دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

گزارش سوم

پیاده سازی جستجوی محلی برای کمینه سازی مسائل

نگارش

سروش آریانا

استاد

دکتر مهدی قطعی

فروردین ۱۴۰۰

مقدمه

در این گزارش، به مقایسه دو الگوریتم ژنتیک و تبرید شبیه‌سازی شده در مساله ۸-وزیر پرداختیم.

۱- تابع fitness

برای حل مساله ۸-وزیر تابع fitness ما در واقع همان تعداد وزیرهایی است که مورد حمله واقع شده‌اند. در روشی که ما برای حل الگوریتم ژنتیک و تبرید استفاده کردیم ما همه وزیرها را در صفحه قرار می‌دهیم و در جهت کم کردن تعداد وزیرهایی که مورد حمله‌اند حرکت می‌کنیم.

برای نمایش بورد بازی هم از یک روش خیلی ساده استفاده کردیم بصورتی که محل قرارگیری هر کدام از ۸ وزیر در هر ستون را در یک آرایه ۸تایی ذخیره کردیم. مثلا آرایه [۱, ۶, ۲, ۵, ۷, ۴, ۰, ۳] به این معنی است که اولین وزیر در اولین ستون و ردیف دوم قرار دارد، دومین وزیر در دومین ستون و ۷امین ردیف قرار دارد و ... .

در این روش ما نگران حملات در یک ستون نیستیم چون دو وزیر در یک ستون نمیتوانند قرار بگیرند. برای بررسی حملات در یک ردیف کافی است بصورت دو به دو مقادیر ایندکس‌ها را بررسی کنیم؛ در صورت مساوی بودن دو مقدار یعنی حمله رخ داده است. برای بررسی حملات قطری، قدر مطلق اختلاف شماره ستون‌ها و شماره ردیف را بررسی میکنیم. در صورت برابر بودن دو مقدار یعنی حمله رخ داده است. کد تابع به شرح زیر است و از همین پیاده‌سازی در هردو الگوریتم استفاده شده است:

```
def fitness(board):
    total_under_attack = 0
    # Count how many queens are in same index for horizontal collision
    for x1, y1 in enumerate(board):
        for x2, y2 in enumerate(board):
            if x1 != x2 and (y1 == y2 or abs(x1-x2) == abs(y1-y2)):
                total_under_attack = total_under_attack + 1
    return total_under_attack
```

کد ۱ تابع fitness

۲- پیاده سازی الگوریتم ژنتیک

در روش ژنتیک، در هربار اجرای الگوریتم ابتدا یک احتمال با توجه جواب تابع fitness روی هر عضو نسل قبل نسبت می‌دهیم. سپس بر اساس این احتمالات دو جواب از نسل قبل را انتخاب کرده و یک فرزند از روی آن‌ها تولید می‌کنیم. برای تولید فرزند، بصورت رندوم بخشی از فرزند را از روی والد اولد و بخشی را از روی والد دوم کپی میکنیم. همچنین برای دوری از گیر کردن در نقطه مینیمم نسبی، بصورت رندوم بعضی از فرزندهای تولید شده را جهش می‌دهیم به اینصورت که یک بیت رندوم از آنها را تغییر می‌دهیم. پارامتر احتمال جهش در هر فرزند را ۰,۲ مقداردهی شده است.

۳- پیاده سازی الگوریتم تبرید شبیه سازی شده

در روش تبرید شبیه‌سازی شده، درهربار اجرای الگوریتم ابتدا یک بورد جایگزین تولید می‌کنیم به این صورت که بصورت رندوم یکی از وزیرهای بورد قبلی را جابجا می‌کنیم. سپس تفاوت مقدار fitness روی بورد قبلی و بورد فعلی را حساب می‌کنیم. درصورتی که fitness کم شده بود (یعنی تعداد حملات کم شده بود) بورد جدید جایگزین می‌شود. همچنین در هربار اجرا دما بر اساس شماره اجرا کم می‌شود و اگر fitness بورد جدید بیشتر از حالت قبلی بود بر اساس احتمالی که از میزان دما بدست می‌آید ممکن است با بورد قبلی جایگزین شود و با این روش از گیر افتادن در مینیمم نسبی جلوگیری می‌کنیم.

۴- مقایسه دو الگوریتم از نظر زمان و حافظه مصرفی

در پیاده سازی ما برای الگوریتم ژنتیک پارامتر احتمال جهش ۰,۲ مقداردهی شده است. همچنین برای پیاده سازی الگوریتم تبرید شبیه سازی شده مقدار اولیه دما ۴۰۰۰ در نظر گرفته شده است.

	1	2	3	4	5	Average
Genetic	11370496	11337728	11206656	11157504	11141120	11,242,700.8
SA	11206656	11272192	11239424	11255808	11337728	11,262,361.6

جدول ۱ مقایسه میزان رم مصرفی الگوریتم ژنتیک و تبرید شبیه‌سازی شده در مساله ۸-وزیر بر اساس بایت

	1	2	3	4	5	Average
Genetic	0.158427	0.144792	0.771474	1.600170	0.207842	0.576541
SA	0.043622	0.037510	0.133856	0.070531	0.057200	0.0685438

جدول ۲ مقایسه زمان اجرای الگوریتم ژنتیک و تبرید شبیه‌سازی شده در مساله ۸-وزیر بر اساس ثانیه

همانطور که در مقایسه دو الگوریتم مشاهده می‌شود، هر دو الگوریتم تقریبا به اندازه مساوی رم استفاده کرده‌اند. همچنین زمان اجرای الگوریتم بصورت کلی وابسته به روند اجرای الگوریتم است و در هربار

اجرای الگوریتم ممکن است مقادیر متفاوتی به دست بیاید اما بصورت میـانگین الگـوریتم تبریـد شـبیه-سازی شده بطور قابل توجهی زمان اجرای کمتری در آزمایش‌های ما داشت.

منابع و مراجع

8 Queen Puzzle Optimization Using a Genetic Algorithm in Python. (2021, 4 4). Retrieved from Towards Data Science: https://heartbeat.fritz.ai/8-queen-puzzle-optimization-using-a-genetic-algorithm-in-python-1d9ca769ede8

Checking 2-dimensional array. (2021, 4 3). Retrieved from Stackoverflow: https://stackoverflow.com/questions/384874/checking-2-dimensional-array-like-eight-queens-puzzle

Genetic algorithm . (2021, 4 4). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Genetic_algorithm

How can I formulate 8-Puzzle for SA algorithm? . (2021, 4 3). Retrieved from Stackoverflow: https://stackoverflow.com/questions/23032305/how-can-i-formulate-8-puzzle-for-sa-algorithm

Simulated annealing . (2021, 4 4). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Simulated_annealing

پیوست‌ها

سورس کد حل مساله ۸-وزیر با الگوریتم ژنتیک

```python
#!/usr/bin/env python

import os
import psutil
import random
import time

N = 8
POPULATION = 100

maxFitness = 28


# Counting how many queens are under attack
def fitness(board):
    total_under_attack = 0
    # Count how many queens are in same index for horizontal collision
    for x1, y1 in enumerate(board):
        for x2, y2 in enumerate(board):
            if x1 != x2 and (y1 == y2 or abs(x1-x2) == abs(y1-y2)):
                total_under_attack = total_under_attack + 1
    return total_under_attack


def probability(board, fitness):
    return (28 - fitness(board)) / maxFitness


def random_pick(population, probabilities):
    total = sum(w for c, w in zip(population, probabilities))
    r = random.uniform(0, total)
```

```
    upto = 0
    for c, w in zip(population, probabilities):
        if upto + w >= r:
            return c
        upto += w


def crossover(x, y):
    c = random.randint(0, N - 1)
    return x[0:c] + y[c:N]


def mutate(x):
    c = random.randint(0, N - 1)
    m = random.randint(0, N - 1)
    x[c] = m
    return x


def genetic(population, fitness):
    mutation_probability = 0.2
    new_population = []
    probabilities = [probability(n, fitness) for n in population]
    for i in range(len(population)):
        x = random_pick(population, probabilities)
        y = random_pick(population, probabilities)
        child = crossover(x, y)
        if random.random() < mutation_probability:
            child = mutate(child)
        # print_individual(child)
        new_population.append(child)
```

```
        if fitness(child) == 0: break
    return new_population


def print_board(x):
    print("{},  collision = {}, probability = {:.6f}"
          .format(str(x), fitness(x), probability(x, fitness)))


if __name__ == "__main__":
    t0 = time.time()

    # Populate first generation
    population = [[random.randint(0, N - 1) for _ in range(N)] for _ in range(POPULATION)]

    i = 0
    while 0 not in [fitness(x) for x in population]:
        population = genetic(population, fitness)
        print("Min collision in generation {} = {}".format(i, min([fitness(n) for n in population])))
        i += 1

    print("Solved in Generation {}!".format(i - 1))
    for x in population:
        if fitness(x) == 28:
            print_board(x)

    process = psutil.Process(os.getpid())
    print("\nRam usage: {} bytes".format(process.memory_info().rss))
    print("Time: {} seconds".format(time.time()-t0))
```

سورس کد حل مساله ۸-وزیر با الگوریتم تبرید شبیه سازی شده(simulated annealing)

```python
from math import exp
import random
import os
import psutil
import time

N = 8
T = 4000

def fitness(board):
    total_under_attack = 0
    # Count how many queens are in same index for horizontal collision
    for x1, y1 in enumerate(board):
        for x2, y2 in enumerate(board):
            if x1 != x2 and (y1 == y2 or abs(x1 - x2) == abs(y1 - y2)):
                total_under_attack = total_under_attack + 1
    return total_under_attack


def print_board(k, x):
    print("k = {}, {},  collision = {}".format(k, str(x), fitness(x)))


if __name__ == "__main__":
    t0 = time.time()

    board = [random.randint(0, N - 1) for _ in range(N)]

    k = 0
    temperature = T
    while True:
```

```
        k = k + 1
        temperature = T / float(k + 1)
        successor_board = board.copy()
        while successor_board == board:
            random_index = random.randint(0, N-1)
            random_value = random.randint(0, N-1)
            successor_board[random_index] = random_value
        diff = fitness(successor_board) - fitness(board)
        metropolis = exp(-diff / temperature)
        if diff < 0 or random.uniform(0, 1) < metropolis:
            board = successor_board
        if fitness(board) == 0:
            print('\nSolution Found:')
            print_board(k, board)
            break
        elif k % 100 == 0:
            print_board(k, board)

    process = psutil.Process(os.getpid())
    print("\nRam usage: {} bytes".format(process.memory_info().rss))
    print("Time: {} seconds".format(time.time()-t0))
```