

به نام خداوند بخشنده و مهربان



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

درس هوش مصنوعی

گزارش پروژه

پیااده‌سازی جستجوی محلی برای کمینه‌سازی مسائل مختلف با نمونه مسائل تصادفی

استاد درس:

دکتر قطعی

نام دانشجو:

امیررضا رادجو

۹۷۱۳۰۱۸

بهار ۱۴۰۰

در این گزارش به مقایسه دو الگوریتم تپه نوردی و ذوب فلزات میپردازیم و از مزایا و معایب هر کدام نام برده و آنها را در عمل با یکدیگر مقایسه میکنیم تا دریابیم در چه زمانی استفاده از کدام یک از این الگوریتم ها سود بیشتری به ما خواهد رساند

ابتدا لازم است با هر یک از الگوریتم ها به طور کلی آشنا شویم.

الگوریتم اول: الگوریتم جستجوی تپه نوردی

یک وضعیت دلخواه را به عنوان وضعیت فعلی در نظر میگیرد و سپس از میان همسایه های وضعیت فعلی بهترین همسایه را انتخاب مینماید

- مدام در جهت بهبودی اوضاع حرکت میکند
- زمانی متوقف میشود که هیچ همسایه بهتری موجود نباشد
- به جای ثبت حالت مسیر تنها کافیست گره مد نظر ثبت شود

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

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

loop do

neighbor ← a highest-valued successor of *current*

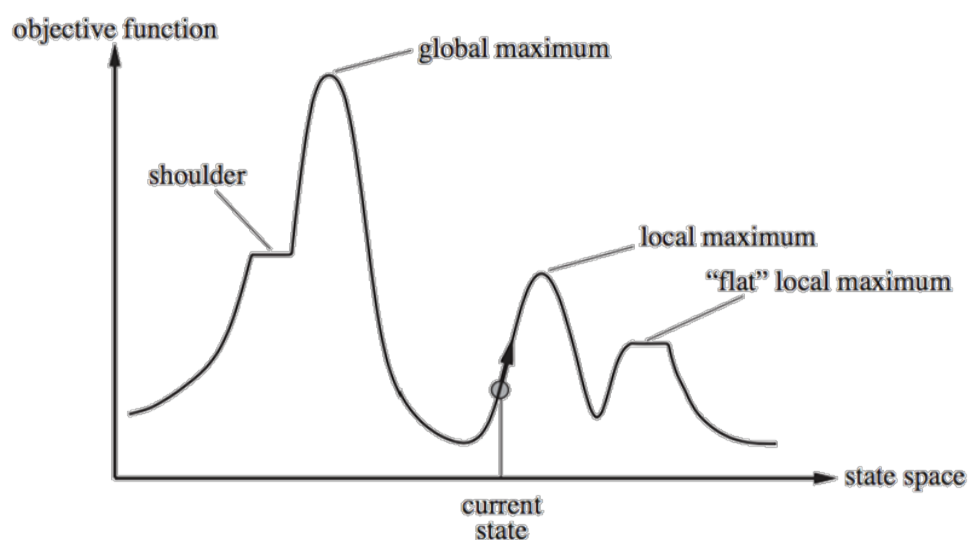
if *neighbor*.VALUE ≤ *current*.VALUE **then return** *current*.STATE

current ← *neighbor*

از مشکلات جستجوی تپه نوردی میتوان به گیر کردن در ماکسیمم های محلی اشاره کرد.

احتمال رخ دادن این حالت زمانی بیشتر است که نقطه تصادفی انتخابی در نزدیکی بیشینه های محلی قرار داشته باشد در این صورت الگوریتم به بیشینه محلی مورد نظر میرسد و آن را به عنوان جواب نهایی انتخاب میکند بنابراین میتوان بیان کرد که الگوریتم جستجوی تپه نوردی به نقطه تصادفی انتخابی وابسته است و نقاط تصادفی متفاوت امکان دادن جواب های متنوع را دارند بنابراین در یک مساله خاص با الگوریتم جستجوی تپه نوردی حالتی به مانند تابع ثابت به وجود نخواهد آمد زیرا تابع جواب به نقطه تصادفی انتخابی وابسته است.

یکی دیگر از مواردی که برای الگوریتم جستجوی تپه نوردی مشکل ایجاد میکند فلات ها میباشد. فلات ها شامل بیشینه های محلی ثابت یا شانه ها میباشد که دارای مقداری ثابت در قسمت هایی از تابع هستند که باعث گیر کردن الگوریتم در این نقاط میشوند.



یکی از بزرگترین مشکلات برای تمامی الگوریتم های حریصانه دماغه ها میباشدند. دماغه ها در واقع تعدادی بیشینه محلی پشت سر هم هستند که گذشتن از آن ها برای الگوریتم های حریصانه کاری بسیار دشوار است و پیاده سازی های خاص و منحصر به فردی برای مواجهه با آنها نیاز است.



الگوریتم بعدی: الگوریتم ذوب فلزات

در این الگوریتم تلاش شده تا شانس به وجود بیاید تا الگوریتم بتواند از ماکسیمم های مرکزی فرار کند. این الگوریتم در واقع ترکیبی از دو الگوریتم قدم زدن تصادفی و الگوریتم جستجوی تپه نوردی میباشد

در صورتی که الگوریتم قدم زدن تصادفی را در تعداد بسیار بالا انجام دهیم تقریباً شانس پیمایش تمامی نود ها برابر ۱ خواهد بود بنابراین این الگوریتم یک الگوریتم کامل حساب میشود. ولی در عین حال کارامدی پایینی دارد زیرا زمان بسیار زیادی نیاز است تا شانس تمامی نود ها به ۱ میل کند. در سوی دیگر الگوریتم جستجوی تپه نوردی الگوریتم کارامدی به شمار میرود با این تفاوت که تمامی نود ها شانس پیمایش شدن را ندارند و احتمال گیر کردن در ماکسیمم های نسبی به نسبت بسیار زیاد است

الگوریتم ذوب فلزات ترکیبی از این دو الگوریتم است که شانس رهایی از نقاط ماکسیمم نسبی را افزایش میدهد. روند الگوریتم به شکل زیر است

یک حالت را به عنوان حالت فعلی در نظر بگیر

هر بار یکی از همسایه ها را به شکل تصادفی انتخاب کن

اگر حالت انتخاب شده بهتر بود آن را جایگزین و در غیر این صورت به احتمال مشخصی آن را جایگزین کن.

function SIMULATED-ANNEALING(*problem, schedule*) **returns** a solution state

inputs: *problem*, a problem

schedule, a mapping from time to “temperature”

current \leftarrow MAKE-NODE(*problem*.INITIAL-STATE)

for $t = 1$ **to** ∞ **do**

$T \leftarrow$ *schedule*(t)

if $T = 0$ **then return** *current*

next \leftarrow a randomly selected successor of *current*

$\Delta E \leftarrow$ *next*.VALUE – *current*.VALUE

if $\Delta E > 0$ **then** *current* \leftarrow *next*

else *current* \leftarrow *next* only with probability $e^{\Delta E/T}$

حال برای مقایسه دو الگوریتم موجود آن ها را پیاده سازی کرده ایم و هربار با دیتاهایی خاص آن ها را مورد ارزیابی قرار میدهم تا تفاوت آن ها را در عمل نیز مشاهده کنیم.

لینک الگوریتم های پیاده سازی شده:

<https://github.com/amirradjou/HillTSP>

<https://github.com/amirradjou/SimulatedAnnealingTSP>

بخشی از کدها:

```

10     new_neighbor = swapPositions(route, i, j).copy()
11     my_neighbors.append(new_neighbor)
12
13     return my_neighbors
14
15 n = int(input("Please Insert the number of cities: "))
16
17 cities = generate_random_cities(n)
18 # for i in range(0, n):
19 #     print(cities[i])
20
21 random_route = generate_random_route(n)
22 # print(random_route)
23
24 cost = calculate_cost(random_route)
25 # print(cost)
26
27 # neighbors = generate_neighbors(random_route, n)
28 # print(neighbors)
29
30 best_route = random_route
31 steps = 0
32 tic = time.time()
33 while True:
34     cost = calculate_cost(best_route)
35     neighbors = generate_neighbors(best_route, n)
36     best_cost = cost
37     for route in neighbors:
38         new_route_cost = calculate_cost(route)
39         if new_route_cost < best_cost:
40             best_route = route.copy()
41             best_cost = new_route_cost
42     steps += 1
43     if best_cost == cost:
44         break
45
46 toc = time.time()
47 sec = toc - tic
48 print("Best Route is: " + str(best_route))
49 print("Best Cost is: " + str(best_cost))
50 print("This Algorithm Solved this Problem in {steps} steps")
51 print("We reached this route in {sec} second")

```

```

10 def calculate_cost(route):
11     cost = 0
12     for i in range(0, n):
13         city_A = route[i - 1]
14         city_B = route[i]
15         cost = cost + cities[city_A][city_B]
16     return cost
17
18
19 def generate_random_route(number_of_cities):
20     l = list(range(number_of_cities))
21     random.shuffle(l)
22     return l
23
24
25 def generate_random_cities(number_of_cities):
26     cities = [[0 for i in range(number_of_cities)] for j
27               for i in range(0, number_of_cities):
28                   for j in range(0, number_of_cities):
29                       if i > j:
30                           random_number = random.randint(1, 50)
31                           cities[i][j] = random_number
32                           cities[j][i] = random_number
33     return cities
34
35
36 def generate_neighbors(route, n):
37     my_neighbors = []
38     for i in range(0, n - 1):
39         for j in range(i + 1, n):
40             new_neighbor = swapPositions(route, i, j).copy()
41             my_neighbors.append(new_neighbor)
42
43     return my_neighbors
44
45
46 n = int(input("Please Insert the number of cities: "))
47
48 cities = generate_random_cities(n)
49 # for i in range(0, n):
50 #     print(cities[i])
51
52 random_route = generate_random_route(n)
53 # print(random_route)

```

ابتدا برای ۱۰۰ شهر و با محدودین زمانی نیم ثانیه الگوریتیم جستجوی تپه نوردی را انجام میدهم:

Best Cost is: 1636 This Algorithm Solved this Problem in 10 steps We reached this route in 0.5311064720153809 second ----- Best Cost is: 1916 This Algorithm Solved this Problem in 6 steps We reached this route in 0.5131685733795166 second ----- Best Cost is: 1971 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5541098117828369 second ----- Best Cost is: 1867 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5277588367462158 second ----- Best Cost is: 2097 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5354354381561279 second ----- Best Cost is: 2082 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5378878116607666 second ----- Best Cost is: 2035 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5350179672241211 second ----- Best Cost is: 2174 This Algorithm Solved this Problem in 5 steps We reached this route in 0.543032169342041 second ----- Best Cost is: 2058 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5459320545196533 second ----- Best Cost is: 2122 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5365808857543945 second ----- Best Cost is: 2129	----- Best Cost is: 2129 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5442805290222168 second ----- Best Cost is: 1950 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5481741428375244 second ----- Best Cost is: 2111 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5404665470123291 second ----- Best Cost is: 1975 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5655083656311035 second ----- Best Cost is: 2010 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5671465396881104 second ----- Best Cost is: 2163 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5466809272766113 second ----- Best Cost is: 2069 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5593655109405518 second ----- Best Cost is: 2176 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5433987348937988 second ----- Best Cost is: 1898 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5409914225006104 second ----- Best Cost is: 2247 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5307285785675049 second -----	We reached this route in 0.5523529052734375 second ----- Best Cost is: 1857 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5648212432861328 second ----- Best Cost is: 2267 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5696678161621094 second ----- Best Cost is: 2053 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5845704078674316 second ----- Best Cost is: 2034 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5571672916412354 second ----- Best Cost is: 1958 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5595943927764893 second ----- Best Cost is: 1852 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5461938381195068 second ----- Best Cost is: 2015 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5272688865661621 second ----- Best Cost is: 2028 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5320403575897217 second ----- Best Cost is: 2376 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5350849628448486 second ----- Best Cost is: 1986 This Algorithm Solved this Problem in 5 steps We reached this route in 0.5423920154671533 second
--	---	--

حال با همین شرایط الگوریتم ذوب فلزات را پیاده سازی میکنیم:

Best Cost: 2281 47 steps 0.5015931129455566 second Final T is: 95.40649618417363 ----- Best Cost: 2547 44 steps 0.5000400543212891 second Final T is: 95.69328906720132 ----- Best Cost: 2492 49 steps 0.507495641708374 second Final T is: 95.21577859830147 ----- Best Cost: 2694 46 steps 0.5042591094970703 second Final T is: 95.50199818235599 ----- Best Cost: 2626 47 steps 0.5003345012664795 second Final T is: 95.40649618417363 ----- Best Cost: 2289 49 steps 0.5015068054199219 second Final T is: 95.21577859830147 ----- Best Cost: 2447 45 steps 0.5065910816192627 second Final T is: 95.59759577813412 ----- Best Cost: 2288 45 steps 0.5015971660614014 second Final T is: 95.59759577813412 ----- Best Cost: 2370 44 steps 0.5019724369049072 second Final T is: 95.69328906720132 ----- Best Cost: 2427 42 steps 0.5058660507202148 second Final T is: 95.88496310845512 ----- Best Cost: 2573 47 steps 0.5052700842724609 second Final T is: 95.40649618417363 ----- Best Cost: 2723 47 steps 0.5054500102996826 second Final T is: 95.40649618417363 ----- Best Cost: 2419 44 steps 0.5050156116485596 second Final T is: 95.69328906720132 ----- Best Cost: 2790 46 steps 0.50539231300354 second Final T is: 95.50199818235599 ----- Best Cost: 2724 47 steps 0.5072920322418213 second Final T is: 95.40649618417363 ----- Best Cost: 2273 44 steps 0.5025434494018555 second Final T is: 95.69328906720132 -----	----- Best Cost: 2589 44 steps 0.5199556350708008 second Final T is: 95.69328906720132 ----- Best Cost: 2658 45 steps 0.507697582244873 second Final T is: 95.59759577813412 ----- Best Cost: 2491 48 steps 0.5011248588562012 second Final T is: 95.31108968798947 ----- Best Cost: 2514 46 steps 0.502312421798706 second Final T is: 95.50199818235599 ----- Best Cost: 2606 49 steps 0.5022199153900146 second Final T is: 95.21577859830147 ----- Best Cost: 2504 51 steps 0.5032942295074463 second Final T is: 95.02544225688347 ----- Best Cost: 2794 98 steps 0.5002269744873047 second Final T is: 90.66044494080761 ----- Best Cost: 2124 99 steps 0.5029034614562988 second Final T is: 90.5697844958668 -----	main x ----- Best Cost: 2666 44 steps 0.5019724369049072 second Final T is: 95.69328906720132 ----- Best Cost: 2427 42 steps 0.5058660507202148 second Final T is: 95.88496310845512 ----- Best Cost: 2573 47 steps 0.5052700842724609 second Final T is: 95.40649618417363 ----- Best Cost: 2723 47 steps 0.5054500102996826 second Final T is: 95.40649618417363 ----- Best Cost: 2419 44 steps 0.5050156116485596 second Final T is: 95.69328906720132 ----- Best Cost: 2790 46 steps 0.50539231300354 second Final T is: 95.50199818235599 ----- Best Cost: 2724 47 steps 0.5072920322418213 second Final T is: 95.40649618417363 ----- Best Cost: 2273 44 steps 0.5025434494018555 second Final T is: 95.69328906720132 -----
---	--	---

مشاهده میکنیم تعداد مراحل انجام شده در الگوریتم ذوب فلزات چندین مرتبه بیشتر از الگوریتم تپه نوردی در زمان و تعداد شهر یکسان میباشد ولی در عین حال مسیر یافته شده توسط الگوریتم تپه نوردی با فاصله تقریباً کمی بهتر میباشد اما تفاوت چندانی دیده نمیشود.

حال این الگوریتم ها را در زمان کمتر و با تعداد شهر کمتر تست میکنیم. تعداد شهر ها را به ۵۰ و مدت زمان را به ۲ دهم ثانیه کاهش میدهیم.

الگوریتم تپه نوردی:

Best Cost is: 320 This Algorithm Solved this Problem in 25 steps We reached this route in 0.1546180248260498 second -----	Best Cost is: 506 This Algorithm Solved this Problem in 16 steps We reached this route in 0.21324968338012695 second -----
Best Cost is: 372 This Algorithm Solved this Problem in 21 steps We reached this route in 0.1264963150024414 second -----	Best Cost is: 454 This Algorithm Solved this Problem in 15 steps We reached this route in 0.20244836807250977 second -----
Best Cost is: 437 This Algorithm Solved this Problem in 19 steps We reached this route in 0.11559700965881348 second -----	Best Cost is: 436 This Algorithm Solved this Problem in 15 steps We reached this route in 0.20058345794677734 second -----
Best Cost is: 331 This Algorithm Solved this Problem in 20 steps We reached this route in 0.12046408653259277 second -----	Best Cost is: 441 This Algorithm Solved this Problem in 15 steps We reached this route in 0.2039649486541748 second -----
Best Cost is: 369 This Algorithm Solved this Problem in 26 steps We reached this route in 0.1568922996520996 second -----	Best Cost is: 446 This Algorithm Solved this Problem in 14 steps We reached this route in 0.20929360389709473 second -----
Best Cost is: 289 This Algorithm Solved this Problem in 30 steps We reached this route in 0.1821727752685547 second -----	Best Cost is: 459 This Algorithm Solved this Problem in 15 steps We reached this route in 0.21041226387023926 second -----
Best Cost is: 328 This Algorithm Solved this Problem in 22 steps We reached this route in 0.15849041938781738 second -----	Best Cost is: 407 This Algorithm Solved this Problem in 16 steps We reached this route in 0.21251821517944336 second -----
Best Cost is: 345 This Algorithm Solved this Problem in 27 steps We reached this route in 0.16392135620117188 second -----	Best Cost is: 422 This Algorithm Solved this Problem in 15 steps We reached this route in 0.20260262489318848 second -----
Best Cost is: 326 This Algorithm Solved this Problem in 24 steps We reached this route in 0.21031641960144043 second -----	Best Cost is: 483 This Algorithm Solved this Problem in 15 steps We reached this route in 0.20811128616333008 second -----
Best Cost is: 573 This Algorithm Solved this Problem in 16 steps We reached this route in 0.21312713623046875 second -----	Best Cost is: 380 This Algorithm Solved this Problem in 15 steps We reached this route in 0.2008826732635498 second -----

الكوريتم ذوب فلزات:

```

main x
Best Cost: 1185
340 steps
0.20024538040161133 second
Final T is: 71.16492513781145
-----
Best Cost: 989
360 steps
0.20037579536437988 second
Final T is: 69.75506718651023
-----
Best Cost: 1338
351 steps
0.200331449508667 second
Final T is: 70.38601331341707
-----
Best Cost: 1208
209 steps
0.20097041130065918 second
Final T is: 81.1310392707735
-----
Best Cost: 1277
164 steps
0.20027446746826172 second
Final T is: 84.86723814588913
-----
Best Cost: 1244
163 steps
0.20091938972473145 second
Final T is: 84.95219033622536
-----
Best Cost: 1246
164 steps
0.20097756385803223 second
Final T is: 84.86723814588913
-----
Best Cost: 1260
162 steps
0.20038270950317383 second
Final T is: 85.03722756378914
-----
Best Cost: 1307
163 steps
0.20094537734985352 second
Final T is: 84.95219033622536
-----
Best Cost: 1408
162 steps
0.20003128051757812 second
Final T is: 85.03722756378914
-----
Best Cost: 1178
162 steps
0.2001791000366211 second
Final T is: 85.03722756378914
-----
Best Cost: 1591
161 steps
0.20070767402648926 second
Final T is: 85.12234991370285
-----
Best Cost: 1255
162 steps
0.20072698593139648 second
Final T is: 85.03722756378914

```

مشاهده میشود که در این حالت تفاوت در بهترین مسیر انتخابی بسیار قابل توجه خواهد شد بنابراین میتوان نتیجه گرفت که الگوریتم تپه نوردی با اینکه نود های کمتری را پیمایش میکند اما در زمان و تعداد شهر کمتر بهتر از الگوریتم ذوب فلزات عمل میکند

این در حالیکه در صورت عدم وجود محدودیت زمانی الگوریتم ذوب فلزات میتواند مسیرهای بهتری را پیدا کند زیرا شانس عبور آن از ماکسیممهای نسبی بیشتر است.

References:

<https://towardsdatascience.com/how-to-implement-the-hill-climbing-algorithm-in-python-1c65c29469de>

جزوه هوش مصنوعی دکتر فاطمه موسوی