دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)
دانشکدهٔ ریاضی و علوم‌کامپیوتر

گزارش ۳: پیاده‌سازی روش‌های جستجوی محلی و تحلیل آن

نگارش
نیما حسینی دشت بیاض

استاد
دکتر مهدی قطعی

فروردین ۱۴۰۰

مقدمه

با بزرگ شدن اندازه‌ی مسئله‌ها، پیاده‌سازی روش‌های جستجوی ساده مانند BFS یا جستجوی همراه با هیوریستیک می‌تواند هزینه‌ی زیادی از نظر حافظه و زمان داشته باشد. در برخی از مسائل، صرفا جواب نهایی مورد نظر است و این‌که چه مسیری برای رسیدن به این جواب لازم است مطرح نیست. به‌طور مثال، در بازی شطرنج، مسیر رسیدن به جواب، که خارج کردن تمام مهره‌های حریف است، اهمیت دارد؛ اما در یک مسئله‌ی بهینه سازی صرفا جواب نهایی و بهینه برای ما مهم است. در چنین مسائلی می‌توان از الگوریتم‌های جستجوی محلی استفاده کرد.

این الگوریتم‌ها با رسیدن به یک گره در گراف مسئله، به‌جای بررسی تمام گره‌های مجاور، تنها یک گره را انتخاب و مورد بررسی قرار می‌دهند. انتخاب این گره در الگوریتمی مانند Random Walk تصادفی و در سایر الگوریتم‌ها می‌تواند براساس یک تابع هیوریستیک باشد. همچنین برای صرفه‌جویی در حافظه، گره‌های پیموده شده در مسیر ذخیره نمی‌شوند.

در ادامه دو مسئله‌ی Travelling Salesman و N-Queens را با استفاده از سه الگوریتم Hill Climbing, Local Beam Search و Simulated Annealing حل می‌کنیم و برای هر کدام، روش‌ها را تحلیل می‌کنیم.

مسئله‌ی N-Queens[۱] [۲]

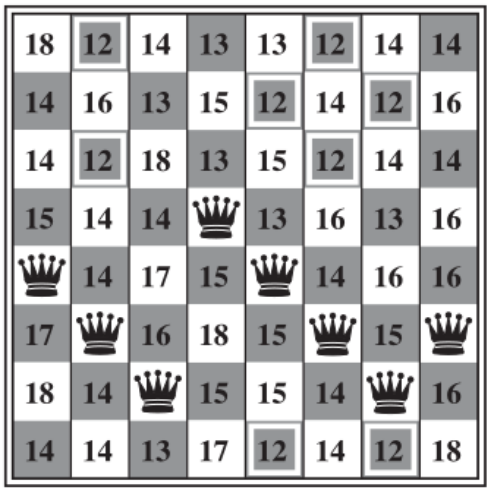
برای حل این مسئله با کمک الگوریتم‌های جستجوی محلی، لازم است که یک تابع هیوریستیک تعریف کنیم که در هر state، هزینه‌ی آن را مشخص کند. برای تعریف این تابع، تعداد جفت ملکه‌هایی که می‌توانند به هم دیگر حمله کنند را می‌شماریم. هر ملکه می‌تواند در چهار جهت بالا، پایین و دو قطرش حرکت کند. در این صورت اگر دو ملکه در یکی از جهت‌های مجاز هم‌دیگر قرار بگیرند، می‌توانند به هم دیگر حمله کنند. تعداد این جفت‌ها ملکه‌ها را به عنوان هیوریستیک در نظر می‌گیریم. همچنین از آنجایی که در جواب مسئله، در هر ستون فقط یک ملکه می‌تواند قرار داشته باشد، از ابتدا در هر ستون یک ملکه قرار می‌دهیم و فقط اجازه‌ی حرکت در همان ستون را ملکه‌ها می‌دهیم. در عکس ۱ نمونه‌ای از این هیوریستیک آورده شده است.[1]

در ادامه جواب به دست آمده از سه الگوریتم ذکر شده را بررسی می‌کنیم.

---

۱ https://en.wikipedia.org/wiki/Eight_queens_puzzle

۲ مسئله و روش حل آن به همراه هیوریستیک‌ها از کتاب Artificial Intelligence A Modern Approach, Third Edition برداشته شده است.



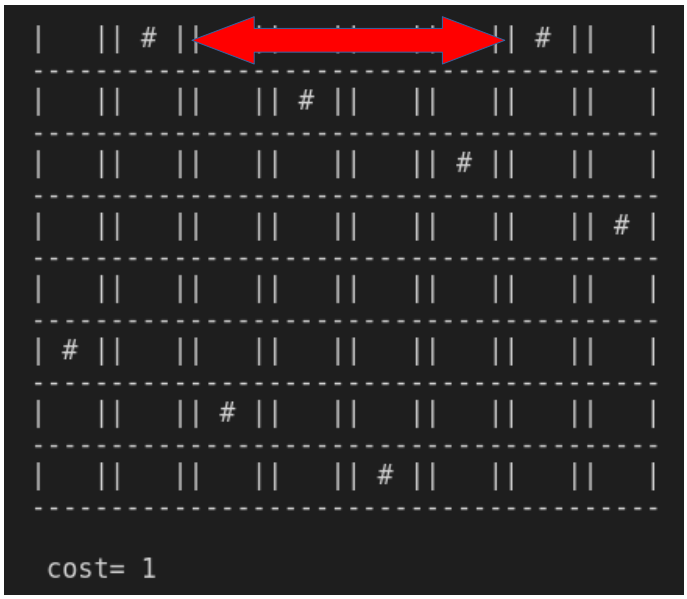
عکس ۱: مثالی از مقادیر هیوریستیک تعریف شده در صورت حرکت یک ملکه به خانه‌های بالا یا پایینش. مقدار هیوریستیک اولیه برابر ۱۸ است.

الگوریتم Hill Climbing

این الگوریتم در هر مرحله، بهترین همسایه‌ی یک گره را انتخاب کرده و بقیه‌ی همسایه‌ها را در نظر نمی‌گیرد. این الگوریتم را برای مسئله‌ی N Queens با اندازه‌های ۸، ۱۲ و ۲۰ اجرا کردیم. جایگیری اولیه ملکه‌ها در هر اجرا به طور رندوم مشخص شده و در هر ستون دقیقا یک ملکه قرار داشته است. همچنین برای هر اندازه، الگوریتم را ۱۰ بار تکرار کردیم تا عملکرد میانگین آن با ورودی‌های رندوم متفاوت به دست آید.

تصویر مقابل، نمونه‌ی خروجی این الگوریتم برای اندازه‌ی ۸ است. همانطور که مشخص شده، این الگوریتم در این مثال در رسیدن به جواب بهینه شکست خورده است و به جوابی با هزینه‌ی هیوریستیک ۱ رسیده است.

در جدول زیر خروجی الگوریتم را برای اندازه‌های دیگر نیز بررسی کرده‌ایم. مقدار مقادیر میانگین برای هر مسئله هم مشخص شده است.



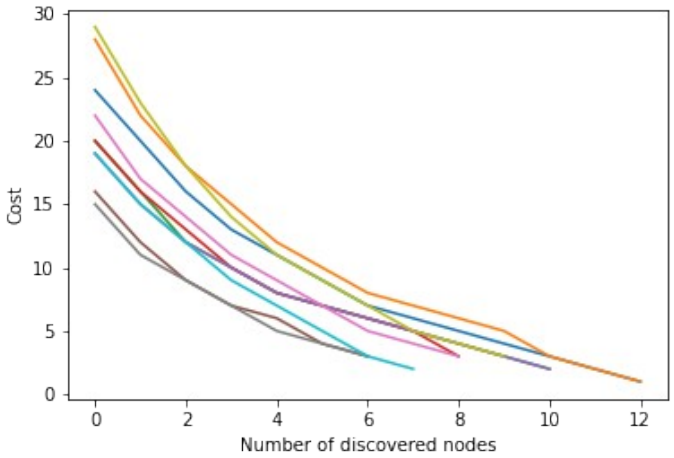
عکس ۲: نمونه خروجی الگوریتم Hill Climbing برای مسئله‌ی 8Queens

جدول ۱: میانگین خروجی‌های روش Hill Climbing برای اندازه‌های ۸، ۱۲ و ۲۰ پس از ۱۰ بار اجرا برای هر اندازه

اندازه	تعداد جواب بهینه در ۱۰ اجرا	میانگین تعداد گره‌های کشف شده تا توقف الگوریتم	میانگین زمان اجرای الگوریتم (میلی ثانیه)	میانگین هزینه‌ی هیوریستیک جواب نهایی (بهینه = ۰)
۸	۲	۲.۹	۳	۱.۴
۱۲	۱	۵	۱۷.۴	۱.۵
۲۰	۰	۸.۸	۱۷۵.۱	۲.۳

همانطور که در جدول مشخص است، در بیشتر مواقع این الگوریتم نمی‌تواند جواب بهینه را برای مسئله‌ی N-Queens پیدا کند. همچنین با بزرگتر شدن مسئله، شانس پیدا کردن جواب بهینه کاهش می‌یابد و همچنین میانگین هزینه‌ی جواب‌های پیدا شده نیز بیشتر می‌شود. این نشان می‌دهد که با بزرگتر شدن مسئله، تعداد مقادیر بهینه‌ی محلی[۳] افزایش می‌یابد و الگوریتم در موراد بیشتری در این چاله‌ها گیر می‌کند.

در نمودار زیر روند حرکت الگوریتم برای اندازه‌ی ۲۰ در هر کدام از ۱۰ تکرار مشخص شده است. هیچ‌کدام به جواب بهینه با هزینه‌ی صفر نرسیده‌اند.



عکس ۳: هزینه‌ی هیوریتسیک برای مسئله با اندازه‌ی ۲۰ در هر کدام از ۱۰ اجرای الگوریتم

بزرگ‌ترین مزیت این الگوریتم زمان کوتاه اجرا است که در جدول مشخص است. همچنین از آنجایی که در هر مرحله فقط یک گره در حافظه نگه‌داری می‌شود، مصرف حافظه نیز بهینه و اندک است.

۳ Local Optima

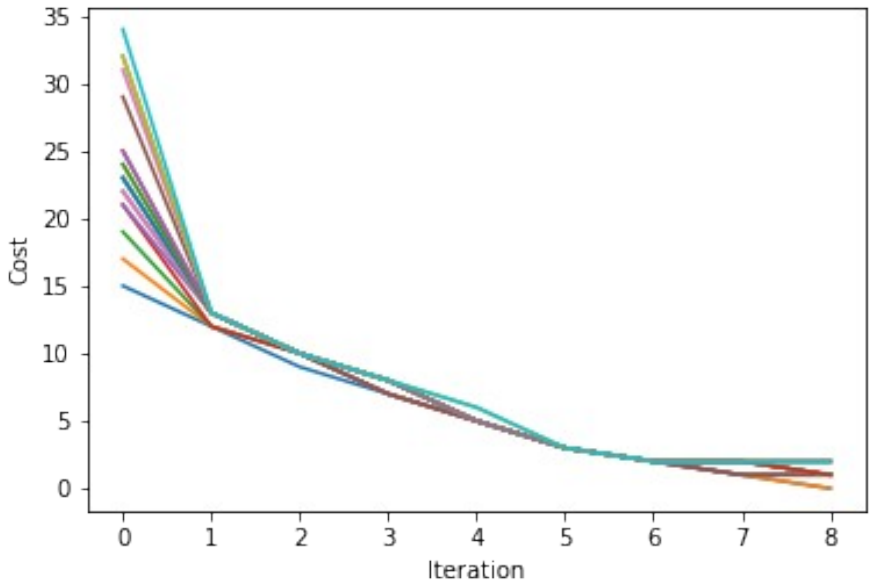
الگوریتم Local Beam Search

در این الگوریتم، برخلاف روش قبل که فقط یک گره را نگه می‌داشتیم، با تعداد مشخصی (k) گره اولیه شروع می‌کنیم و به طور موازی از هر کدام پیش‌روی می‌کنیم. برای این کار، ابتدا لازم است که k را مشخص کنیم. در این‌جا ما الگوریتم را به ازای $k = 10$ و $k = 20$ و مانند قسمت قبل برای اندازه‌های ۸، ۱۲ و ۲۰ اجرا می‌کنیم. همچنین مجدداً هر بار الگوریتم ۱۰ بار تکرار می‌شود تا میانگین به دست آید.

جدول ۲: نتایج حاصل از روش Local Beam Search برای مسئله‌ی N-Queens

تعداد گره k	اندازهٔ مسئله	تعداد جواب بهینه	میانگین تعداد گره کشف شده	میانگین زمان اجرا (میکرو ثانیه)	میانگین هزینه‌ی هیوریستیک
۱۰	۸	۵	۳۵	۲۹.۷	۰.۵
۱۰	۱۲	۴	۴۹	۱۵۰.۵	۰.۶
۱۰	۲۰	۲	۹۵	۲۰۱۷.۴	۱.۴
۲۰	۸	۱۰	۵۴	۳۹.۱	۰
۲۰	۱۲	۵	۱۰۸	۳۶۹.۴	۰.۶
۲۰	۲۰	۲	۱۸۴	۳۵۰۴	۱.۳

همان‌طور که مشاهده می‌شود، عملکرد این الگوریتم بسیار بهتر از Hill Climbing می‌باشد و با افزایش k بهتر هم می‌شود: اما این بهبود به قیمت زمان اجرا و حافظه‌ی بیشتر تمام می‌شود.



عکس ۴: نحوه‌ی کاهش هزینه‌ی جایگاه‌های ۱ تا k در یک بار اجرای الگوریتم Beam Search برای مسئله با اندازه‌ی ۲۰ و k=۲۰.

الگوریتم Simulated Annealing

در این روش با استفاده از یک روش stochastic، این امکان بوجود می‌آید که در برخی مراحل یک گره با هزینه‌ی بیشتر از گرهٔ فعلی انتخاب شود. این کار امکان فرار کردن از مینیمم‌های محلی را به الگوریتم می‌دهد. پیاده‌سازی الگوریتم بر اساس شبهه کد ارائه شده در کتاب است. همچنین برای محاسبه‌ی دما[۴] در هر مرحله، دمای مرحله‌ی قبل را بر یک عدد کوچک بزگتر از صفر مانند ۰.۰۰۱ تقسیم میکنیم.

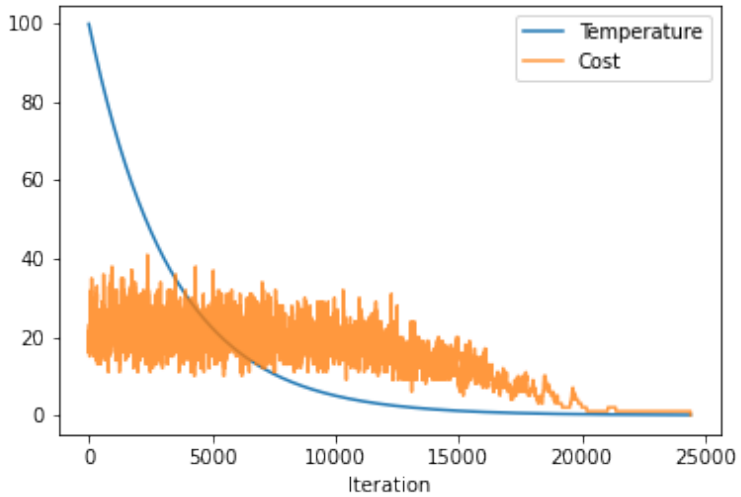
در این‌جا نیز مانند قبل این الگوریتم را برای مسئله اجرا کردیم که نتایج آن در جدول آمده است. البته این الگوریتم نیازمند پارامترهای مختلفی برای تعیین دمای اولیه، ضریب کاهش دما، حداکثر تعداد تکرار حلقه و حداقل دمای توقف می‌باشد. نتایج پایین بعد از امتحان پارامترهای مختلف به دست آمده است. در این جدول، نتایج با دمای اولیه‌ی ۱۰۰، ضریب کاهش دمای ۱.۰۰۰۳، حداکثر تکرار ۱۰۰,۰۰۰ و حداقل دمای ۰.۰۰۰۱ حاصل شده است.

جدول ۳: نتایج حاصل از روش Simulated Annealing برای مسئله‌ی N-Queens

اندازهٔ مسئله	تعداد جواب بهینه	میانگین تعداد گره کشف شده	میانگین زمان اجرا (میکرو ثانیه)	میانگین هزینه‌ی هیوریستیک
۸	۱۰	۱۲۹۵۸.۲	۴۶۰	۰
۱۲	۱۰	۱۴۹۷۰.۱	۱۲۰۲.۲	۰
۲۰	۱۰	۱۵۱۱۳.۷	۳۴۵۸	۰

مشخص است که این الگوریتم عملکرد بسیار بهتری از دو الگوریتم قبلی دارد؛ هرچند تعداد گره‌های بسیار بیشتری را بازدید می‌کند و زمان بسیار بیشتر هم می‌گیرد.

در شکل مقابل روند کاهش دما و هزینه در این الگوریتم برای n=20 مشخص شده است. در این نمودار انتخاب گره‌هایی با هزینه‌ی بیشتر به خوبی دیده می‌شود.



عکس ۵: روند کاهش دما و هزینه در الگوریتم Simulated Annealing برای مسئله N-Queens با اندازه‌ی ۲۰.

۴ Temperature

مسئله‌ی Traveling Salesman[۵] [۶]

در این مسئله باید ترتیبی از پیمایش شهر ها با کم‌ترین هزینه را پیدا کنیم. یک تابع هزینه‌ی بدیهی برای این مسئله، مجموع فواصلی است که فروشنده باید برای انجام ماموریتش طی کند. پس همین تابع را به عنوان هیوریستیک در نظر می‌گیریم. همچنین مسئله را با شروع از یک ترتیب رندوم از شهرها شروع می‌کنیم و برای پیدا کردن گره‌های همسایه، هر بار جای دو شهر را در این ترتیب عوض می‌کنیم.[2]

در این مسئله از دیتاست‌های TSPLIB و به‌طور مشخص دیتاست DANTZIG42 استفاده شده است که شامل ۴۲ شهر است. جواب بهینه در این دیتاست مسیری به مسافت 699 است.[3] در تمام روش‌های زیر، الگوریتم ۴ بار اجرا شده است و نتایج نمایش داده شده است.

روش Hill Climbing

جدول ۴: نتایج حاصل از روش Hill Climbing برای مسئله‌ی TSP

تکرار	هزینه‌ی نهایی	تعداد گره کشف شده	زمان اجرا (میکرو ثانیه)
۱	۱۰۵۸	۳۳	۱۶۵۶
۲	۹۲۴	۳۳	۱۷۵۰
۳	۹۸۳	۳۰	۱۵۴۴
۴	۱۰۵۲	۳۵	۱۷۶۶

روش Beam Search

این نتایج با اسفاده از k=5 به دست آمده است.

جدول ۵: نتایج حاصل از روش Beam Search برای مسئله‌ی TSP

تکرار	هزینه‌ی نهایی	تعداد گره کشف شده	زمان اجرا (میکرو ثانیه)
۱	۹۴۸	۱۵۵	4979
۲	۸۹۰	۱۹۵	۹۹۵۹
۳	۹۳۹	۱۶۵	۹۷۶۷
۴	۱۱۱۶	۱۲۵	۷۰۷۰

۵ https://en.wikipedia.org/wiki/Travelling_salesman_problem

۶ ایده‌ی حل این مسئله و منبع آن توسط دوست من، امیر بابامحمودی، به من منتقل شد.

روش Simulated Annealing

این روش با دمای اولیه 120 و ضریب کاهش هزینه‌ی 1.0002 اجرا شد و نتایج زیر به دست آمد.

جدول ۶: نتایج حاصل از روش Simulated Annealing برای مسئله‌ی TSP

تکرار	هزینه‌ی نهایی	تعداد گره کشف شده	زمان اجرا (میکرو ثانیه)
۱	۷۲۵	۴۳۷۱	۴۹۷۹
۲	۸۱۵	۴۳۵۱	۵۱۹۱
۳	۸۸۵	۴۸۰۲	۵۳۶۹
۴	۹۰۳	۴۵۶۱	۴۹۸۱

جمع‌بندی

در این مسئله هم مانند N-Queens می‌بینیم که الگوریتم Simulated Annealing جواب بهتری از هر دو الگوریتم دیگر پیدا می‌کند. همچنین زمان اجرا و مصرف حافظه‌ی آن نیز از الگوریتم Beam Search بهتر است.

جزئیات پیاده سازی

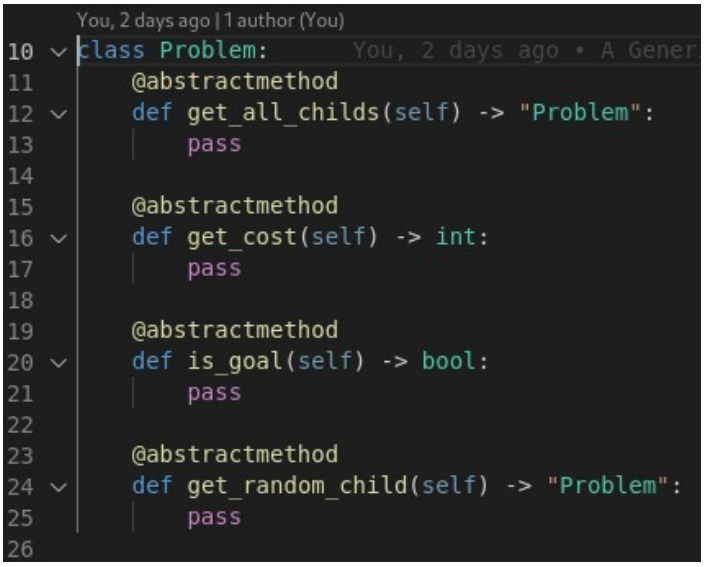
پیاده سازی کامل در گیت‌هاب قرار داده شده است. در پایین فقط ساختار پروژه مشخص شده است.

https://github.com/nimahsn/AI-Course-Projects/tree/main/report02-LocalSearch

پروژه شامل دو فایل پایتون و یک فایل notebook می‌باشد. فایل‌های problems.py و solver.py به ترتیب شامل مدل‌های مسئله‌ها و پیاده‌سازی الگوریتم‌ها هستند و در نوت‌بوک این پیاده سازی ها اجرا شده و نتایج این گزارش به دست آمده است.

Problems.py

این فایل شامل سه کلاس است. کلاس Problem یک کلاس abstract است. در این کلاس چهار تابع وجود دارد که فرزندان این کلاس باید آن‌ها را پیاده کنند. این توابع برای اجرای الگوریتم‌های جستجو ضروری است.



کلاس‌هـای Nqueens و TravellingSalesPerson بـرای مدل‌سـازی این دو مسـئله سـاخته شـده‌اند و کلاس problem را به ارث می‌برند. قسمت عمده‌ی این کلاس‌ها، پیاده سازی توابع فوق است. همچنین کلاس TSP دارای یـک تابع به نام load_from_file می‌باشد که برای ایجاد مدل مسئله از روی فایل دیتاست TSPLIB نوشته شده است. هر دوی این کلاس‌ها در سازنده‌ی خود (__init__)، در صورتی که جوابی به آن‌ها ارائه نشده باشد، یک جواب اولیه رندوم برای مسئله می‌سازند. در Nqueens با قرار دادن هر ملکه در یک ستون در یک ردیف تصادفی و در مسئلهي TSP با ایجاد یک ترتیب تصادفی از شهرها این جواب اولیه ساخته می‌شود.

Solver.py[۷] [۸]

این فایـل شـامل پیـاده سـازی هـر ۳ الگـوریتم، بـه علاوه‌ی یـک کلاس بـرای تهیه‌ی log از اجـرای الگوریتم‌هـا می‌باشد. ساختار فایل در عکس زیر مشخص است.

۷ الگوریتم‌های HillClimb و SimulatedAnnealing بر اساس شبهه کدهای داخل کتاب نوشته شده‌اند. همچنین تابع schedule کلاس SA براساس روش‌های پایین آوردن Learning Rate که در یادگیری ماشین کاربرد دارد طراحی شده است.

۸ الگوریتم BeamSearch بر اساس متن کتاب و با استفاده از منبع زیر طراحی شده است، گرچه که روش پیاده سازی تفاوت زیادی دارد.

https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/

هر ۳ کلاس شامل یک تابع solve می‌باشند که با دریافت یک نمونه از کلاس problem، آن را حل می‌کنند. همچنین برخی پارامتر های دیگر در BeamSearch و SimulatedAnnealling قابل تنظیم است. علاوه بر این تابع و تابع سازنده، کلاس SimulatedAnnealing دارای یک تابع scheduler است که برای تنظیم دما در هر iteration نوشته شده است.

Runner.ipynb

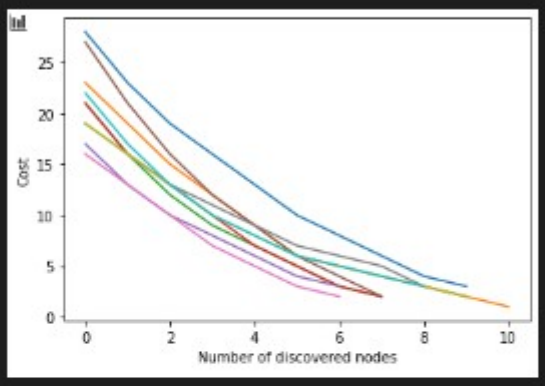
این نوت‌بوک برای اجرای الگوریتم‌ها و گرفتن نتایج نوشته شده است. تمام داده‌های داخل این گزارش و همچنین نمودارها در این نوت‌بوک به دست آمده است. کافیست تمام cell های آن را مجددا اجرا کنید تا مسائل داخل این گزارش حل شده و جواب هر کدام چاپ شود.

```
|  ||  ||  ||  ||  ||  ||  ||  ||  ||  ||  || # ||  ||  ||
```

[19]

```
for log in logs:
    plt.plot(log.costs)
plt.xlabel("Number of discovered nodes")
plt.ylabel("Cost")
```

```
Text(0, 0.5, 'Cost')
```



Beam Search

[25]

```
k = 10
beam_search = BeamSearch(beam=k, save_log= True)
n=8
logs = []
for _ in range(repeats):
    probs = [NQueens(n) for _ in range(k)]
    print(beam_search.solve(probs))
    logs.append(beam_search.log)
for log in logs:
    print(log.costs[-1])
print(np.mean([log.depth*k for log in logs]))
print(np.mean([log.elapsed_time//1e+6 for log in logs]))
print(np.mean([log.costs[-1][0] for log in logs]))
```

منابع

۱. Artificial Intelligence A Modern Approach, Third Edition, Stuart J. Russell and Peter Norvig.

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

۳ https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html