امیررضا محسنی 400243070

گزارش تمرین کامپیوتری دوم:

کلیت صورت پروژه:

در این تمرین می‌خواستیم مسئله‌ی 3SAT که مسئله‌ای NP می‌باشد به دو روش الگوریتم ژنتیک و الگوریتم simulated annealing حل کنیم. این الگوریتم‌ها جواب‌هایی تولید می‌کنند که بر اساس پارامترهای انتخابی ما، قابل کنترل هستند و جواب‌های نسبتاً بهینه را به ما می‌دهند.

کلیت روش عملکرد دو الگوریتم: ژنتیک

این دو الگوریتم هرکدام روش کار خود را دارند که به طور زیر شرح داده شده اند:

در الگوریتم ژنتیک در ابتدا یک سری جواب احتمالی ایجاد می‌کنیم که در حل مسئله آنها را pure random انتخاب کردیم سپس شروع به ساخت generation بعدی کردیم

در ساخت generation بعدی عضوی که بیشترین fitness را دارد (بیشترین تعداد پارتز را true می‌کند) را به همراه عضوی که کمترین fitness را دارد برمی‌داریم و سپس به اندازه‌ی GEN_SIZE-2 دفعه فرزند می‌سازیم به این صورت که دو والد به صورت رندوم و زنبار براساس تابع fitness آنها انتخاب می‌کنیم در نهایت نصف اول child را از والد اول و نصف دیگر را از والد دوم انتخاب می‌کنیم

(روش‌های دیگر مثل انتخاب random breakpoint را نیز تست کردیم ولی این روش بازدهی بهتری داشت)

سپس به احتمال p یک جایگاه رندوم در child را not می‌کنیم و به عبارتی به احتمال p یک mutation رخ می‌دهد که آنرا یکبار p=0.1 و یکبار p=0.5 در نظر گرفتیم که هرکدام جواب‌های بهینه‌ای می‌ساختند ولی p=0.5 میزان همگرایی را بشدت تحت تاثیر قرار می‌دهد و گاهی اوقات جوابی ضعیف تولید می‌کند

در این سوال GEN SIZE را 200 در نظر گرفتم وقتی این عدد 100 بود در ابتدا خیلی تنوع زیاد نیست و جواب به سرعت تغییر می‌کند و به ازای 500 نیز سرعت بازدهی پایین می‌آمد

تعداد Gen ها را نیز 1000 در نظر گرفتم که باعث می‌شد مطمئن از ثابت شدن جواب و همچنین سریع بودن به اندازه‌ی کافی solution بیشویم

$$\int_0^1 p(x)\,dx = 1$$

شرح simulated annealing :

در این روش در ابتدا از جای بالا شروع می کنیم $T = 100$ و به بعد $T = 0.01$ می رویم و به این صورت عمل می کنیم که یک حرکت در صورتی که جواب ما را بهتر می کرد (تعداد پرانتز های بیشتری را 1 می کرد) آنرا انجام می دهیم و در غیر این صورت به یک احتمال آنرا انجام می دهیم که با کم شدن T، کم میشود و براساس تأثیرش به صورت نمایی احتمال آن کم میشود.

روش انتخاب یک همسایه از حالت فعلی به این صورت است که یک متغیر رندوم از بین متغیر هایمان برمیداریم و مقدار assign شده به آن را Not میکنیم

برای انتخاب یک حالت که جواب را بهتر می کند از احتمال $e^{\frac{\Delta E}{T}}$ استفاده میکنیم که هرگاه شروط مورد استفاده ما را دارد ←

probability ⟶ $e^{\frac{-(new\_cost - old\_cost)}{T}}$ ⟶ هرچه new - old بزرگتر شود کم احتمال تر میشود

در پایان هر مرحله T را با ضریب α کوچک میکنیم که برای این سوال α را 0.05 در نظر گرفتیم

Temp ← Temp * α

در نهایت انتظار داریم که دو الگوریتم روی تست کیس های داده شده به شکل قابل قبولی بازده داشته باشند که برای نتیجه های آنها با plot با کمک کتابخانه matplotlib ساخته ایم. در صفحه ی بعدی این نمودارها حضور دارند و مقایسه شده اند.

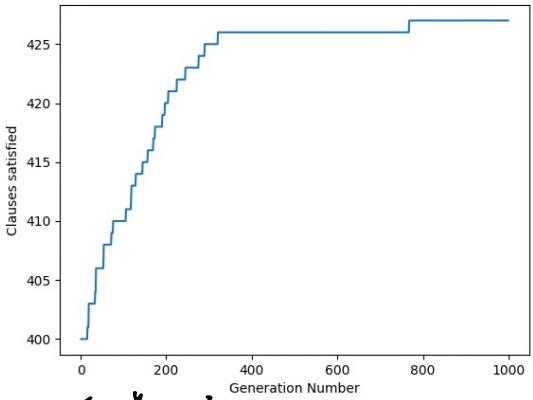
```python
def acceptance_probability(self, old_cost, new_cost, temperature):
    if new_cost < old_cost:
        return 1.0
    else:
        return math.exp(-(new_cost - old_cost) / temperature)
```

```python
while temp > self.tempmin and counter < self.maxcalls:
    i = 1
    success = 0
    while i <= self.maxpert:
        sol = self.sat.perturbation(sol_out)
        sol_fo = self.sat.eval(sol)
        delta = sol_out_fo - sol_fo
        ap = self.acceptance_probability(sol_out_fo, sol_fo, temp)
        if ap > uniform(0, 1):
            sol_out = sol
            sol_out_fo = sol_fo
            success += 1
        i += 1
    temp = temp*self.alpha
    self.temp_list.append(temp)
    self.cost_list.append(self.sat.num_clauses - sol_out_fo)
    counter += 1
    print(sol_out_fo)
```

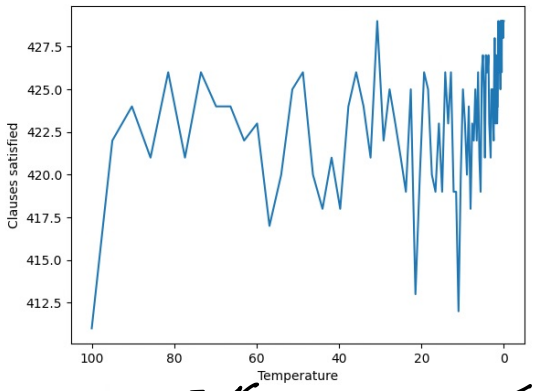
```python
mutation_probability = 0.1
new_population = []
sorted_population = []
probabilities = []
for n in cur_population:
    f = fitness(n)
    probabilities.append(f / maxFitness)
    sorted_population.append([f, n])

sorted_population.sort(reverse=True)

# Elitism
new_population.append(sorted_population[0][1])  # the best gen
new_population.append(sorted_population[-1][1])  # the worst gen

for i in range(len(population) - 2):

    chromosome_1 = random_pick(population, probabilities)
    chromosome_2 = random_pick(population, probabilities)

    # Creating two new chromosomes from 2 chromosomes
    child = crossover(chromosome_1, chromosome_2)

    # Mutation
    if random.random() < mutation_probability:
        child = mutate(child)

    new_population.append(child)
    if fitness(child) == maxFitness:
        break
return new_population
```

```python
def crossover(x, y):
    n = len(x)
    child = [0] * n
#    index_break = random.randint(0, n)
    index_break = n // 2
    for i in range(n):
        child = x[:index_break] + y[index_break:]
    return child
```

الگوریتم ژنتیک



الگوریتم SA



همانطور که مشخص است هردوالگوریتم به جواب‌های خوبی رسیده‌اند در زمان محدود برای یک مسئله‌ی کلان ولی در SA به جواب تقریباً کامل (429) رسیده‌ایم درحالی که در genetic پس از 1000 generation درهمان زمان به 427 رسیده‌ایم که 2 تا از جواب اصلی فاصله دارد

در نتیجه می‌توان گفت در این مسئله و testcase ها SA عملکرد بهتری داشته‌است و memory و time کمتری برای رسیدن به این جواب نیاز داشته‌است

از طرفی دیگر همانطور که در نمودار قابل رؤیت است در genetic در هر مرحله‌ی جدید، جواب بهتر و حداقل (بخاطر انتخاب بهترین جواب در مرحله‌ی بعدی) و با توجه به تزریق احتمال محول سعی بر این است که از یک حالت به حالت‌های بهتر برویم که صعودی بودن نمودار را نشان می‌دهد ولی در SA ممکن است به یک جواب بسیار رفتم و نامطلوب برویم و به مرور زمان سعی کنیم به یک جواب fixed برسیم بازه $T(20\to0)$ که درنتیجه باعث می‌شود نوسان پرازپیچ‌وتاب شدید در ابتدای کار و پیچ‌وتاب نسبتاً ملایم در آخرهای آن باشیم.

روند بهبود جواب در ژنتیک

```
(base) amirreza@Amirrezas-MacBook-Pro SatSolver % python3 main.py
=== Generation 10 ===
Maximum Fitness = 400
=== Generation 20 ===
Maximum Fitness = 400
=== Generation 30 ===
Maximum Fitness = 400
=== Generation 40 ===
Maximum Fitness = 404
=== Generation 50 ===
Maximum Fitness = 404
=== Generation 60 ===
Maximum Fitness = 406
=== Generation 70 ===
Maximum Fitness = 408
=== Generation 80 ===
Maximum Fitness = 409
=== Generation 90 ===
Maximum Fitness = 409
=== Generation 100 ===
Maximum Fitness = 410
=== Generation 110 ===
Maximum Fitness = 411
```

```
=== Generation 570 ===
Maximum Fitness = 420
=== Generation 580 ===
Maximum Fitness = 420
=== Generation 590 ===
Maximum Fitness = 420
=== Generation 600 ===
Maximum Fitness = 420
=== Generation 610 ===
Maximum Fitness = 420
=== Generation 620 ===
Maximum Fitness = 420
=== Generation 630 ===
Maximum Fitness = 420
=== Generation 640 ===
Maximum Fitness = 421
=== Generation 650 ===
Maximum Fitness = 421
=== Generation 660 ===
Maximum Fitness = 421
=== Generation 670 ===
Maximum Fitness = 421
=== Generation 680 ===
```

```
Maximum Fitness = 427
=== Generation 890 ===
Maximum Fitness = 427
=== Generation 900 ===
Maximum Fitness = 427
=== Generation 910 ===
Maximum Fitness = 427
=== Generation 920 ===
Maximum Fitness = 427
=== Generation 930 ===
Maximum Fitness = 427
=== Generation 940 ===
Maximum Fitness = 427
=== Generation 950 ===
Maximum Fitness = 427
=== Generation 960 ===
Maximum Fitness = 427
=== Generation 970 ===
Maximum Fitness = 427
=== Generation 980 ===
Maximum Fitness = 427
=== Generation 990 ===
Maximum Fitness = 427
```