندارد؛ بلکه خود حالت هدف مهم است.

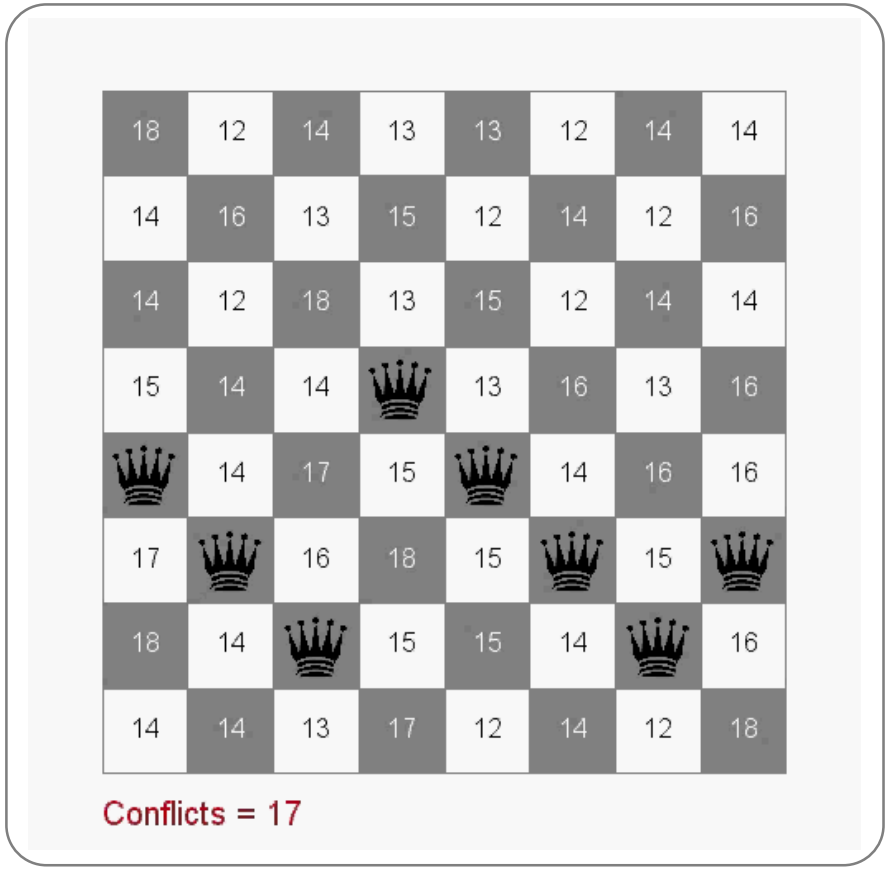


- فضای حالت. یک مجموعه از «پیکره‌بندی‌های کامل»

مسئله فروشنده دوره‌گرد: یافتن یک تور بهینه

جستجوی محلی

- ایده. در بسیاری از مسائل بهینه‌سازی مسیر راه‌حل اهمیت ندارد؛ بلکه خود حالت هدف مهم است.

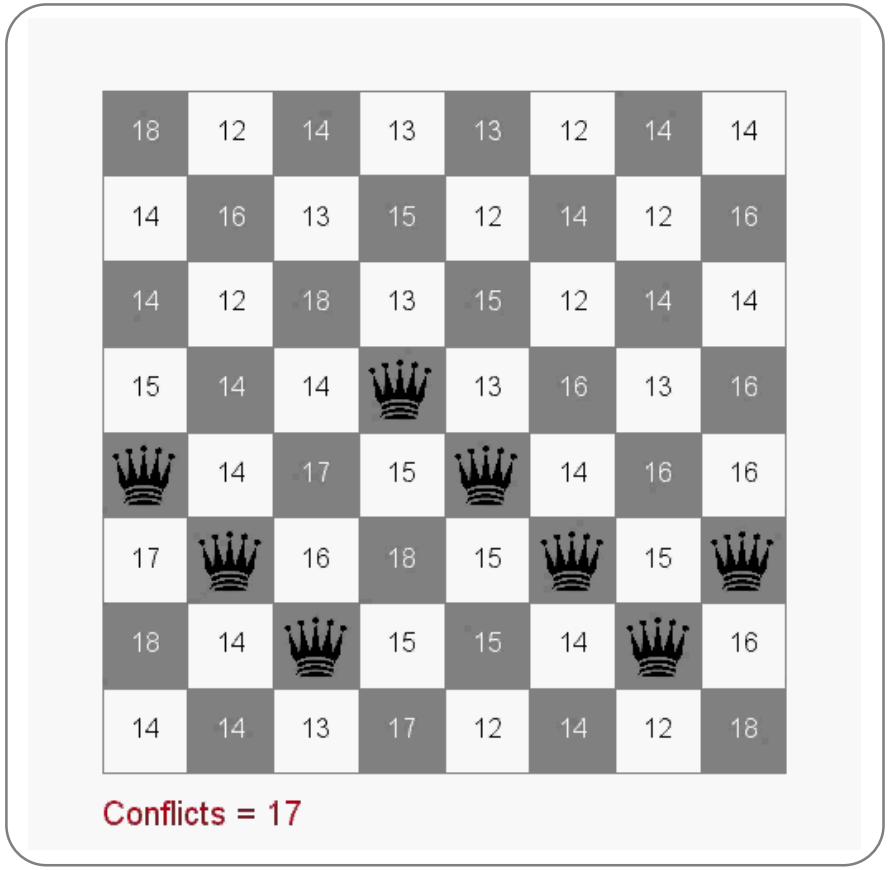


- فضای حالت. یک مجموعه از «پیکره‌بندی‌های کامل»

مسئله N-وزیر: یافتن یک چیدمان که تمام محدودیت‌های مسئله را برآورده می‌کند.

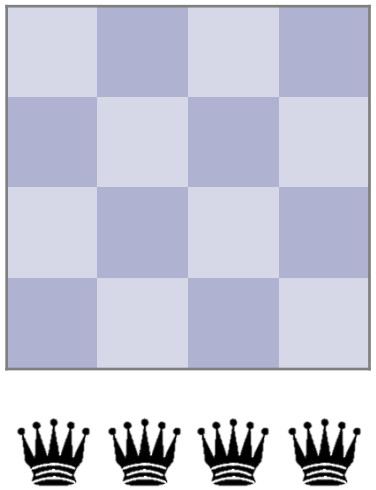
جستجوی محلی

- جستجوی محلی.
  - استفاده از یک حالت به عنوان حالت فعلی
  - بهبود حالت فعلی از طریق حرکت به یکی از حالت‌های همسایه
- مزایا.
  - مصرف حافظه بسیار کم [عدم ذخیره‌سازی گره‌های حاشیه‌ای]
  - یافتن راه‌حل‌های معقول در فضاهای بسیار بزرگ یا نامحدود
- مفید برای مسائل بهینه‌سازی محض.
  - یافتن بهترین پاسخ بر طبق تابع هدف



مثال: مسئله N-وزیر

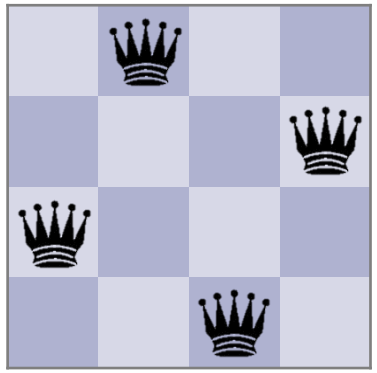
- مسئله. N وزیر را در یک صفحه شطرنج N در N به گونه‌ای قرار بده که هیچ دو وزیری در یک سطر، ستون یا قطر قرار نگیرند.



مثال: مسئله N-وزیر

- مسئله. N وزیر را در یک صفحه شطرنج N در N به گونه‌ای قرار بده که هیچ دو وزیری در یک سطر، ستون یا قطر قرار نگیرند.
- اعمال. جابجایی یک وزیر در ستون مربوط به خودش.

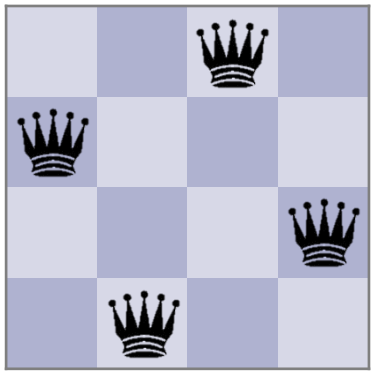
اندازه فضای حالت: N^N



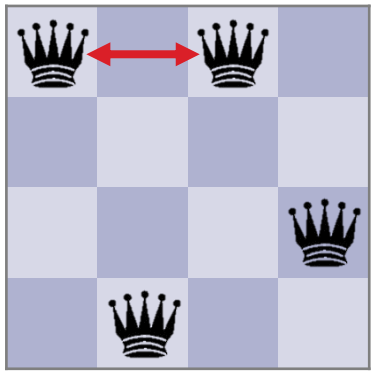
مثال: مسئله N-وزیر

- مسئله. N وزیر را در یک صفحه شطرنج N در N به گونه‌ای قرار بده که هیچ دو وزیری در یک سطر، ستون یا قطر قرار نگیرند.

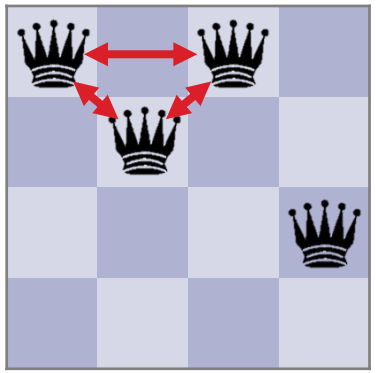
- اعمال. جابجایی یک وزیر در ستون مربوط به خودش.



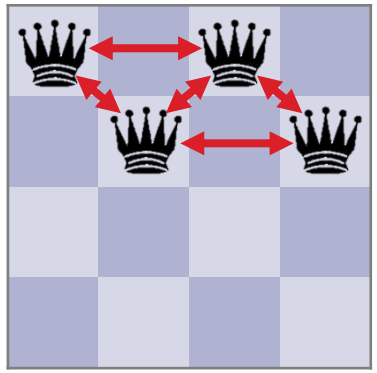
تعداد درگیری‌ها = ۰



تعداد درگیری‌ها = ۱

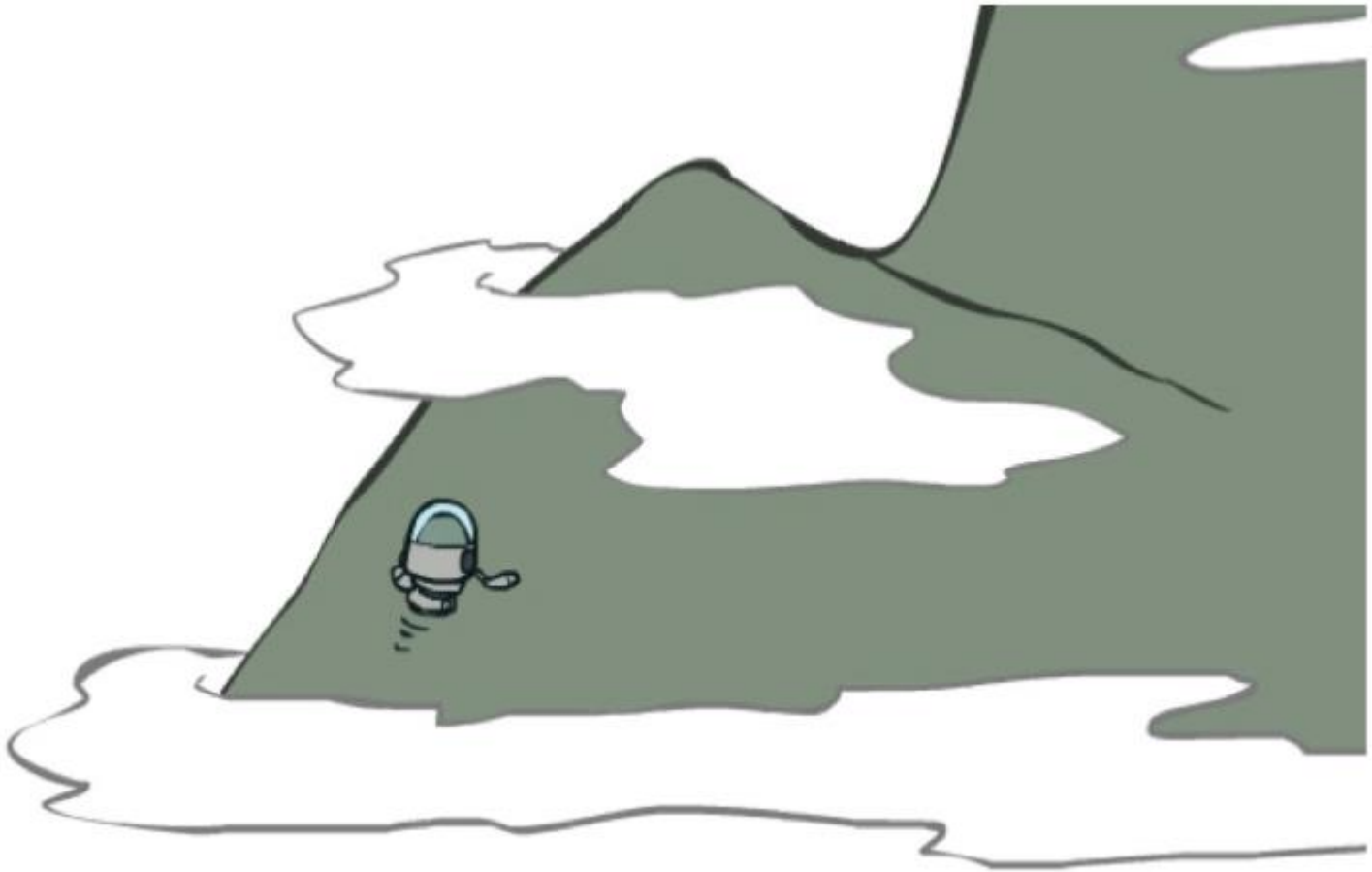


تعداد درگیری‌ها = ۳



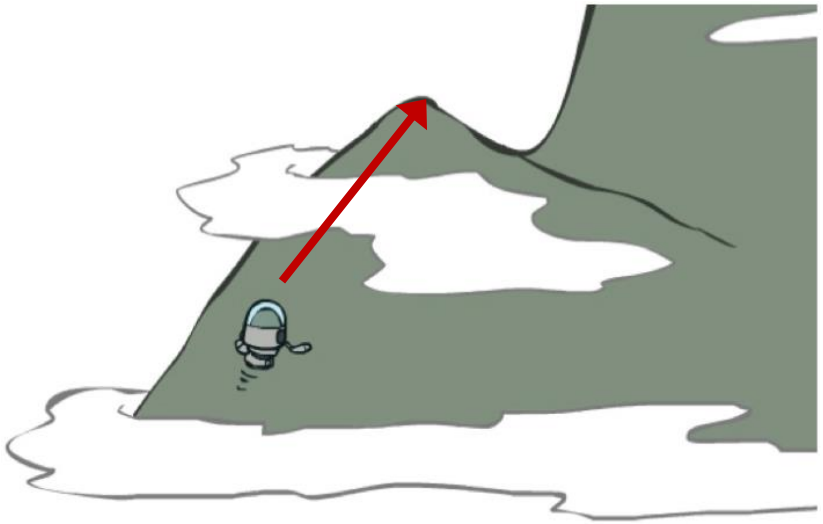
تعداد درگیری‌ها = ۵

تپه‌نوردی



تپه‌نوردی

- یک ایده ساده و کلی.
  - شروع: از یک حالت دلخواه
  - تکرار: حرکت به بهترین حالت همسایه
  - توقف: اگر هیچ یک از همسایه‌ها از خود حالت فعلی بهتر نیستند.



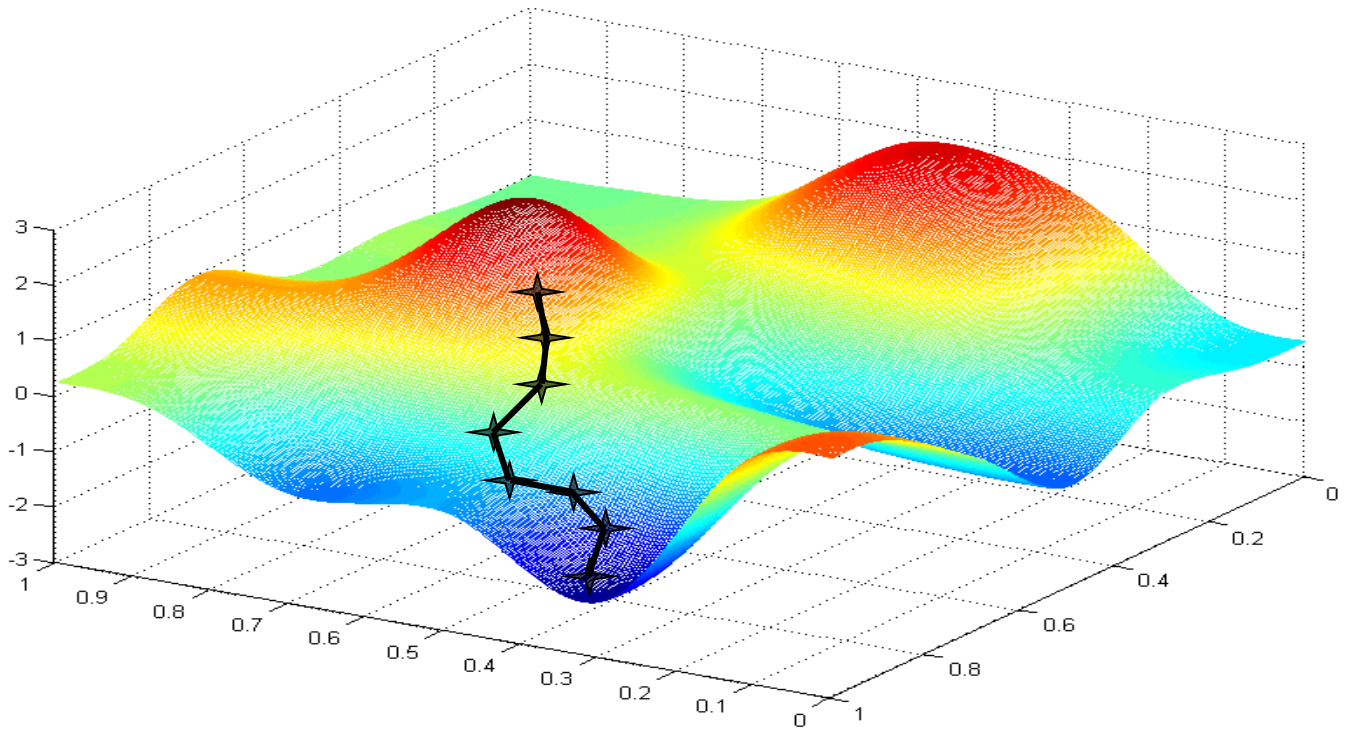
- مزایا.
  - سرعت بالا،
  - مصرف حافظه کم
- معایب.
  - کامل؟ خیر
  - بهینه؟ خیر

تپه‌نوردی (گرادیان کاهشی یا افزایشی)

- تپه‌نوردی. مانند «بالا رفتن از کوه اورست در مه غلیظ با ضعف حافظه»

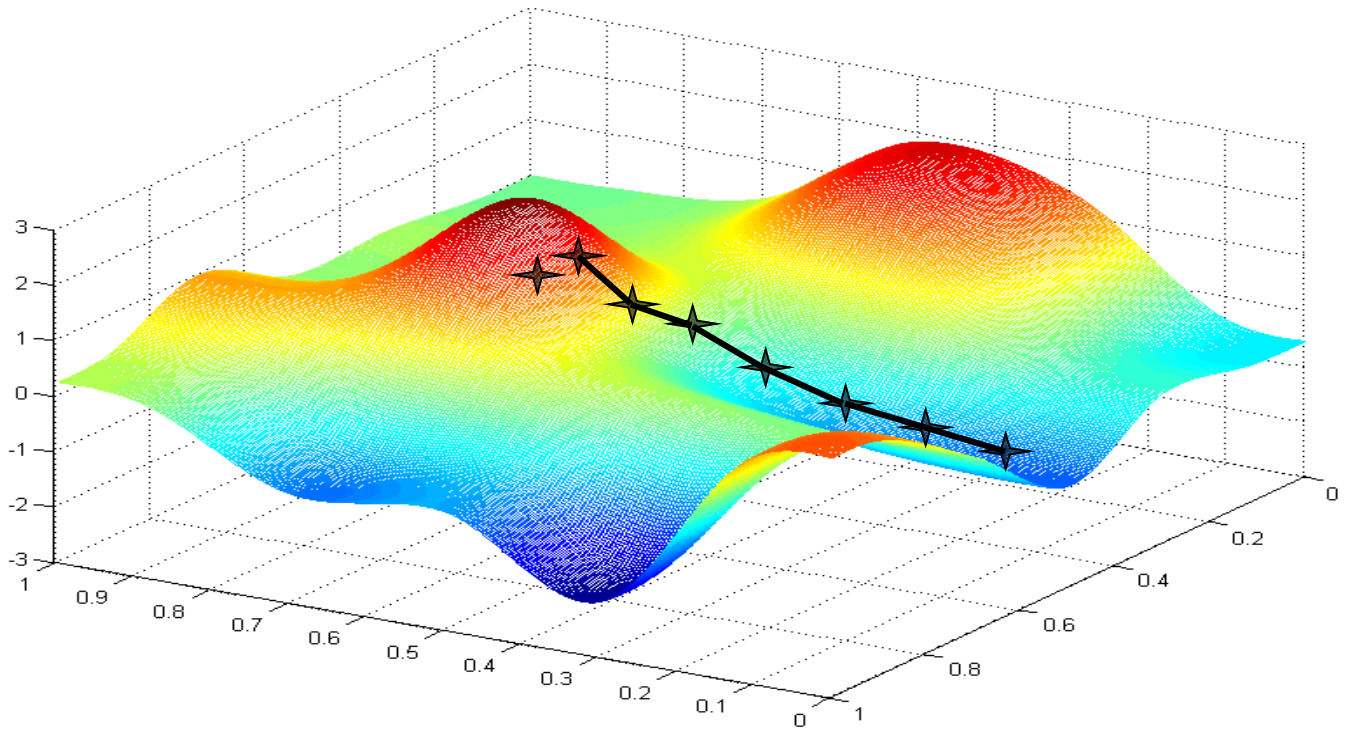
```
function HILL-CLIMBING(problem) returns a state that is a local maximum

  inputs: problem, a problem
  local variables: current, a node
                   neighbor, a node

  current ← MAKE-NODE(problem.INITIAL-STATE)

  loop do
    neighbor ← a highest valued successor of current
    if neighbor.VALUE ≤ current.VALUE then return current.STATE
    current ← neighbor
```

الگوریتم تپه‌نوردی: بهینه سراسری



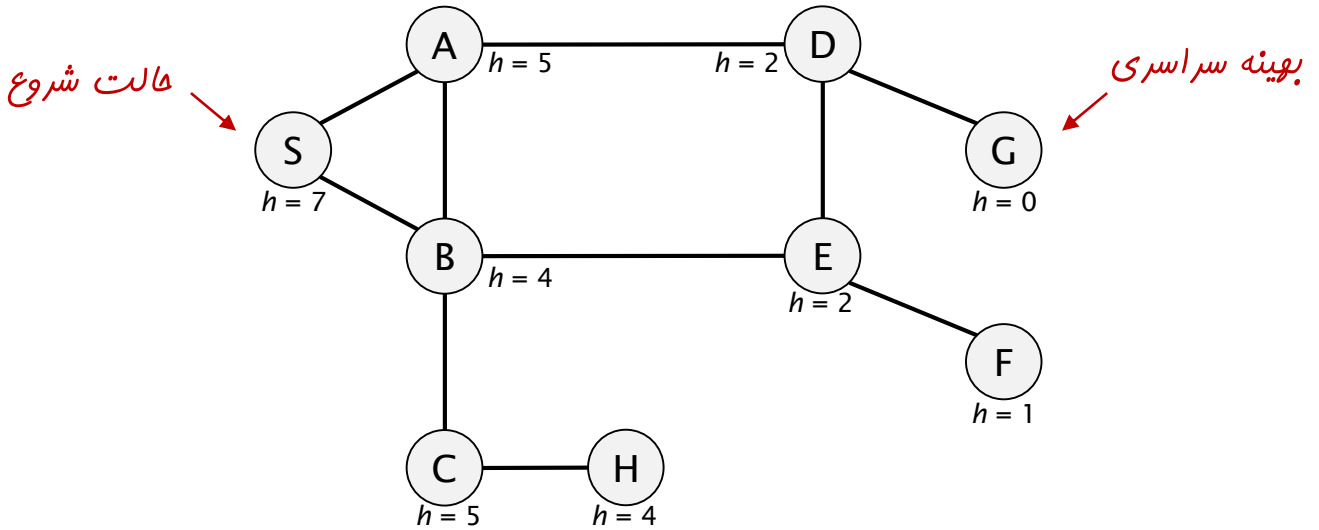
تپه‌نوردی در یک فضای حالت پیوسته برای یافتن کمینه

الگوریتم تپه‌نوردی: بهینه محلی



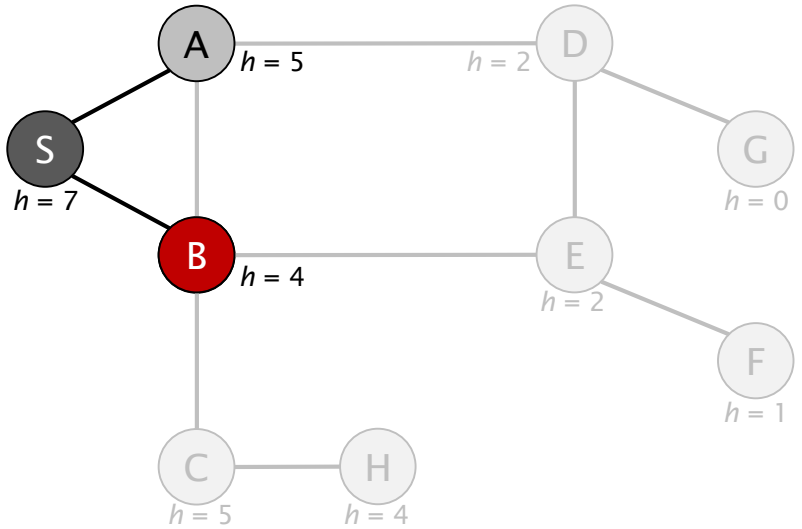
تپه‌نوردی در یک فضای حالت پیوسته برای یافتن کمینه

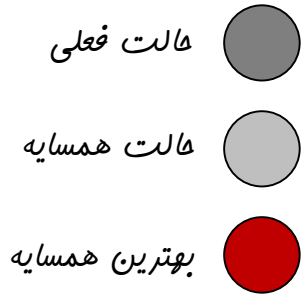
الگوریتم تپه‌نوردی



تپه‌نوردی در یک فضای حالت گسسته برای یافتن کمینه

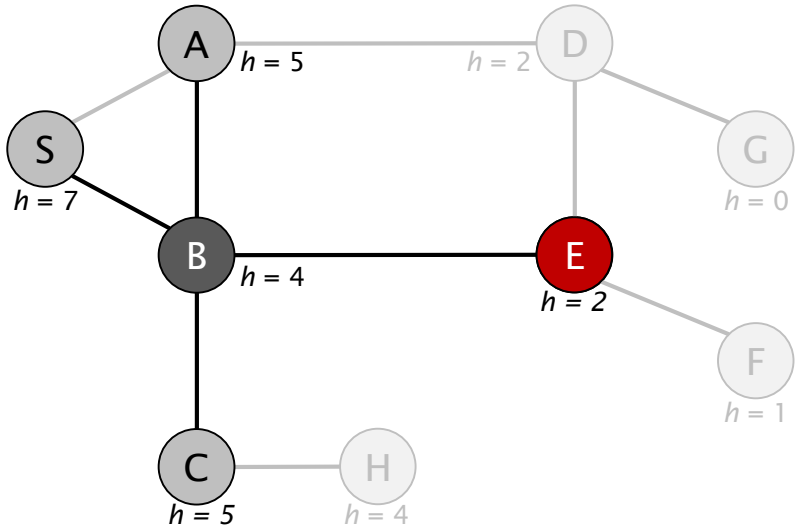
الگوریتم تپه‌نوردی

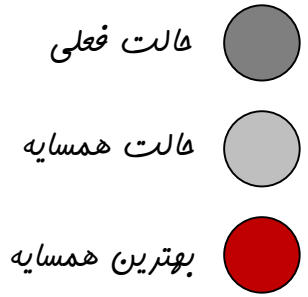




تپه‌نوردی در یک فضای حالت گسسته برای یافتن کمینه

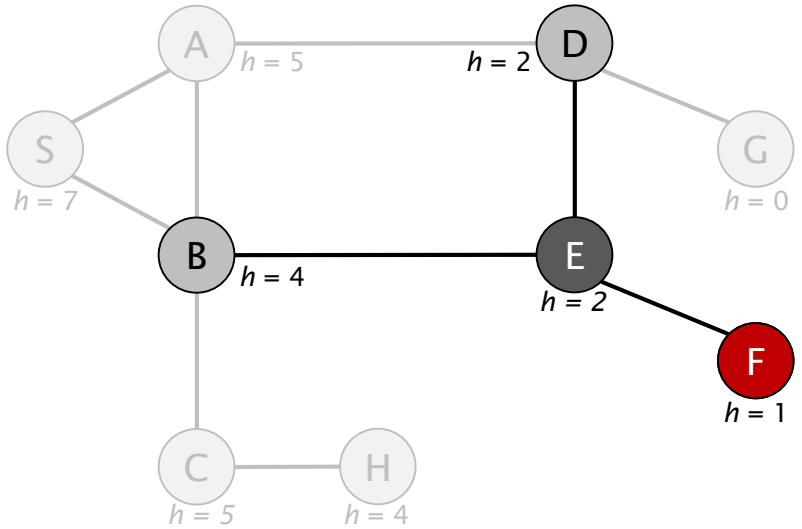
الگوریتم تپه‌نوردی

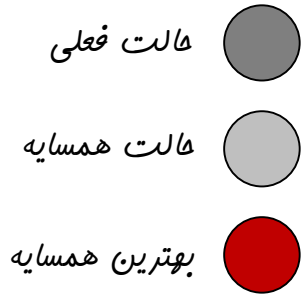




تپه‌نوردی در یک فضای حالت گسسته برای یافتن کمینه

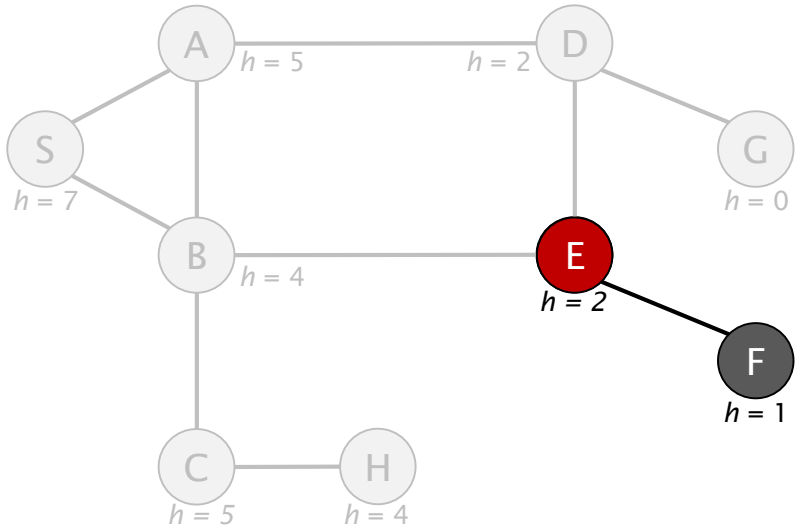
الگوریتم تپه‌نوردی

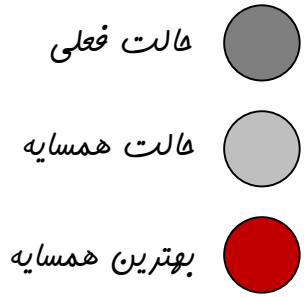




تپه‌نوردی در یک فضای حالت گسسته برای یافتن کمینه

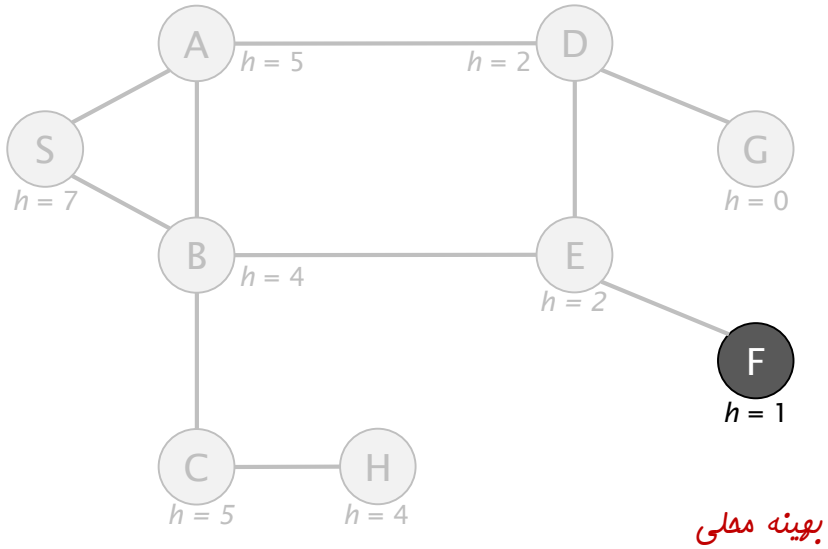
الگوریتم تپه‌نوردی

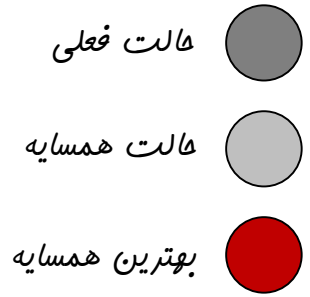




تپه‌نوردی در یک فضای حالت گسسته برای یافتن کمینه

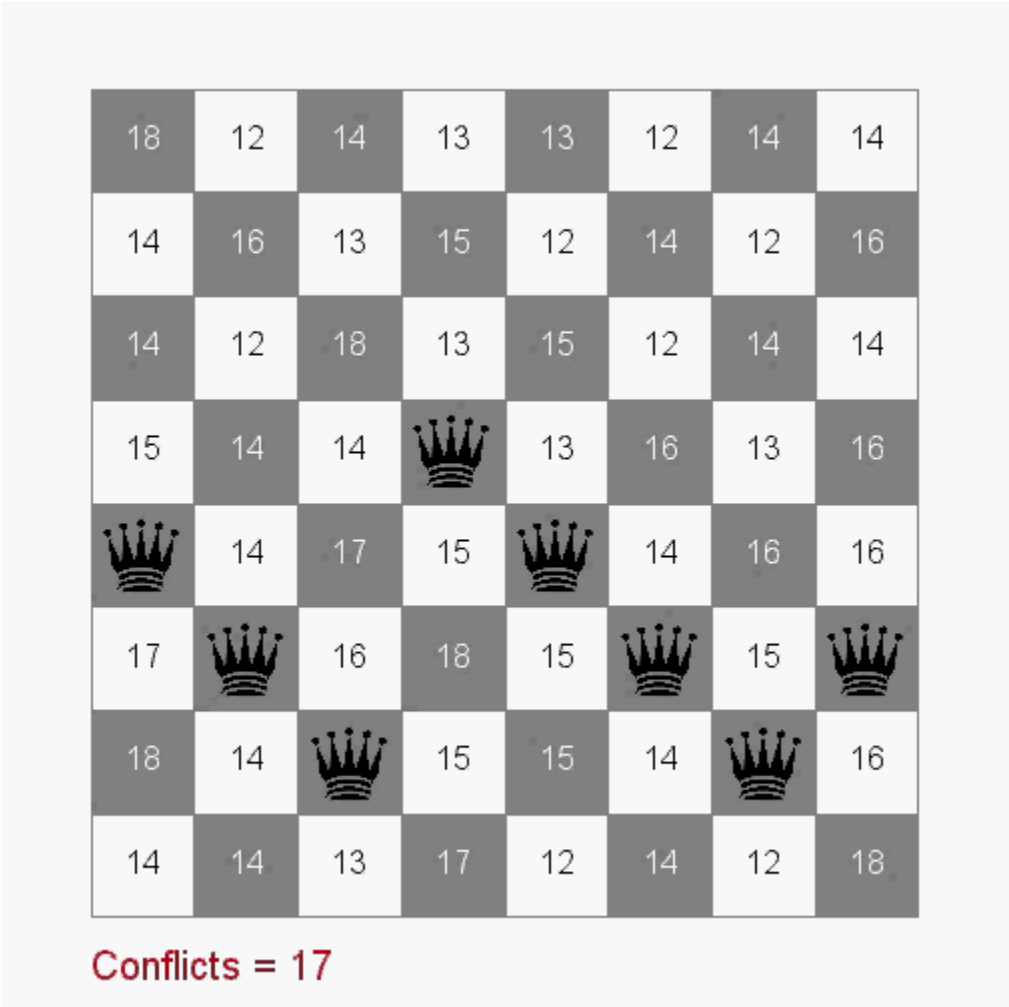
الگوریتم تپه‌نوردی





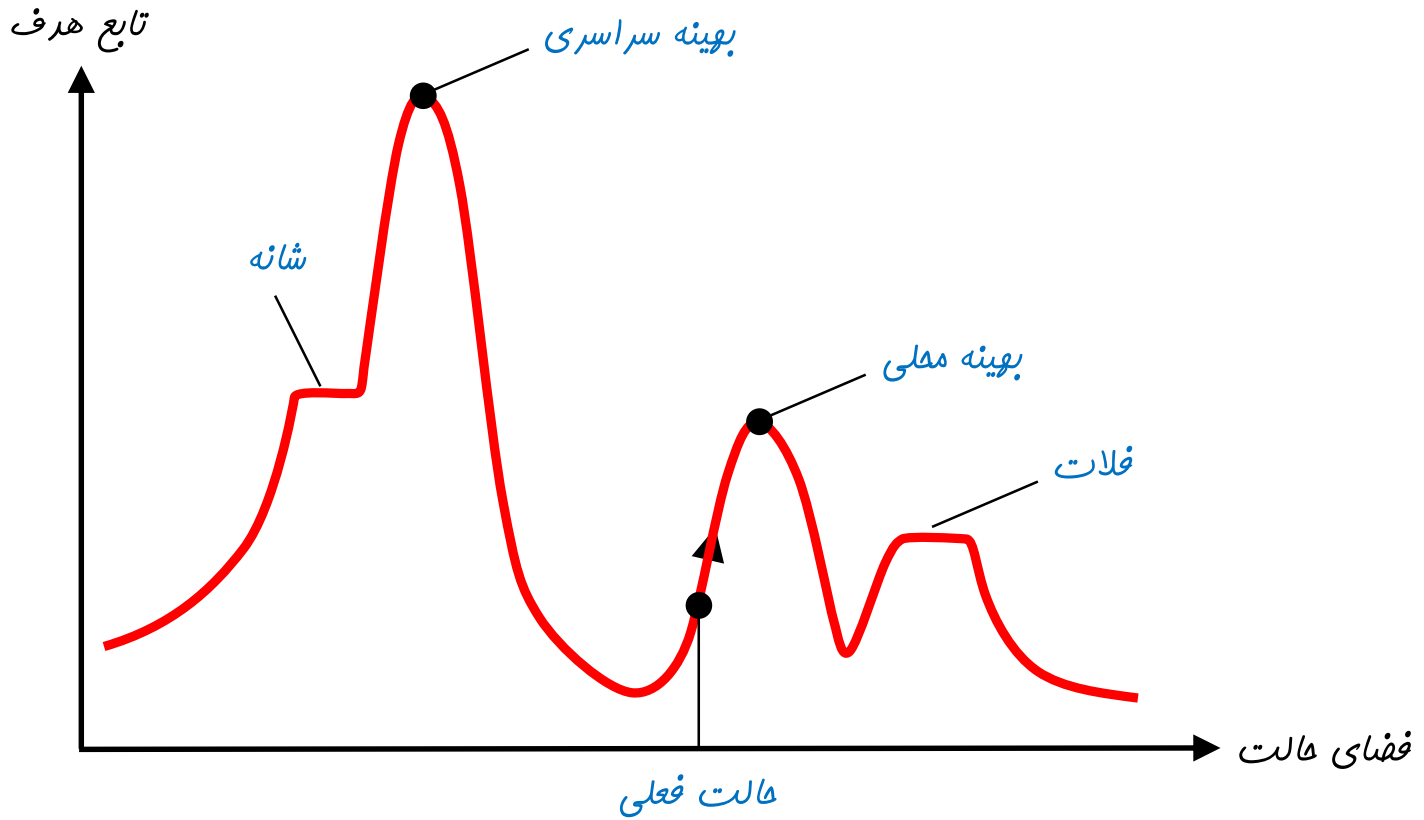
تپه‌نوردی در یک فضای حالت گسسته برای یافتن کمینه

الگوریتم تپه‌نوردی: اجرای نمایشی

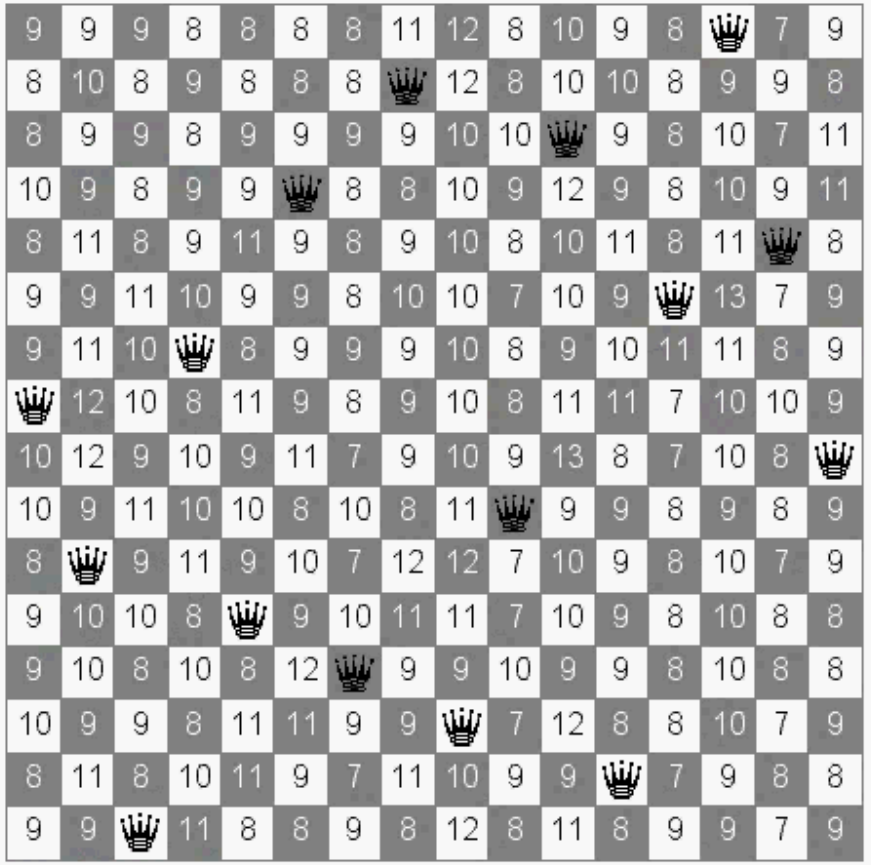


تپه‌نوردی

- مشکل. بسته به حالت شروع ممکن است در دام بهینه محلی گرفتار شود.



تپه‌نوردی به همراه حرکت‌های جانبی



الگوریتم سرد سازی شبیه‌سازی شده

الگوریتم سرد سازی شبیه‌سازی شده

- ایده. از بهینه محلی با انجام حرکت‌های بد فرار کن.

[اما به تدریج اندازه و تعداد حرکت‌های بد رو به پایین را کاهش بده]

function SIMULATED-ANNEALIING(*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**
  $T \leftarrow schedule[t]$
  if $T = 0$ **then return** *current*
  next ← a randomly selected successor of *current*
  $\Delta E \leftarrow next$.VALUE - *current*.VALUE
  if $\Delta E > 0$ **then** *current* ← *next*
  else *current* ← *next* only with probability $e^{\Delta E/T}$

الگوریتم سرد سازی شبیه‌سازی شده

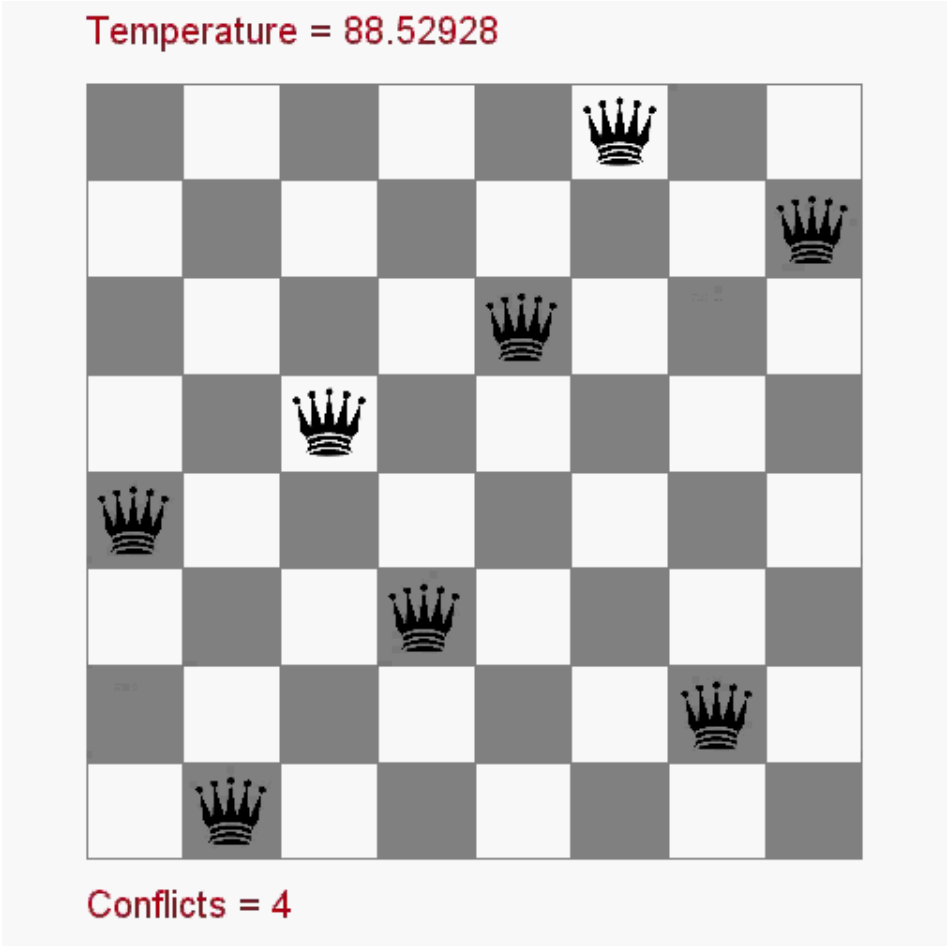
- الگوریتم سرد سازی شبیه‌سازی شده.
  - همانند تپه‌نوردی، اما در هر مرحله حالت بعدی به طور تصادفی انتخاب می‌شود.
  - اگر حالت بعدی انتخاب شده از حالت فعلی بهتر باشد، به آن حالت می‌رویم.
  - در غیر این صورت، تنها با احتمال $e^{\Delta E/T}$ به آن حالت خواهیم رفت.
    - هر چقدر حالت بعدی بدتر باشد، این احتمال به صورت نمایی کاهش می‌یابد.
    - همچنین با کاهش دما این احتمال کاهش می‌یابد.

- پارامتر دما. تعیین کننده تعداد مراحل جستجو
  - هر چقدر دما آهسته‌تر کاهش یابد، تعداد مراحل جستجو و در نتیجه احتمال یافتن بهینه سراسری بیشتر است.

الگوریتم سرد سازی شبیه‌سازی شده

- نقش پارامتر دما.
  - دمای بالاتر: احتمال انجام حرکات بد بیشتر [مانند جستجوی تصادفی]
  - دمای پایین‌تر: احتمال انجام حرکات بد کمتر [مانند جستجوی تپه نوردی]
- گزاره. اگر دما به آرامی کاهش یابد، الگوریتم پاسخ بهینه سراسری را با احتمالی که به سمت یک میل می‌کند، پیدا خواهد نمود.
- کاربردها.
  - برنامه‌ریزی
  - زمان‌بندی خطوط هواپیمایی
  - حل مسائل VLSI

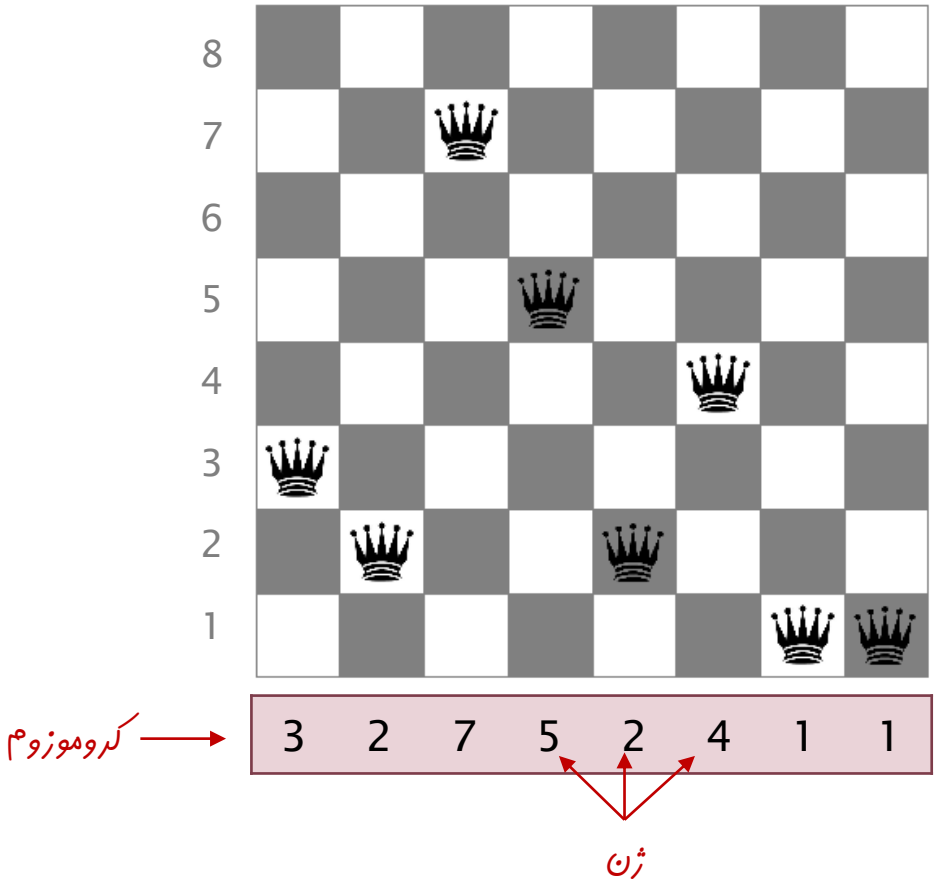
الگوریتم سرد سازی شبیه‌سازی شده



الگوریتم‌های ژنتیک

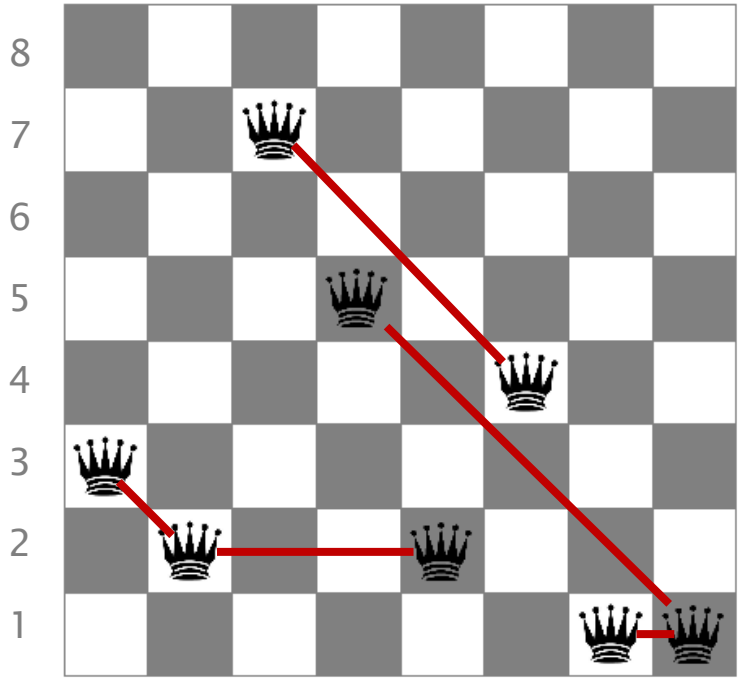
الگوریتم‌های ژنتیک: بازنمایی

- بازنمایی. انتخاب نحوه نمایش راه‌حل‌های بالقوه.



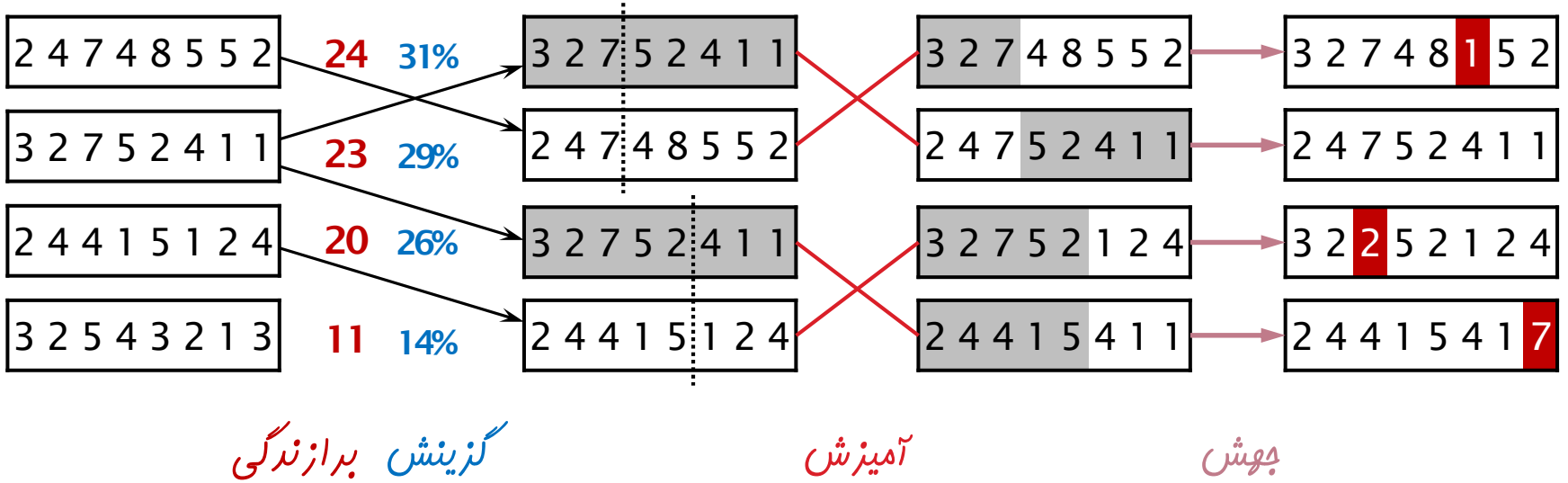
الگوریتم‌های ژنتیک: تابع برازندگی

- تابع برازندگی. محاسبه کیفیت راه‌حل‌های بالقوه.



$$Fitness = \binom{n}{2} - conflicts$$

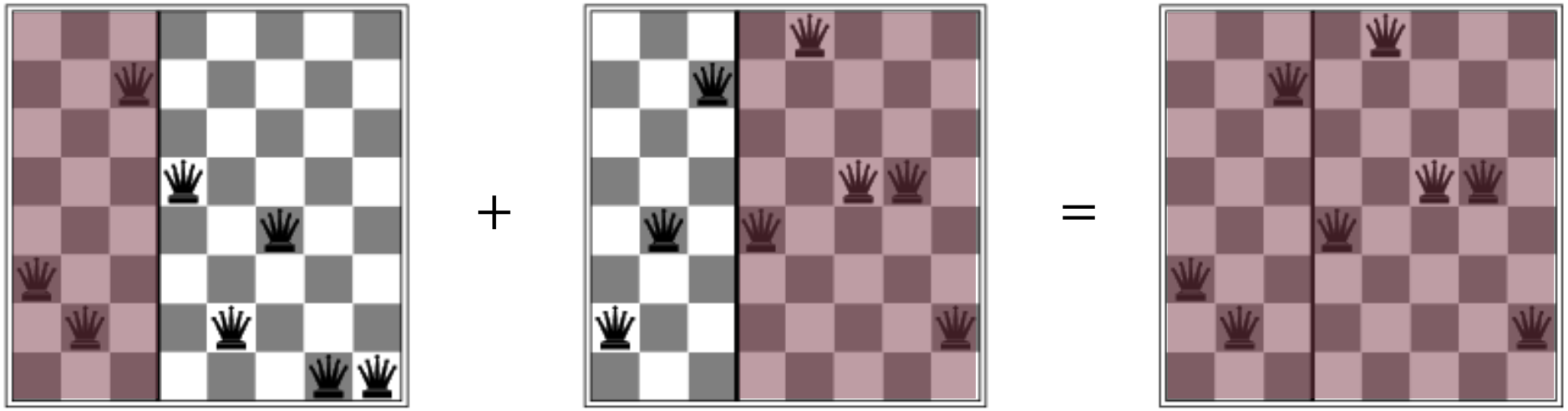
الگوریتم‌های ژنتیک



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

الگوریتم‌های ژنتیک

- عملگر آمیزش.



الگوریتم‌های ژنتیک

function GENETIC-ALGORITHM(*population,* FITNESS-FN) **returns** an individual

input: *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 ← CROSSOVER(*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