دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)
دانشکده ریاضی و علوم کامپیوتر

پایان‌نامه کارشناسی‌ارشد
گرایش ریاضی

هوش مصنوعی - گزارش ۰۲ - پیاده سازی الگوریتم های هیوریستیک روی مسئله مجموعه‌ی غالب

پایان‌نامه

نگارش
آترین حجت

استاد راهنما
نام کامل استاد راهنما

استاد مشاور
نام کامل استاد مشاور

فروردین ۱۴۰۰


بسم الله الرحمن الرحيم

صفحه فرم ارزیابی و تصویب پایان نامه- فرم تأیید اعضاء کمیته دفاع

در این صفحه فرم دفاع یا تایید و تصویب پایان نامه موسوم به فرم کمیته دفاع- موجود در پرونده آموزشی- را قرار دهید.

نکات مهم:

- نگارش پایان نامه/رساله باید به زبان فارسی و بر اساس آخرین نسخه دستورالعمل و راهنمای تدوین پایان نامه های دانشگاه صنعتی امیرکبیر باشد.(دستورالعمل و راهنمای حاضر)
- رنگ جلد پایان نامه/رساله چاپی کارشناسی، کارشناسی ارشد و دکترا باید به ترتیب مشکی، طوسی و سفید رنگ باشد.
- چاپ و صحافی پایان نامه/رساله بصورت پشت و رو(دورو) بلامانع است و انجام آن توصیه می شود.

دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)

به نام خدا

تعهدنامه اصالت اثر

تاریخ: فروردین ۱۴۰۰

اینجانب **آترین حجت** متعهد می‌شوم که مطالب مندرج در این پایان‌نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان‌نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان‌نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

آترین حجت

امضا

فهرست مطالب

عنوان صفحه

فهرست اشکال

شکل صفحه

فهرست جداول

جدول صفحه

فهرست نمادها

نماد	مفهوم
$n(G)$	تعداد رئوس گراف G
$deg_G(v)$	درجه‌ی راس v در گراف G
$nei(v)$	همسایه‌های راس v در گراف

فصل اول

شرح مسئله و روند کار

۱-۱ مقدمه

مسئله مجموعه‌ی غالب مسئله انتخاب رئوسی از گراف است بطوری که هر راس یا انتخاب شده باشد یا همسایه‌ای انتخاب‌شده داشته باشد. این مسئله NP Complete و دارای الگوریتم تخمین می‌باشد. این مسئله دارای کاربردهای بسیاری در زمینه‌هایی مانند تئوری شبکه‌های اجتماعی [۱] ، شبکه‌های ارتباطی کامپیوترها [۲] ، شبکه‌های بیسیم ادهاک [۳] و شبکه‌های حسگر بیسیم [۴] می‌باشد. [۱]

در این گزارش این این مسئله را با استفاده از راه‌حل‌های Heuristic حل خواهیم کرد.

۱-۲ کدها و خروجی‌های برنامه

تمام کدها و خروجی‌ها (از جمله کد `latex`) در اینجا قابل مشاهده می‌باشد.

[1] Social network theory
[2] Computer communication network
[3] Mobile Ad-hoc network - MANET
[4] Wireless sensor network - WSN

فصل دوم

توابع ابتدایی

۲-۱ تولید گراف تصادفی

برای تولید گراف تصادفی دو تابع مورد استفاده قرار گرفته است.[1] تابع `gen_graph_eq_prob_edges` گرافی با $nodes$ راس تولید میکند که هر یال با احتمال p در آن حضور دارد.

```
def gen_graph_eq_prob_edges(nodes=100, p=0.1):
    """
    Generates a graph with $nodes vertices where each edge has a $p
                                            probibility of existing.
    The graph is represented by a list of vertices each represented as a list
                                            of vertices it's connected to.
    O(nodes ^ 2)
    """
    graph =[[]for i in range(nodes)]

    for i in range(nodes):
        for j in range(i):
            if random.random() <p:
                graph[i].append(j)
                graph[j].append(i)
    return graph
```

تابع `gen_graph_fix_set_edges` گرافی با $nodes$ راس و $edges$ یال تصادفی می‌کند.

```
def gen_graph_fix_set_edges(nodes=100, edges=900):
    """
    Generates a graph with $nodes vertices and $edges edges
    The graph is represented by a list of vertices each represented as a list
                                            of vertices it's connected to.
    O(nodes ^ 2)
    """
```

[1] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/blob/main/Report%2002/Problem%2001%20-%20Vertext%20Cover/generators/__init__.py

```
    graph =[[]for i in range(nodes)]

    all_edges =[]

    for i in range(nodes):
        for j in range(i):
            all_edges.append((i, j, ))

    for i in range(edges):
        x =random.randrange(0, len(all_edges))
        edge =all_edges[x]
        graph[edge[0]].append(edge[1])
        graph[edge[1]].append(edge[0])
        all_edges.pop(x)

    return graph
```

۲-۲ بررسی هدف

تابع utils [۲] در solutions روی تمام یال های رئوس انتخاب شده می‌گردد و رئوس دیده‌شده را علامت می‌زند. زمان اجرای این تابع $O(EV)$ و حافظه‌ی آن $O(V)$ می‌باشد. به دلیل محدود بودن کل عملایات های چک کردن حالت بهینه به NP برای بهینه‌سازی این توابع تلاشی نشده‌است.

```
import math
import heapq
import random

def is_goal(graph, state):
    ls ={}
```

[2] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/blob/main/Report%2002/Problem%2001%20-%20Vertext%20Cover/solutions/utils.py

```
    for i in range(len(graph)):
        ls[i] =""
    for v in state:
        if v in ls:
            del ls[v]
        for u in graph[v]:
            if u in ls:
                del ls[u]
    return len(ls) ==0, [*ls.keys()], ls
```

۲-۳ تصویر سازی

برای نمایش دادن نمودارهای برنامه از matplotlib [۳] و برای گراف‌ها از networkx [۴] استفاده شده. [۵]

[3] https://matplotlib.org/

[4] https://networkx.org/documentation/stable/

[5] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/tree/main/Report%2002/Problem%2001%20-%20Vertext%20Cover/visualizers

فصل سوم

الگوریتم‌ها

۱-۳ $A*$

A* [۱] [۲] نوعی خاصی از Best-first search است که در تابع تخمین وزن، وزن مسیر تا آن لحظه نیز محاسبه می‌شود.

۱-۱-۳ تخمین و روش محاسبه

فرض کنید G گراف مورد نظر ما باشد. در حالتی که در آن مجموعه $C = v_1, ...v_t$ انتخاب شده‌اند برای محاسبه تابع Heuristic از فرمول

$$h(C) = \frac{n(G - C - nei(C))}{\max\limits_{v \in G-C-nei(C)} deg_{G-C-nei(C)}(v) + 1} \quad (۱-۳)$$

$$f(C) = h(C) + n(C) \quad (۲-۳)$$

استفاده میکنیم

۲-۱-۳ بررسی Admissibility

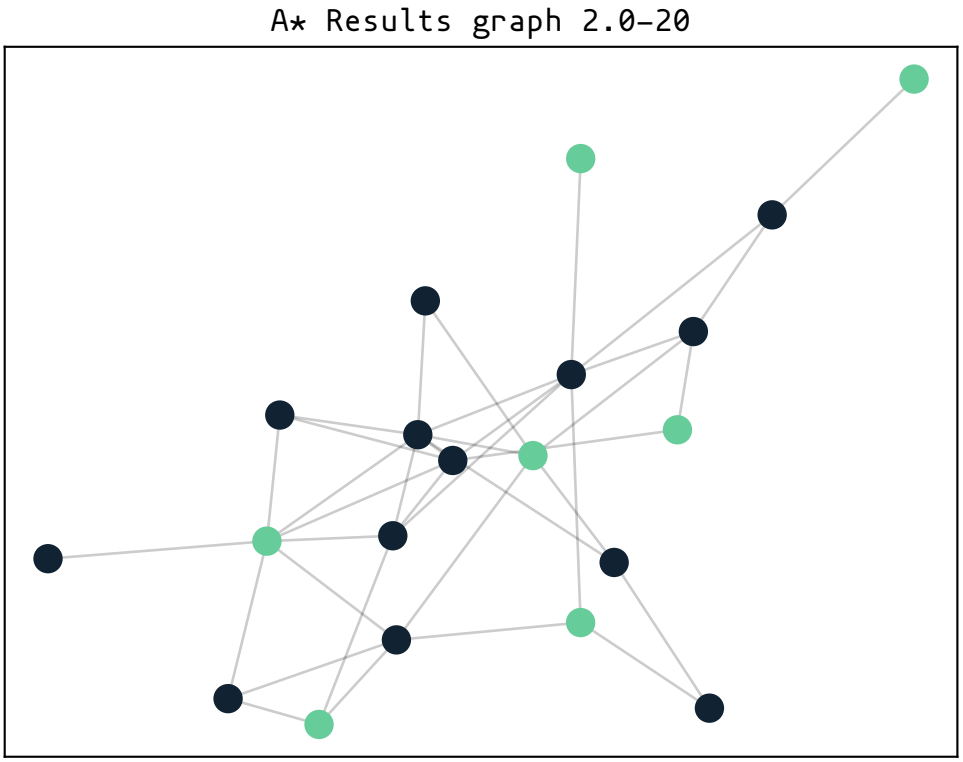
به وضوح، انتخاب هر راس انتخاب نشده حداکثر به اندازه‌ی درجه‌اش در گراف باقی مانده، راس جدید را پوشش میدهد. درنتیجه اگر درجه‌ی همه‌ی رئوسی که از این نقطه انتخاب می‌کنیم، برابر حداکثر درجه‌ی گراف باقی مانده باشد و هیچ راسی را دوبار پوشش ندهیم، $h(C)$ راس دیگر باید انتخاب شود تا همه‌ی رئوس گراف پوشیده شده‌باشد. پس بوضوح داریم :

$$h(C) \leq h^*(C) \quad (۳-۳)$$

یعنی تابع هیوریستیکمان Admissible می‌باشد پس می‌توان نتیجه گرفت که A* جواب بهینه می‌دهد و درنتیجه NP است.

[1] https://en.wikipedia.org/wiki/A*_search_algorithm

[2] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/blob/main/Report%2002/Problem%2001%20-%20Vertext%20Cover/solutions/AStar.py

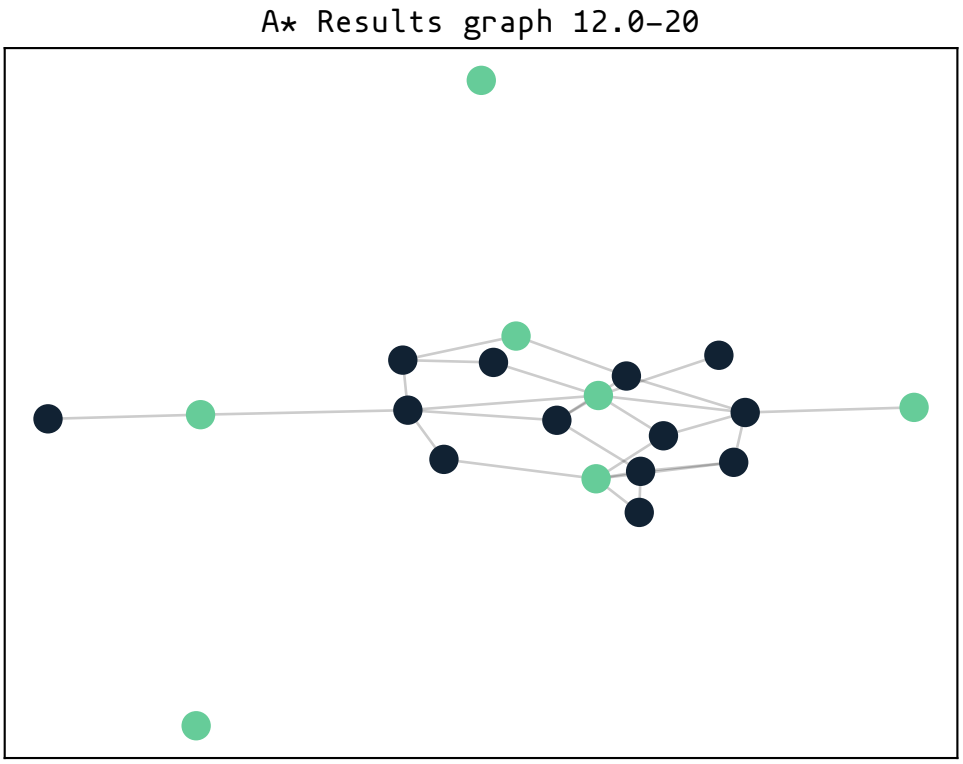


شکل ۳-۱: A* on $N = 20, P = 0.2$

۳-۱-۳ نمونه ها

خروجی الگوریتم برای $N = 20$ و $P = 0.2$ (بخش ۳-۱-۳) ، $P = 0.3$ (بخش ۳-۱-۳) ، و $P = 0.12$ (بخش ۳-۱-۳) نشان داده شده است .

شکل ۳-۲: A* on $N = 20, P = 0.3$



شکل ۳-۳: A* on $N = 20, P = 0.12$

۳-۲ Greedy best-first search

برای این الگوریتم [۳] [۴] نیز از همان تابع هیوریستیکی که در بالا استفاده شد استفاده میکنیم و پیاده سازی این الگوریتم کاملا مشابه $A*$ است البته هزینه‌ی صرف شده تا این نقطه را در نظر نمی‌گیریم یعنی:

$$f(C) = h(C) \tag{۳-۴}$$

۳-۲-۱ نمونه ها

خروجی الگوریتم برای $N = 20$ و $P = 0.2$ (بخش ۳-۲-۱) ، $P = 0.3$ (بخش ۳-۲-۱) ، و $P = 0.12$ (بخش ۳-۲-۱) نشان داده شده است .

۳-۳ Hill-climbing

این الگوریتم [۵] [۶] بالا رفتن از یک تپه را شبیه‌سازی می‌کند.

۳-۳-۱ تابع تخمین ارزش

مقدار این تابع باید طوری باشد که با افزایش راس‌های پوشش یافته زیاد شود و با افزایش تعداد رئوس انتخاب‌شده کاهش یابد. اگر ضریب این دو مقدار برابر باشند یعنی تابع به فرم $-|C|-|G-C-nei(C)|$ باشد، الگوریتم لزومی در انتخاب یال‌های بدیهی نخواهد داشت. برای همین ضریب $|G - C - nei(C)|$ را -2 قرار دادیم و برای مثبت نگه داشتن کل عبارت را با $2|G|$ جمع کردیم. پس تابع به فرم

$$E(C) = 2|G| - 2|G - C - nei(C)| - |C| \tag{۳-۵}$$

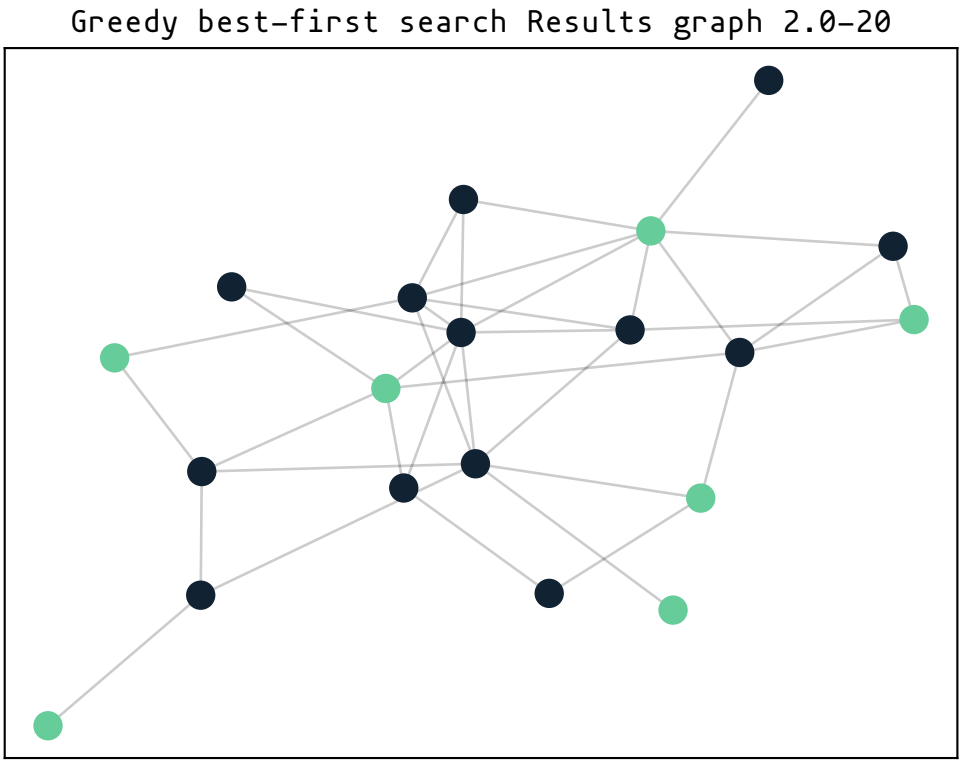
خواهد بود.

[3] https://en.wikipedia.org/wiki/Best-first_search

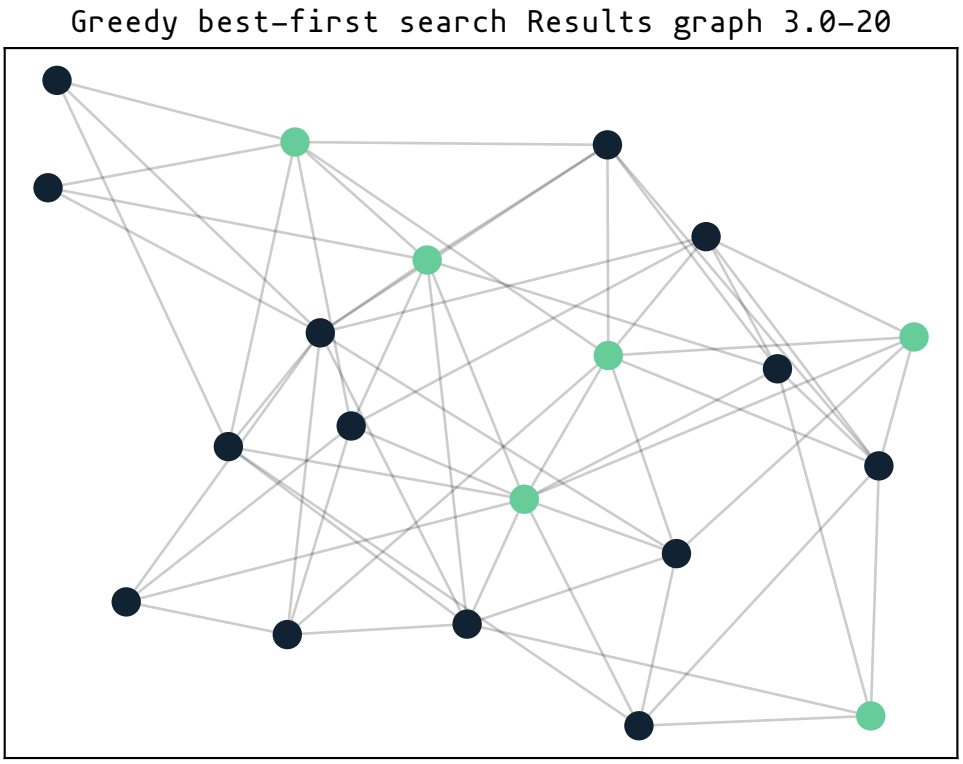
[4] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/blob/main/Report%2002/Problem%2001%20-%20Vertext%20Cover/solutions/GreedyBestFirst.py

[5] https://en.wikipedia.org/wiki/Hill_climbing

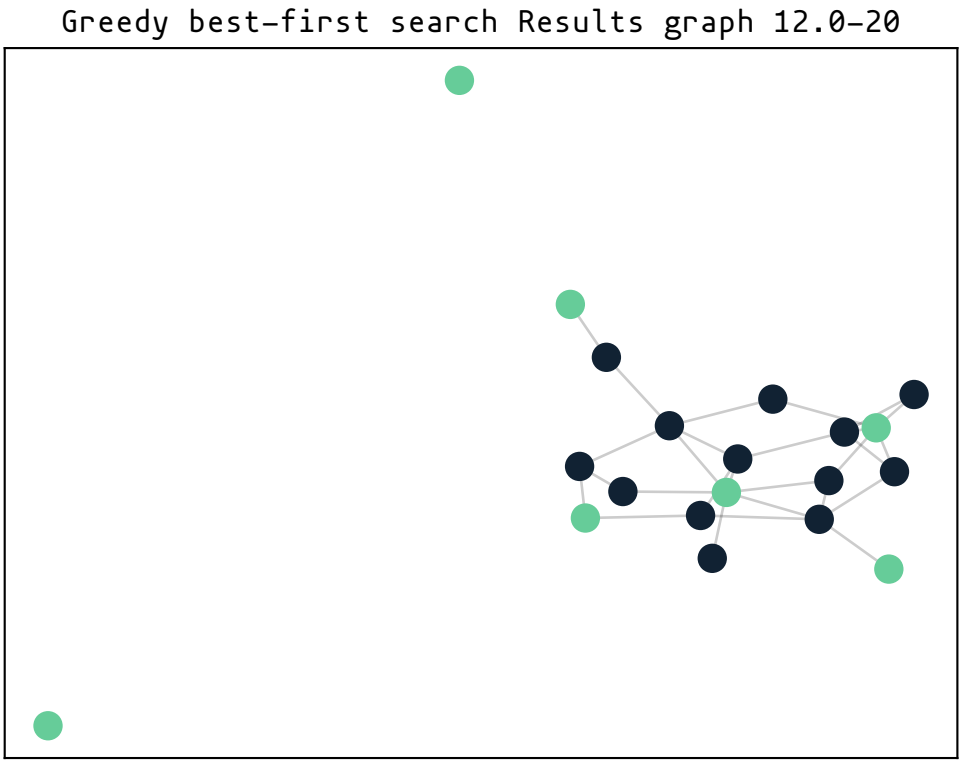
[6] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/blob/main/Report%2002/Problem%2001%20-%20Vertext%20Cover/solutions/HillClimbing.py



شکل ۳-۴: Greedy best-first on $N = 20, P = 0.2$



شکل ۳-۵: Greedy best-first on $N = 20, P = 0.3$



شکل ۳-۶: Greedy best-first on $N = 20, P = 0.12$

۳-۳-۲ همسایگی

دو حالت را همسایه میگوییم هرگاه یکی با حذف دقیقا یک عضو به دیگری تبدیل شود.

۳-۳-۳ شروع تصادفی

برای افزایش احتمال پیدا کردن جواب درست میتوانیم بجای شروع از مجموعه خالی، از مجموعه‌ای از اعضای تصادفی انتخاب شده استفاده کنیم بطوری که احتمال حضور هر یک از آن‌ها در مجموعه‌ی اولیه‌برابر متغییر `rand_start` باشد.

۳-۳-۴ نمونه ها

خروجی الگوریتم برای $N = 20$ و $P = 0.2$ (بخش ۳-۳-۴) ، $P = 0.3$ (بخش ۳-۳-۴) ، و $P = 0.12$ (بخش ۳-۳-۴) نشان داده شده است .

۳-۴ Annealing Search

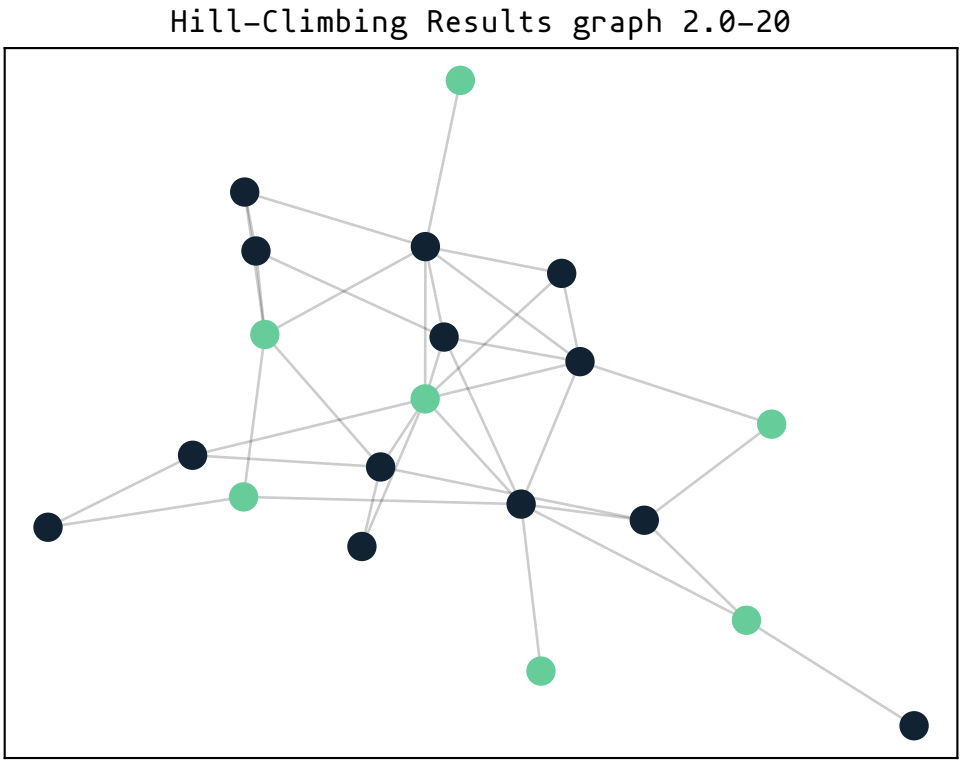
برای پیاده سازی این الگوریتم [۷] [۸] از تابع ارزش‌گذاری مانند بالا استفاده میکنیم. تغییرات دما به فرم $T_{next} = 0.98T_{now}$ با حداقل دمای 0.000001 و دمای اولیه ۱ ثبت شده‌اند.

۳-۴-۱ نمونه ها

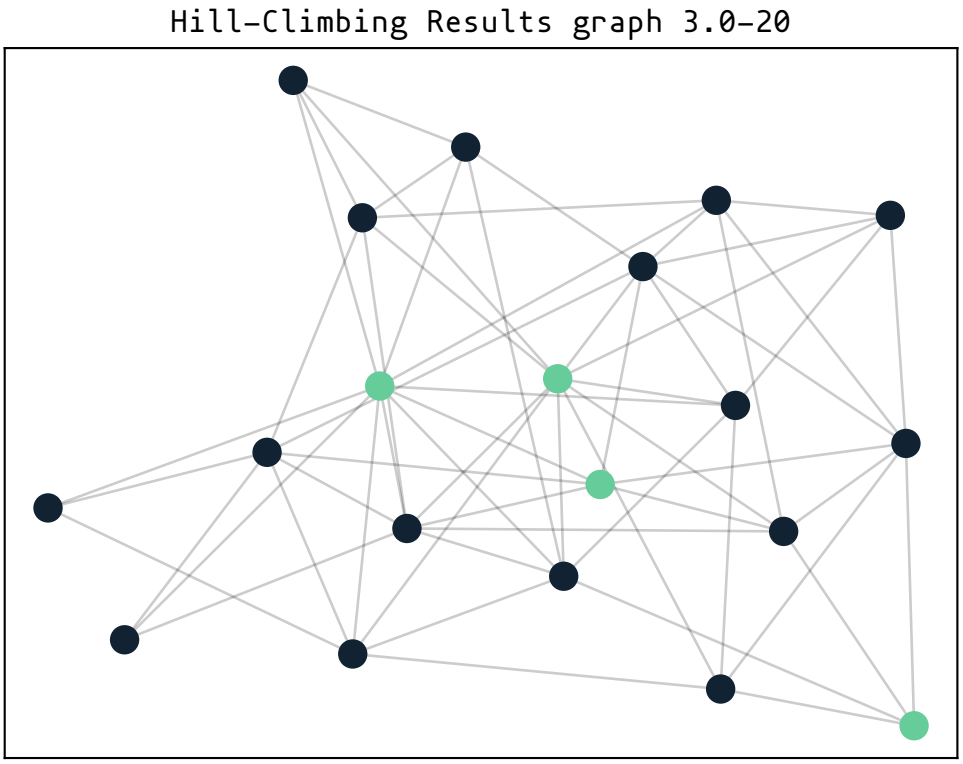
خروجی الگوریتم برای $N = 20$ و $P = 0.2$ (بخش ۳-۴-۱) ، $P = 0.3$ (بخش ۳-۴-۱) ، و $P = 0.12$ (بخش ۳-۴-۱) نشان داده شده است .

[7] https://en.wikipedia.org/wiki/Simulated_annealing

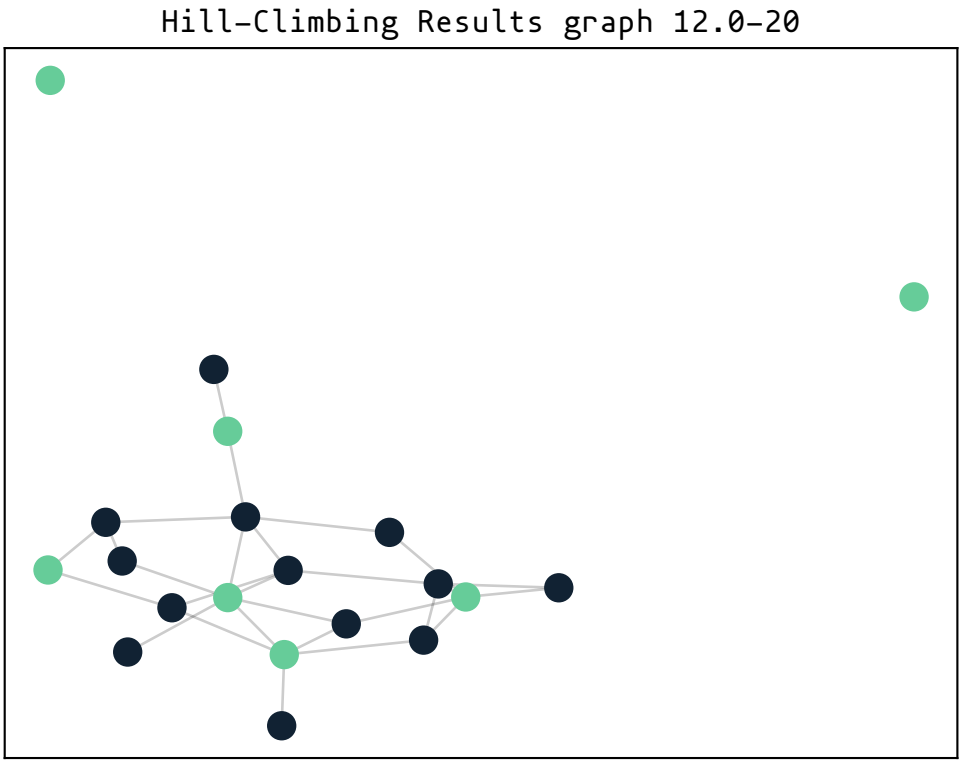
[8] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/blob/main/Report%2002/Problem%2001%20-%20Vertext%20Cover/solutions/AnnealingSearch.py



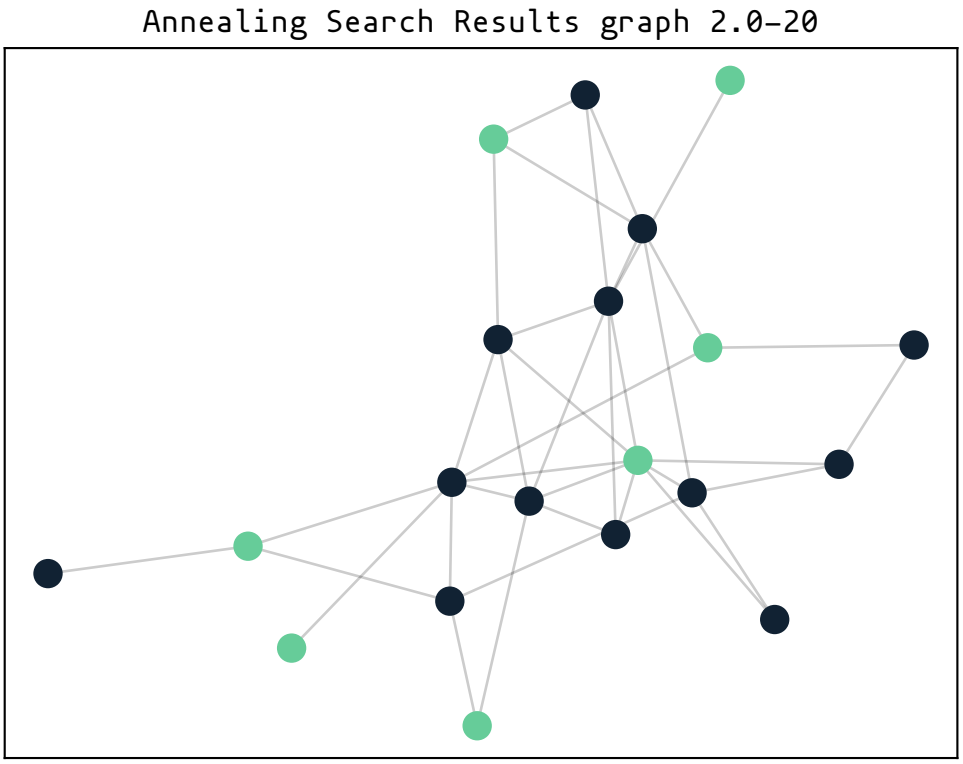
شکل ۳-۷: Hill-climbing on $N = 20, P = 0.2$



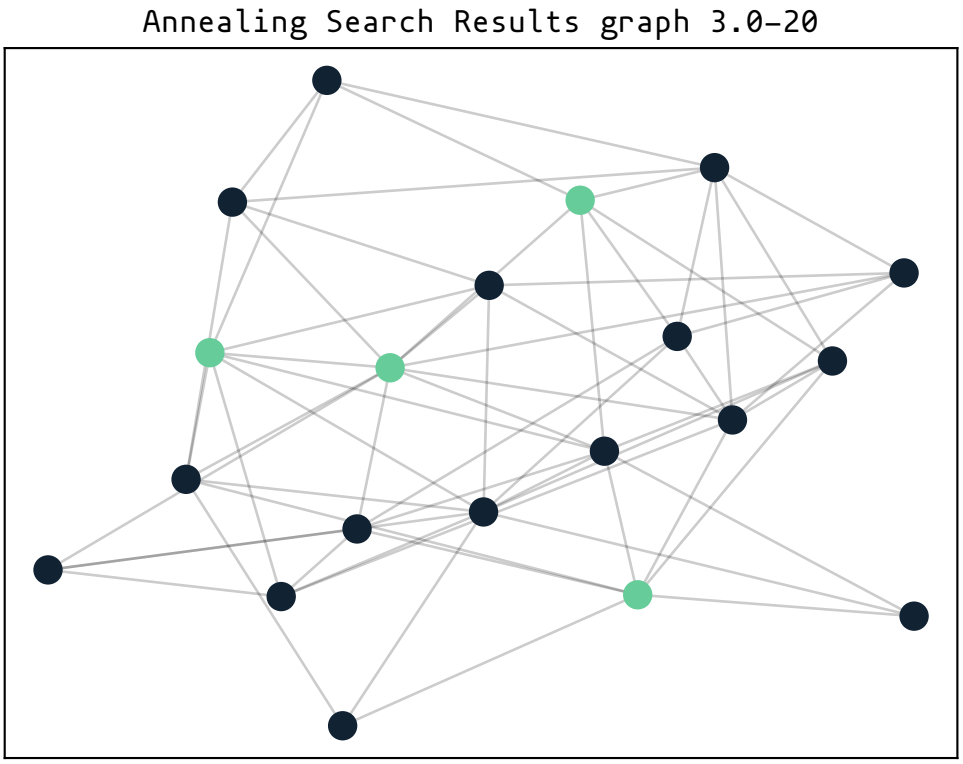
شکل ۳-۸: Hill-climbing on $N = 20, P = 0.3$



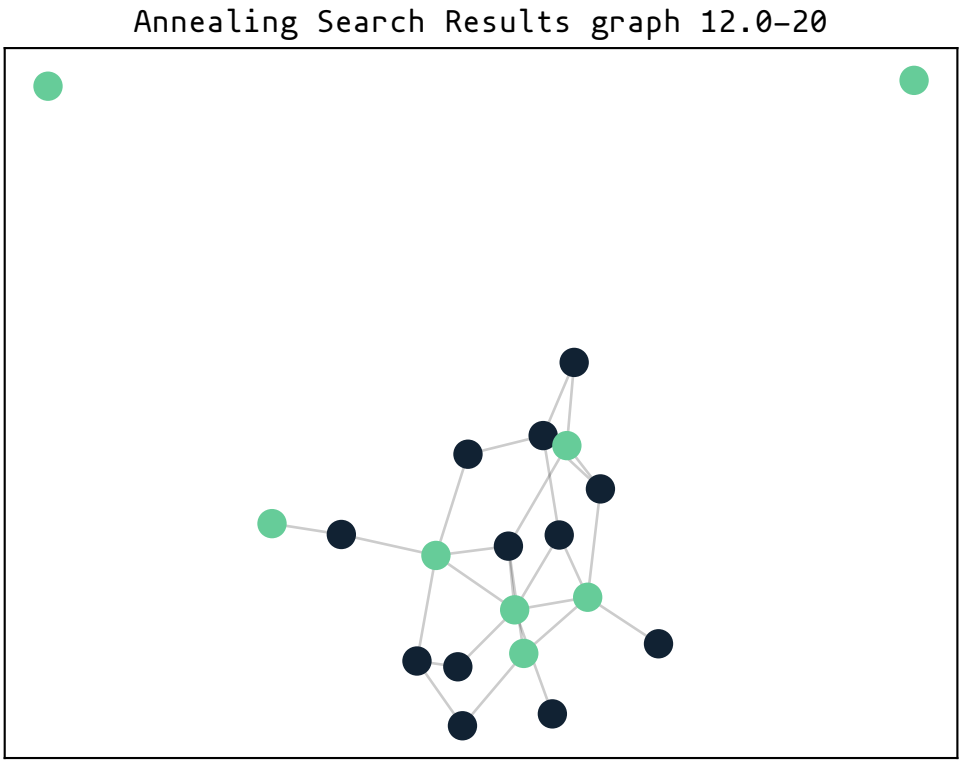
شکل ۳-۹: Hill-climbing on $N = 20, P = 0.12$



شکل ۳-۱۰: Annealing on $N = 20, P = 0.2$



شکل ۳-۱۱: Annealing on $N = 20, P = 0.3$



شکل ۳-۱۲: Annealing on $N = 20, P = 0.12$

فصل چهارم

مقایسه روش‌های متفاوت

۴-۱ تست‌ها و روش اندازه‌گیری

برای مقایسه‌ی روش‌های استفاده شده از دو تابع

```
def TestBy(start, end, diff, tries, nodes):
  pass
def TestByN(start, end, diff, tries, p):
  pass
```

استفاده شده است. تابع `TestByP` به ازای همه‌ی مقادیر $start \leq P < end$ با قدم‌هایی به اندازه‌ی $diff$،هر چهار الگوریتم را `tries` بار روی یک گراف جدید که با تابع `gen_graph_eq_prob_edges` به دست آمده‌اند اجرا کرده و جواب‌ها و تعداد تکرارها را مقایسه می‌کند. تابع `TestByN` نیز با تغییر N روند مشابهی را طی می‌کند. سپس هردوی این توابع، تعداد تکرارها، درصد موفقیت(پیدا کردن کمترین پاسخ صحیح) و تختلاف هریک با کمترین پاسخ صحیح را نمایش می‌دهند. [۱]

مقادیر ورودی این توابع در تست‌ها به شکل زیر می‌باشند:

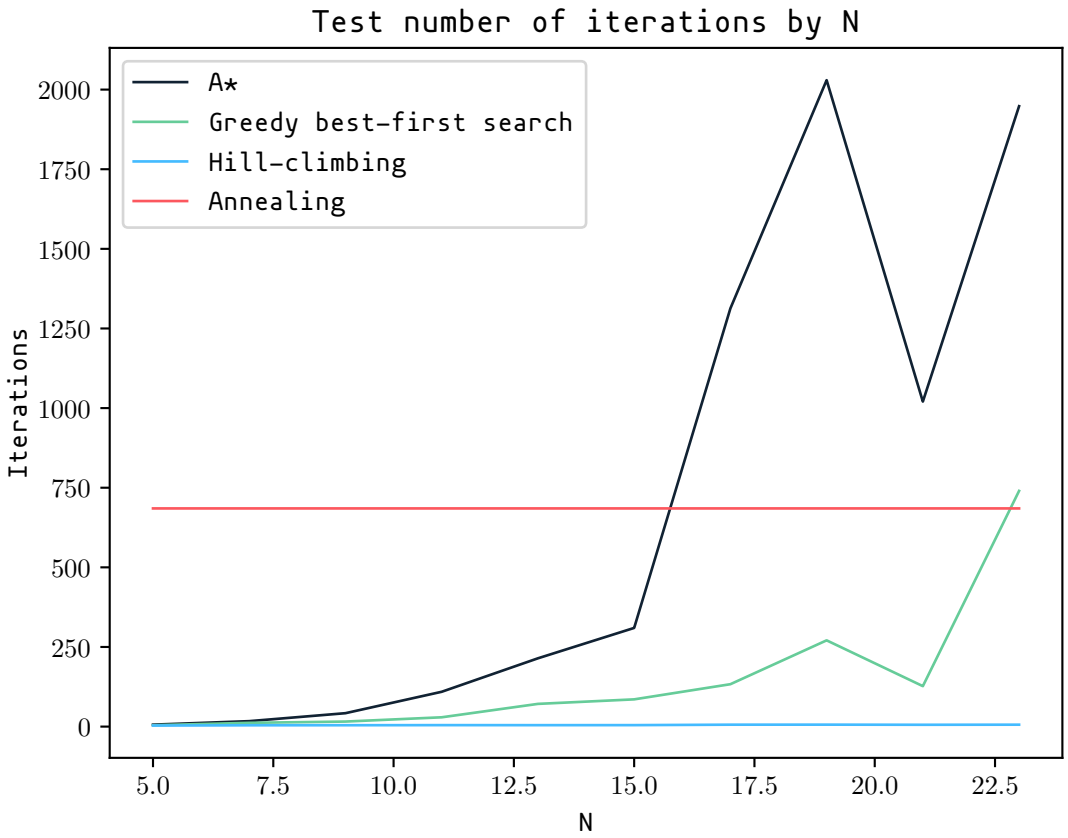
```
TestByP(start=0.12, end=.36, diff=0.04, tries=10, nodes=20)
TestByN(start=5, end=25, diff=2, tries=10, p=0.2)
```

۴-۲ مقایسه‌ی تعداد تکرارها

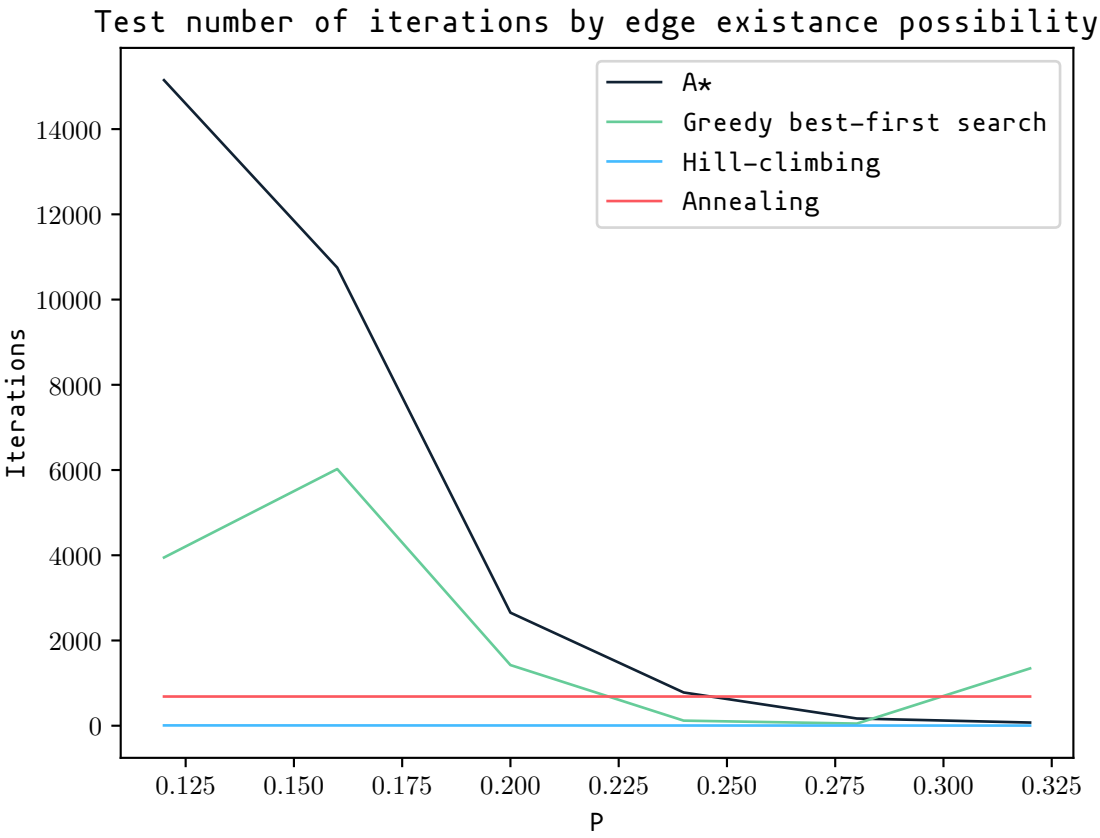
الگوریتم $A*$ و `Greedy best-frist search` به ترتیب بیشترین تعداد تکرارها با کاهش P را داشتند اما $Annealing$ و $Hill-climbing$ با افزایش N و P تغییر چندانی نداشتند. با نزدیک شدن به $P = 0.24$ تعداد تکرارهای $A*$ از $Annealing$ کمتر شد و با نزدیک شدن به $P = 0.3$ به تعداد تکرارها به تکرارهای الگوریتم $Hill-climbing$ رسید. (بخش ۴-۲)

در N های کوچک الگوریتم $Annealing$ به مراتب به تکرارهای بیشتری از بقیه نیاز دارد اما با افزایش N ، الگوریتم‌های دیگر به میزان تکرارهای بیشتری نیاز پیدا می‌کنند، الگوریتم $A*$ با افزایش نمایی در حدود $N = 16$ و `Greedy best-first search` در حدود $23 \leq N \leq 25$. (بخش ۴-۲)

[۱] خروجی در $output/TestBy[N, P]$ ذخیره می‌شود



شکل ۴-۱: تعداد تکرارهای برنامه با افزایش تعداد رئوس



شکل ۴-۲: تعداد تکرارهای برنامه با افزایش احتمال وجود هر یال

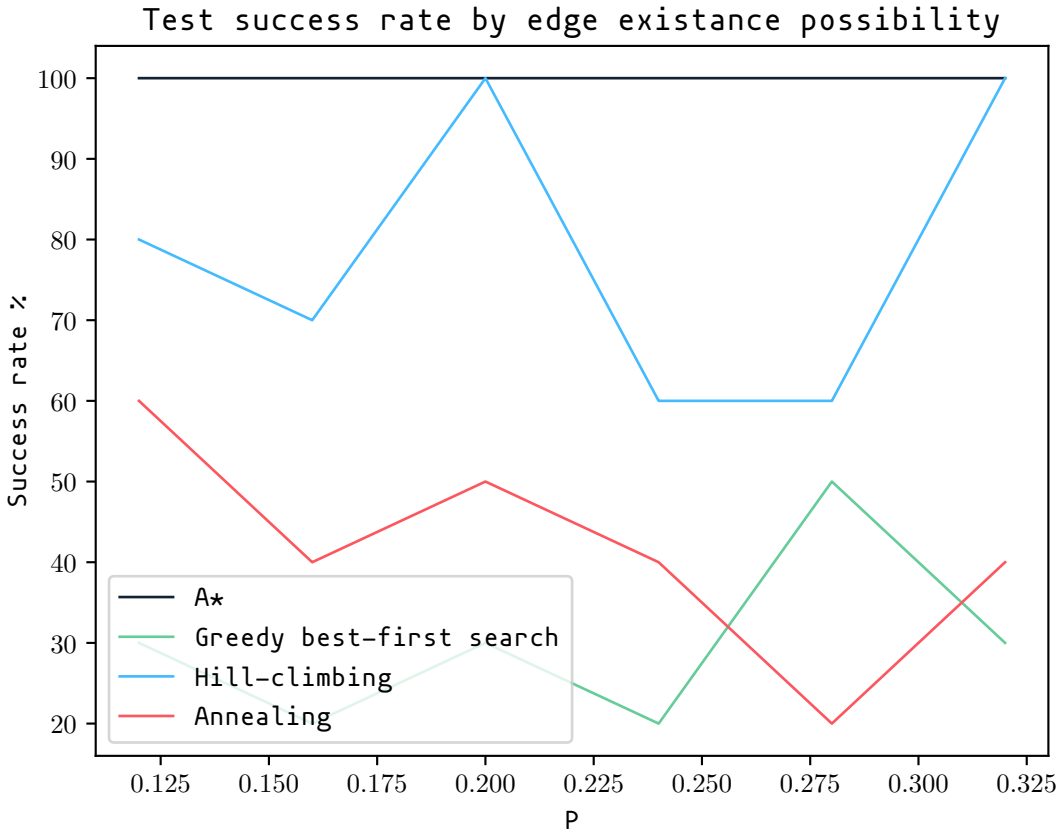
۴-۳ مقایسه صحت

همانطوری که در بخش ۳-۱-۲ بررسی شد، الگوریتم $A*$ همواره جواب بهینه را پیدا می‌کند.

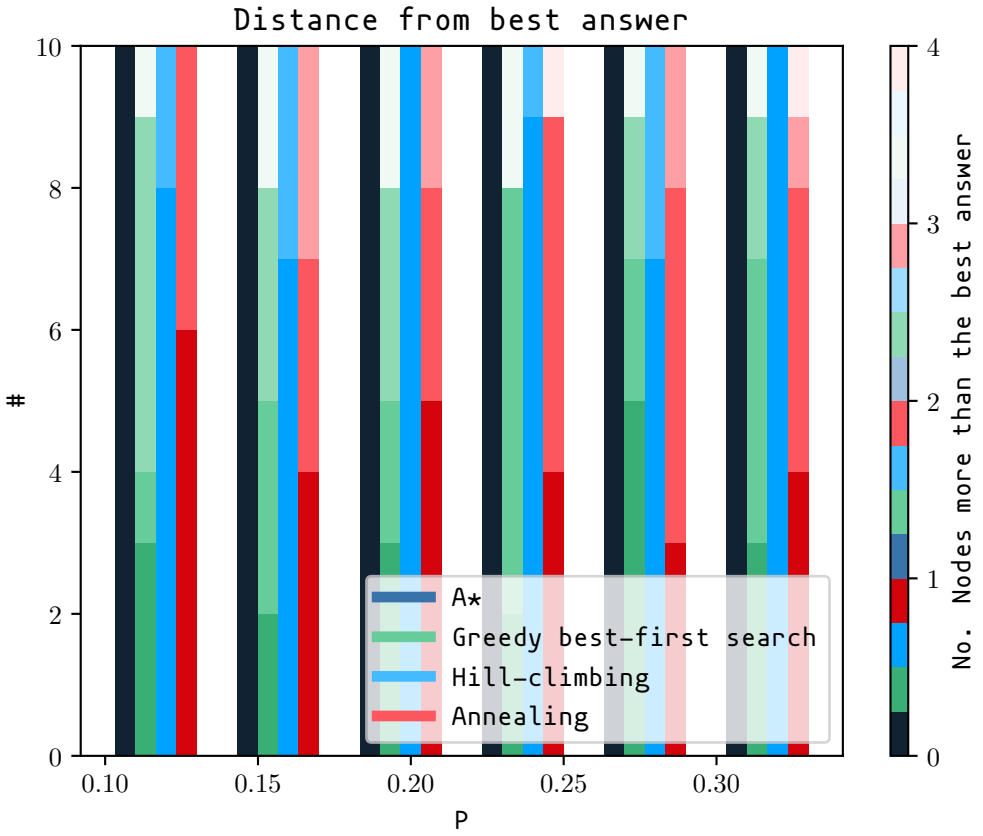
الگوریتم `Greedy best-first search` با تغییر P تغییر چندانی نمی‌کند و حدود ۳۰ تا ۴۰ درطد مواقع جواب بهینه را می‌دهد و بین ۷۰ تا ۸۰ درصد مواقع حاکثر سه راس بیشتر از $A*$ انتخاب می‌کند. از طرفی در $N \leq 10$ همواره جواب بهینه می‌دهد و با افزایش N افت شدیدی در دقت آن مشاهده می‌شود.

الگوریتم `Annealing` در P های کوچک درصد موفقیت بیشتر و تعداد بیشتری جواب با اختلاف حداکثر ۳ با بهترین جواب در مقایسه به `Greedy best-first search` دارد. با افزایش P اختلاف درصد موفقییت ناچیز می‌شود اما تعداد جواب ها با اختلاف حداکثر ۳ از جواب بهینه، اگرچه کم می‌شود اما هنوز وجود دارد. در N های کوچک `Greedy best-first search` عملکرد بهتری در هر دو زمینه نشان می‌دهد و از حدود $N = 11$ دقت و صحت $Annealing$ بیشتر می‌شود.

الگوریتم `Hill-climbing` نزدیک‌ترین جواب به جواب بهینه را می‌دهد و با تغییر N و P همواره دقت و صحت بیشتری از `Greedy best-first search` و `Annealing` دارد.

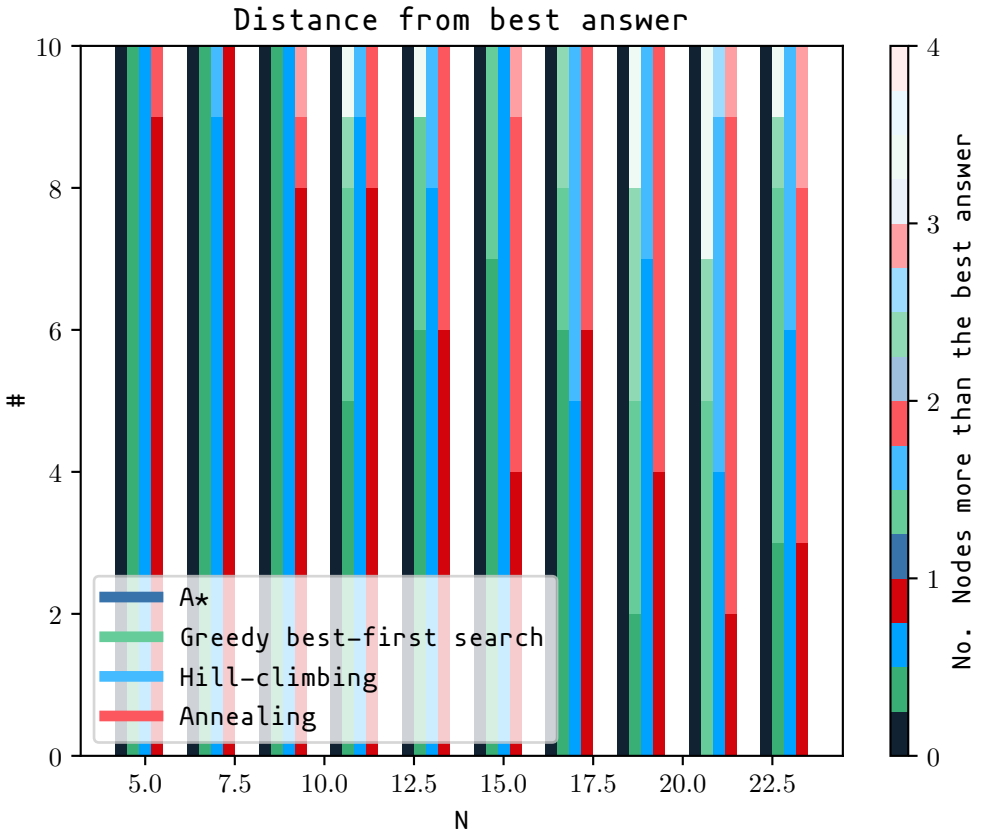


شکل ۴-۳: درصد یافتن جواب درست با افزایش احتمال وجود هر یال



شکل ۴-۴: اختلاف تعداد رئوس با بهترین جواب با تافزایش احتمال وجود هر یال

شکل ۴-۵: درصد یافتن جواب درست با افزایش تعداد رئوس



شکل ۴-۶: اختلاف تعداد رئوس با بهترین جواب با افزایش تعداد رئوس

منابع و مراجع

[1] Sasireka, A and Kishore, AH Nandhu. Applications of dominating set of a graph in computer networks. *Int. J. Eng. Sci. Res. Technol*, 3(1):170–173, 2014.

پیوست