شانلی حاج الف‌خانی اصل
۹۸۱۳۰۱۱۰۹

هوش مصنوعی
گزارش پیاده‌سازی الگوریتم‌های جست و جو

در این گزارش عملکرد چهار الگوریتم را برای مساله‌ی n وزیر بررسی می‌کنیم.

۱ Hill Climbing

در ابتدا صفحه‌ی شطرنج رندوم تولید کرده و شروع می‌کنیم. در هر مرحله از بین تمام همسایه‌های وضعیت فعلی، آن همسایه که در آن کمترین تعداد وزیر یکدیگر را تحدید می‌کنند را انتخاب کرده. اگر تعداد تحدیدهای این وضعیت از وضعیت قبلی کمتر بود، به این وضعیت می‌رویم و در غیر این صورت در وضعیت قبلی خود می‌مانیم و وضعیت فعلی را به عنوان خروجی می‌دهیم.

تابع هیوریستیک در این الگوریتم تعداد وزیرهایی است که همدیگر را تحدید می‌کنند.

تابع ساکسسور همسایه‌ی وضعیت فعلی را با کمترین تعداد تحدید وزیر خروجی می‌دهد.

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

$current \leftarrow$ MAKE-NODE($problem$.INITIAL-STATE)
loop do
  $neighbor \leftarrow$ a highest-valued successor of $current$
  if neighbor.VALUE $\leq$ current.VALUE **then return** $current$.STATE
  $current \leftarrow neighbor$

Figure 4.2 The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor; in this version, that means the neighbor with the highest VALUE, but if a heuristic cost estimate h is used, we would find the neighbor with the lowest h.

۲ K-Beam Search

این الگوریتم عملکردی شبیه به الگوریتم تپه نوردی دارد اما به جای یک وضعیت، k وضعیت را نگه می‌دارد. در هر مرحله از بین تمام همسایه‌های این k وضعیت، kتای بهتر را می‌گیرید.

این کار تا زمانی تکرار می‌شود که تعداد تهدیدهای یکی از این k نقطه برابر صفر شود.

تابع هیوریستیک و ساکسسور مانند الگوریتم تپه نوردی است.

۳ Simulated Annealing

در ابتدا صفحه‌ی شطرنج رندوم تولید کرده و شروع می‌کنیم. در هر مرحله از بین تمام همسایه‌های وضعیت فعلی یکی را رندوم انتخاب کرده. اگر تعداد تحدیدهای این وضعیت از وضعیت قبلی کمتر بود، به این وضعیت می‌رویم و در غیر با احتمال $e^{\Delta E/T}$ تغییر وضعیت می‌دهیم که T نشان‌دهنده‌ی متغیر *tempreture* است که هربار با یک ضریب *schedule* کوچک‌تر می‌شود و ΔE برابر اختلاف تعداد تهدیدهای وضعیت فعلی و وضعیت بعدی است.
تابع هیوریستیک مانند الگوریتم قبلی تعریف می‌شود و تابع ساکسسور همسایه‌ی رندوم وضعیت فعلی را خروجی می‌دهد.

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**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow next$.VALUE – *current*.VALUE
    if ΔE 0 **then** *current* ← *next*
    else *current* ← *next* only with probability $^{\Delta E/T}$

Figure 4.5 The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The *schedule* input determines the value of the temperature T as a function of time.

Genetic ۴

در این الگوریتم هم در ابتدا با توجه به تعداد جمعیت population size وضعیت‌های رندومی تولید می‌شود. و سپس تا وقتی که به وضعیت مطلوب نرسیدیم مراحل زیر را تکرار می‌کنیم.

در ابتدا مقدار fitness را برای هر وضعیت حساب می‌کنیم و با توجه به آن شانس انتخاب شدن selection به وضعیت‌ها می‌دهیم. وضعیت‌های انتخاب شده را دو به دو با هم ترکیب می‌کنیم cross over و سپس با تابع mutate یکی از کروموزوم‌ها را رندوم تغییر می‌دهیم. و بدین طریق نسل بعدی به دست می‌آید.

تابع فیتنس تعداد وزیرهایی هست که همدیگر را تحدید نمی‌کنند.

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))

Figure 4.8 A genetic algorithm. The algorithm is the same as the one diagrammed in Figure 4.6, with one variation: in this more popular version, each mating of two parents produces only one offspring, not two.

در این نمودار به ازای nهای مختلف زمان و تابع هیوریستیک الگوریتم‌های مختلف را با هم مقایسه می‌کنیم.