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

پایان‌نامه کارشناسی‌ارشد
گرایش ریاضی

هوش مصنوعی - گزارش ۰۲ - پیاده سازی الگوریتم های جستجوی محلی روی مسائل مختلف

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

در این گزارش عملکرد الگوریتم‌های جستجوی محلی متفاوت (Annealing Search, Hill-Climbing search, Local beam search, Genetic algorithm) را روی دو مسئله‌ی فروشنده‌ی دوره‌گرد(TSP [۱]) و مسئله صدق‌پذیری دودویی (3-sat [۲]) بررسی خواهیم کرد. هردوی این مسائل NP Complete میباشند و این الگوریتم ها صرفا تخمینی از جواب خواهند بود.

۱-۲ کدها و خروجی‌های برنامه

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

[1] Traveling Salesman problem

[2] Boolean Satisfiability Problem

فصل دوم

توابع ابتدایی

۲-۱ تولید گراف تصادفی

برای تولید گراف تصادفی از تابع زیر استفاده شده است. [1] در این تابع گرافی کامل با nodes راس تولید شده که هر یال آن عددی بین min_weight و max_weight می‌باشد.

```
def gen_weighted_graph(nodes=30, wei_min=1, wei_max=100):

   graph =[[0 for j in range(nodes)] for i in range(nodes)]

   for i in range(nodes):
      for j in range(i):
         wei =math.floor(random.random() *(wei_max -wei_min)) +wei_min
         graph[i][j] =wei
         graph[j][i] =wei
   return graph
```

۲-۲ تولید 3-SAT تصادفی

تابع gen_sat از بین تمام سه‌تای های $x, y, z \in \{x, \neg x | x \in \{x_1, x_2, ...x_n\}\}$ هریک را به احتمال p انتخاب می‌کند.

برای چک کردن همه‌ی این سه‌تای‌ها یک متغیر $xi \in -1, 1$ نگه میداریم که در صورت صفر بودن آن فرض میکنیم که در آن ستایی منظور عکس بوده است.

```
def gen_sat(variables=20, p=0.2):
   sat =[]

   for i in range(variables):
      for xi in range(-1, 2, 2):
         for j in range(i +1):
            for xj in range(-1, 2, 2):
```

[1] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/blob/main/Report%2003/code/generators/__init__.py

```
                for k in range(j +1):
                    for xk in range(-1, 2, 2):
                        if random.random() <p:
                            clause =[xk *(k +1), xj *(j +1), xi *(i +1)]
                            sat.append(tuple(clause))

    return sat
```

۲-۳ تصویر سازی

برای نمایش دادن نمودارهای برنامه از matplotlib [۲] و seaborn [۳] و برای گراف‌ها از networkx [۴] استفاده شده. [۵]

۲-۴ نمایش TSP

برای نمایش TSP از تابع زیر استفاده شده. [۶]

```
def visualize_tsp(graph, path=[], name="",
        output=None):
    plt.figure()
    plt.title(name)
    G =nx.Graph()
    color_map =[]
    for i in range(len(graph)):
        if i in path:
```

[2] https://matplotlib.org/
[3] https://seaborn.pydata.org/
[4] https://networkx.org/documentation/stable/
[5] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/tree/main/Report%2003/code/visualizers
[6] https://github.com/atrin-hojjat/Uni-AI-Course-Reports/tree/main/Report%2003/code/visualizers/GraphVisualizers.py

```
        color_map.append("#66CC99")
    else:
        color_map.append("#112233")
G.add_nodes_from([i for i in range(len(graph))])
path_edges =[sorted((path[i -1], path[i])) for i in range(len(path))]
print(path)
print(graph)
print(path_edges)
G.add_edges_from(
        [
            (i, j, {'weight': graph[i][j],
                'color': '#112233' if (i, j) not in path_edges else
                                                    '#FC575E'})
            for i in range(len(graph)) for j in range(len(graph[i])) if (i <
                                                    j)
        ]
)
G.add_edge(2, 5, weight=3)
for i in range(len(path)):
    G.edges[path[i -1], path[i]]['color'] ='red'
# G.add_edges_from(
#         [
#             (path[i - 1], path[i], {'weight': graph[path[i - 1]][path[i]]})
#             for i in range(len(path))

#         ],
#         color="#FC575E"
# )
pos =nx.shell_layout(G)
edge_colors =nx.get_edge_attributes(G,'color').values()
nx.draw_networkx_edges(G, pos, alpha=0.4, edge_color=edge_colors)
nx.draw_networkx_nodes(G, pos, node_color=color_map, node_size=100)
```

```
labels =nx.get_edge_attributes(G,'weight')
nx.draw_networkx_edge_labels(G,pos,edge_labels=labels)
if output:
    if not os.path.exists(os.path.dirname(os.path.join("./output",
                                          output, f"{name}.jpg"))):
        try: os.makedirs(os.path.dirname(os.path.join("./output", output,
                                            f"{name}.jpg")))
        except OSError as exc: # Guard against race condition
            if exc.errno !=errno.EEXIST:
                raise
    plt.savefig(os.path.join("./output", output, f"{name}.jpg"))
    matplotlib.rcParams.update({
        "pgf.texsystem": "pdflatex",
        'font.family': 'mononoki Nerd Font Mono',
        'text.usetex': True,
        'pgf.rcfonts': False,
    })
    plt.savefig(os.path.join("./output", output, f"{name}.pgf"))
plt.show()
```

فصل سوم

پیاده‌سازی TSP

برای پیاده سازی این الگوریتم در فایل LoadTSP.py دو تابع load_samples و run_tests نوشته شده‌است. تابع load_samples یک گراف تصادفی درست کرده و خروجی هر چهار الگوریتم را روی آن نمایش می‌دهد و تابع run_tests خروجی الگوریتم‌ها را با افزایش N بررسی می‌کند. الگوریتم‌های پیاده شده در اینجا قابل مشاهده هستند.

۳-۱ جواب بهینه

با توجه به سرعت پایین زبان python برای پیداکردن جواب بهینه از C استفاده‌شد. این الگوریتم در زمان $O(n^2.2^n)$ و حافظه‌ی $O(n.2^n)$ اجرا می‌شود. [۱] [۲] برای اجرای این کد دستور زیر را اجرا کنید.

```
./build tsp
```

۳-۲ تخمین ارزش

در پیاده‌سازی هر چهار الگوریتم، برای تخمین، نیاز به ماکسیمایز کردن یک تابع داریم. از آنجایی که هدف TSP حداقل سازی طول مسیر است، تابع مورد نظرمان را تفاضل مجموع یال‌ها و طول مسیر قرار دادیم.

۳-۳ همسایگی

دو دور را همسایه می‌گوییم اگر یکی با جابجا کردن دو عضو متوالی به دیگری تبدیل شود.

۳-۴ Local Beam Search

در این تابع تعداد نمونه‌ها 10 و حداکثر تکرارها برای تست‌ها 1000 و برای نمونه‌ها 10000 قرار داده شده.

[۱] شرح الگوریتم
[۲] کد

۳-۵ Annealing Search

در این تابع تغییرات دما به فرم $T_{next} = 0.98T_{now}$ با حداقل دمای 0.000001 و دمای اولیه ۱ ثبت شده‌اند. در تست‌ها حداقل دما 0.001 قرار داده شده.

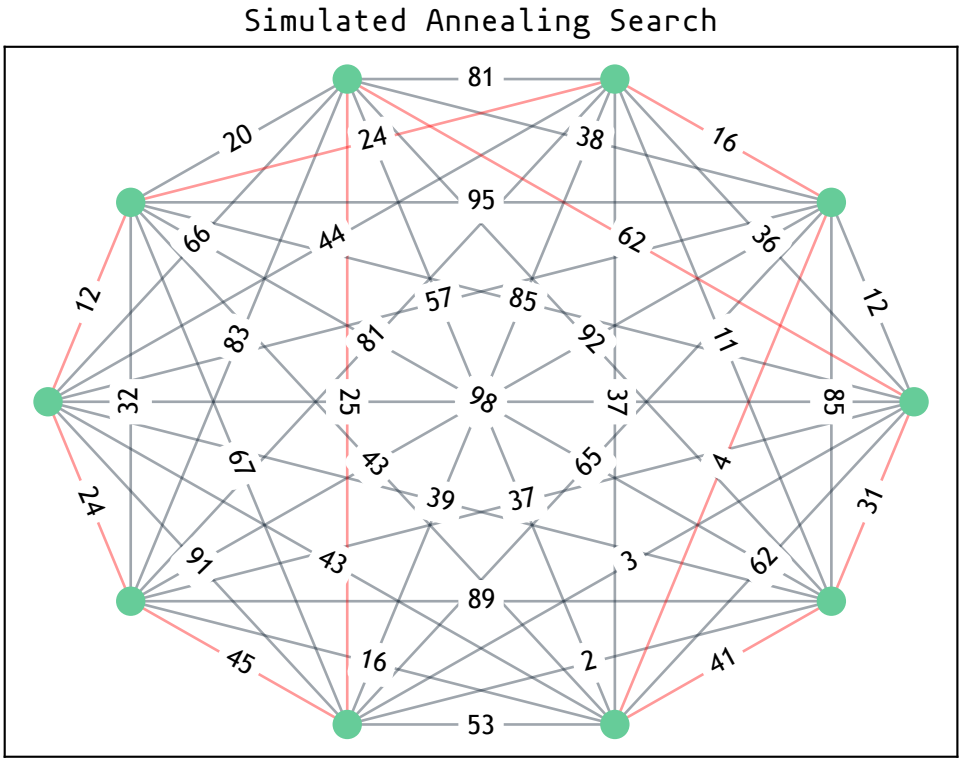
۳-۶ الگوریتم ژنتیکی

در این الگوریتم سایز جمعیت ده و احتمال Mutation برابر 0.05 قرار داده‌شده. حداکثر تکرارها برای تست‌ها 1000 و برای نمونه‌ها 10000 می‌باشد.

۳-۶-۱ تولید مثل

برای تولید بچه‌ی دو مسیر، یک بازه‌ی متوالی از پدر را انتخاب کرده و رئوسی که نیامده‌اند را به‌ترتیب حضورشان در مادر قرار می‌دهیم.

```
def crossover(parent0, parent1):
    start, end =random.randrange(len(graph)), random.randrange(len(graph))
    if start >end:
        start, end =end, start
    child =[None] *len(graph)
    for i in range(start, end +1):
        child[i] =parent0[i]
    ptr =0
    for i in range(len(graph)):
        if child[i] !=None:
            continue
        while parent1[ptr] in child:
            ptr +=1
        child[i] =parent1[ptr]
    return child
```

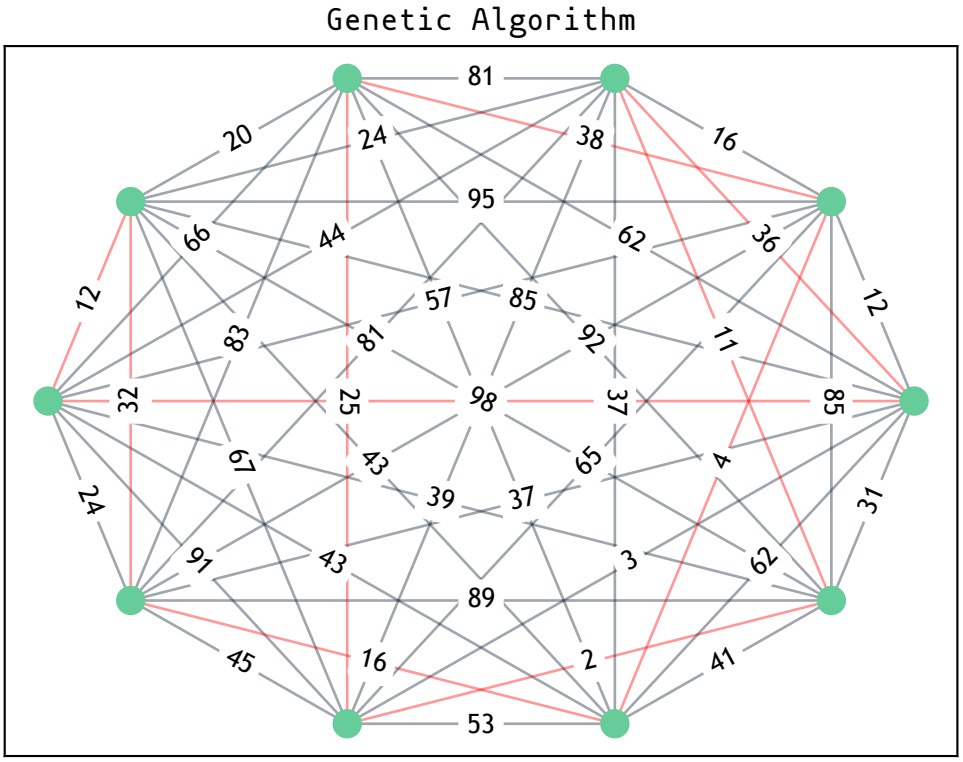


شکل ۳-۱: Simulated Annealing on $N = 20$

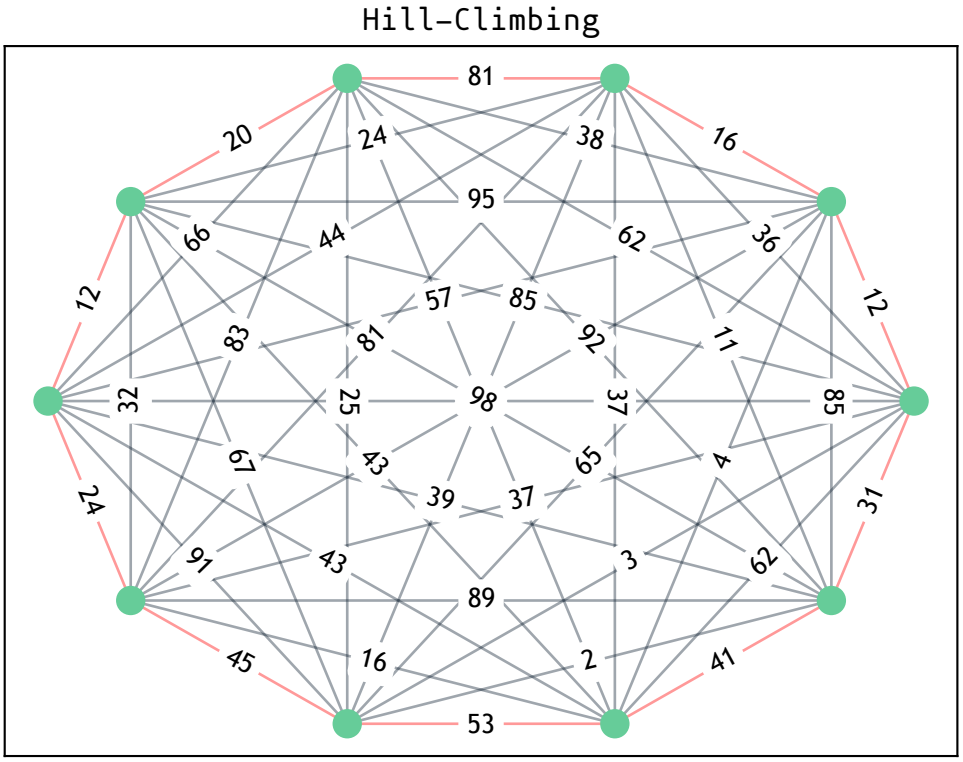
۳-۶-۲ Mutation

Mutation را حاصل جابجای دو عضو متوالی تعریف می‌کنیم.

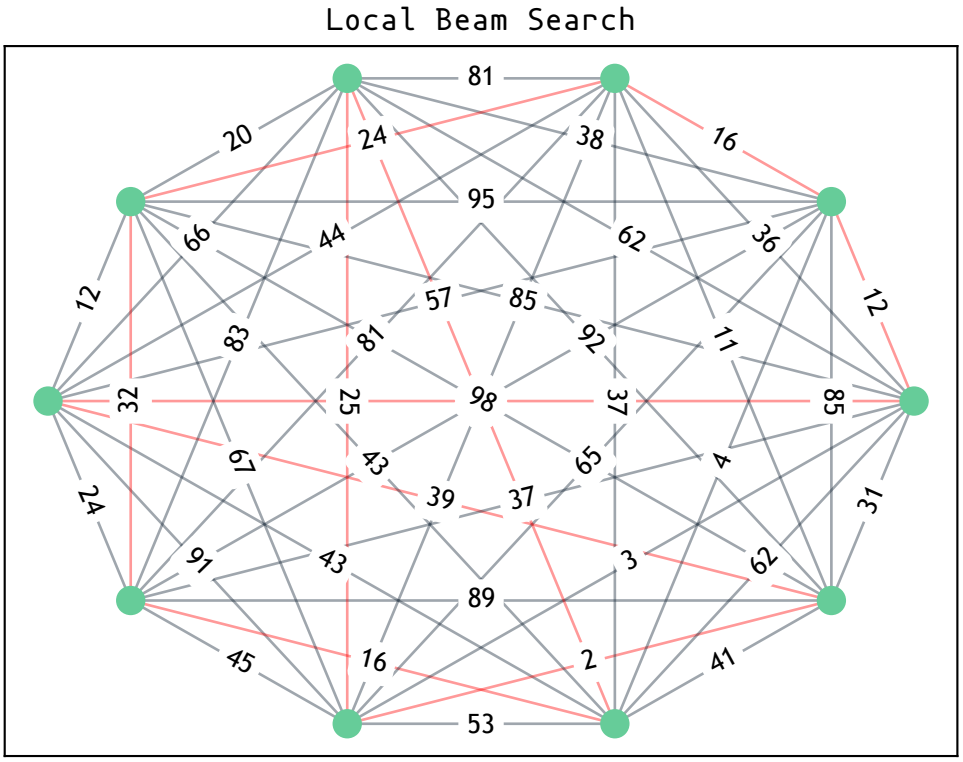
```
def mutate(cell):
    ind =random.randrange(len(graph))
    cell[ind], cell[ind -1] =cell[ind -1], cell[ind]
    return cell
```



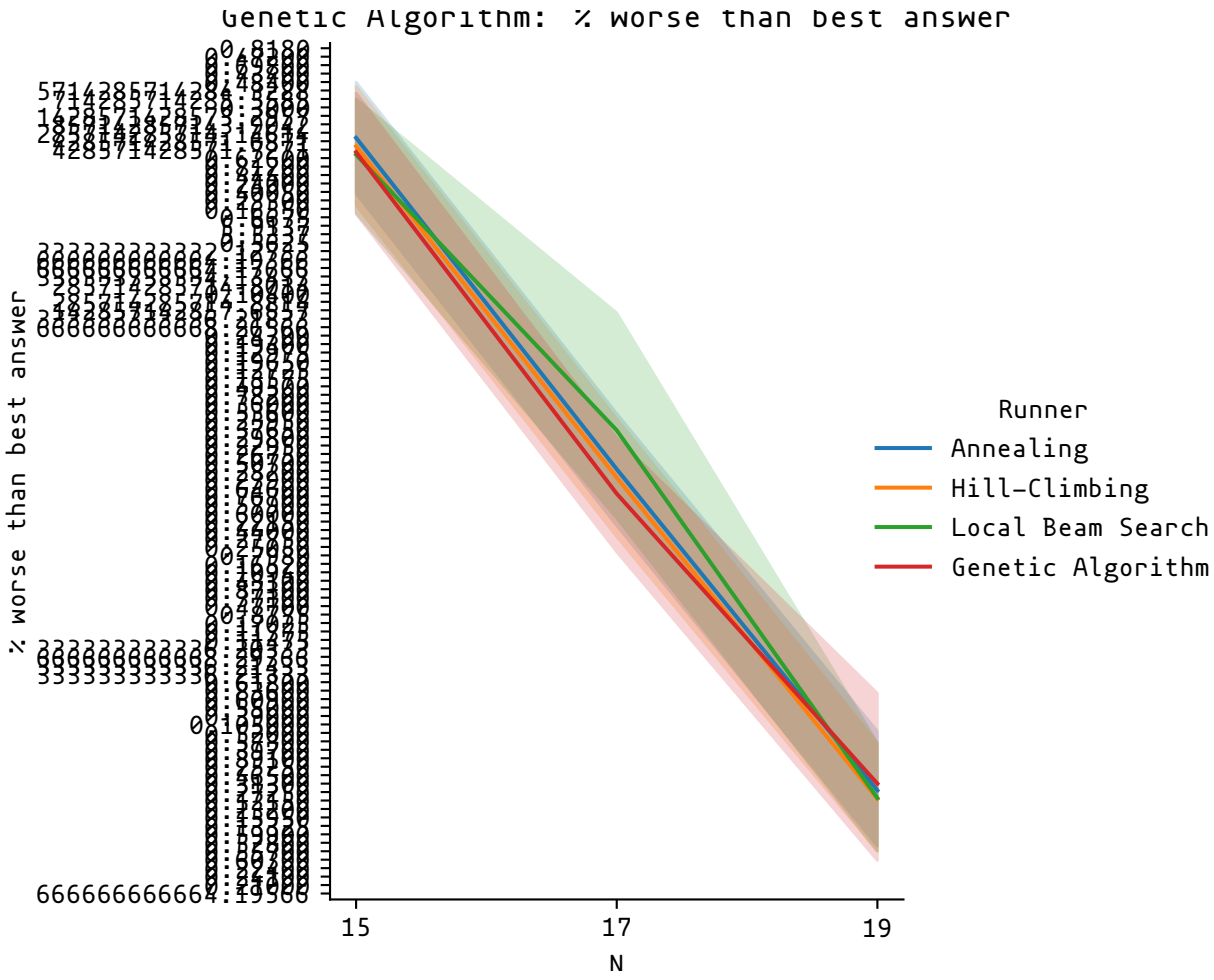
شکل ۳-۲: Genetic Algorithm on $N = 20$



شکل ۳-۳: Hill-Climbing Algorithm on $N = 20$



شکل ۳-۴: Local Beam Search on $N = 20$



شکل ۳-۵: % Worse than best answer

۳-۷ خروجی‌های نمونه

۳-۸ مقایسه‌ی نتایج

به‌طور میانگین Simulated Annealing Search بهترین نتیجه را داشت ولی در کل تفاوت چندانی بین الگوریتم‌ها نبود. به لحاظ زمان همه تا حداکثر تعداد تکرار ها پیش می‌رفتند.

فصل چهارم

پیاده‌سازی 3-SAT

در اینجا از آنجایی که هر مسئله‌ی SAT می‌تواند به یک 3-SAT تبدیل‌شود، [1] صرفا مسئله‌ی 3-SAT بررسی می‌شود. کد الگوریتم‌های این بخش در اینجا قابل دسترسی است.

در فایل `LoadSAT.py` تابع `load_samples` نوشته شده که خروجی‌های الگوریتم‌ها را برای یک مسئله‌ی SAT تصادفی محاسبه می‌کند.

۴-۱ تخمین ارزش

برای تخمین ارزش یک جواب، در هر چهار الگوریتم، از تعداد جملات صحیح در SAT استفاده شده است.

۴-۲ همسایگی

دو موقعییت همسایه‌اند، اگر تنها یک متغییر در آنها مقدار متفاوتی داشته باشد.

۴-۳ Local Beam Search

در این تابع تعداد نمونه‌ها 10 و حداکثر تکرارها برای تست‌ها 1000 و برای نمونه‌ها 10000 قرار داده شده.

۴-۴ Annealing Search

در این تابع تغییرات دما به فرم $T_{next} = 0.98T_{now}$ با حداقل دمای 0.000001 و دمای اولیه ۱ ثبت شده‌اند. در تست‌ها حداقل دما 0.001 قرار داده شده.

۴-۵ الگوریتم ژنتیکی

در این الگوریتم سایز جمعیت ده و احتمال `Mutation` برابر 0.05 قرار داده‌شده. حداکثر تکرارها برای تست‌ها 1000 و برای نمونه‌ها 10000 می‌باشد.

[1] https://en.wikipedia.org/wiki/Boolean_satisfiability_problem#3-satisfiability

۴-۵-۱ تولید مثل

در تولید مثل، از یک prefix مادر و suffix معادل آن در پدر استفاده می‌کنیم.

```
def crossover(parent0, parent1):
    start, end =random.randrange(n), random.randrange(n)
    if start >end:
        start, end =end, start
    child =[None] *n
    for i in range(start, end +1):
        child[i] =parent0[i]
    ptr =0
    for i in range(start):
        child[i] =parent1[ptr]
    for i in range(end +1, n):
        child[i] =parent1[ptr]
    return child
```

۴-۵-۲ Mutation

Mutation را حاصل عوض کردن یک متغییر تعریف می‌کنیم.

```
def mutate(cell):
    ncell =list(cell)
    i =random.randrange(n)
    ncell[i] =not ncell[i]
    return ncell
```

۴-۶ مقایسه‌ی نتایج

در اینجا نیز تفاوت چندانی بین روش‌های مختلف چه از نظر زمانی و چی از نظر درستی مشاهده نمیشود.

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