حل مسئله Building Scale با استفاده از Genetics Algorithm

درس رایانش تکاملی – مدرس: دکتر روحانی

نویسنده: آریا عربان

هدف و راه حل

در این تمرین، خواسته شده که در جهت یافتن پاسخ بهینه‌ای برای مسئله Building Scale(یک مسئله NP-Hard و از نمونه های مسئله QAP) تلاش شود. برای این کار، از الگوریتم ژنتیک استفاده می‌شود. هدف در این مسئله این است که هزینه بین موقعیت قرارگیری دپارتمان ها مینیمم شود. هزینه بصورت حاصل ضرب ماتریس‌های flow وdistance تعریف می شود:

$$\text{Minimize} \sum_{i=1}^{n_{f-1}} \sum_{j=i+1}^{n_{f-1}} f_{ij}\, d_{ij}$$

در صورت مسئله، فرض شده است تعداد 25 دپارتمان در 25 مکان شامل 5 ردیف مستقر می‌شوند. همچنین ماتریس distance و ماتریس flow(که ماتریس‌هایی با ابعاد 25*25 هستند) در اختیار قرار داده شده‌اند.

دقت می‌شود که برای حل این مسئله با اسفاده از GA، هر عضوی از جمعیت به صورت جایگشت 25 تایی تعریف می‌شود، که خانه iم به معنی موقعیت قرارگیری iام، و عدد داخل این خانه به معنی دپارتمان قرار گرفته در این خانه است.

برای انجام عمل انتخاب، از روش Tournament که در تمرین سری پیش پیاده سازی شده است، استفاده خواهد شد.

از آنجایی که در این مسئله، هر عضو جمعیت دارای جایگشت می‌باشد، باید روش مناسبی برای انجام عملیات تقاطع و جهش ارائه شود، که در ادامه به آن پرداخته می‌شود.

Edge Recombination Crossover (ERX):

این روش، یکی از مناسبترین روش‌های تقاطع برای مسائل جایگشتی می‌باشد. این روش تا حد امکان از لبه‌ها یا اتصالات گره‌ای موجود برای تولید فرزندان استفاده می‌کند، و مزیت اصلی این روش در برابر روش‌های تقاطع جایگشتی مانند PMX و Ordered Crossover در همین است. عموما روش ERX، نتیجه بهتری از دگر دو روش ذکر شده خواهد داشت، و تنها عیب آن در این است که معمولا زمان محاسبه بیشتری خواهد داشت.

روند الگوریتم ERX به شکل زیر می‌باشد:

لیست همسایگی های موجود هر گره را با توجه به دو والد ایجاد شود.

کروموزوم خالی = CHILD

1. اولین گره از یک والد تصادفی = X
2. تکرار تا زمانی که کروموزومCHILD پر نشده است:

- X به CHILD اضافه شود
- X از تمامی لیست های همسایگی حذف شود

اگر لیست همسایگی X خالی بود:

- گره تصادفی‌ای که در CHILD وجود ندارد = Z

در غیر این صورت:

- همسایه ای از X که کمترین تعداد همسایه دارد انتخاب شود
- اگر بیش از یک همسایه وجود داشت، به طور تصادفی یکی از آنها انتخاب شود
- گره انتخابی = Z

X = Z

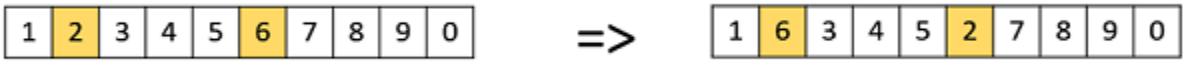
همان گونه که مشاهده می‌شود، در این الگوریتم تنها یک فرزند تولید می‌شود. برای ایجاد دو فرزند، می‌توان مجدد همین الگوریتم تکرار شود، با این تفاوت که والد انتخاب شده برای انتقال گره اول، والد مخالف با والد انتخاب شده برای فرزند اول باشد.

:Swap Mutation

این روش جهش، یکی از رایجترین روش‌ها برای انجام جهش در مسائل جایگشتی می‌باشد.

روش کار آن ساده است. دو موقعیت در فرزند به طور تصادفی انتخاب می‌شود، و مقادیر این دو موقعیت باهم عوض می‌شوند.

مثال:



شرح نتایج

ابتدا با در نظر گرفتن مقدار ثابت احتمال تقاطع 0.7، احتمال جهش 0.1، و اندازه جمعیت 250، از کد ده دفعه اجرا گرفته می‌شود، و نتایج بدست آمده به شکل زیر می‌باشد.

iteration	best_fitness_score	best_solution
1	3950	[11, 19, 18, 0, 12, 3, 2, 9, 6, 14, 24, 7, 15, 5, 13, 1, 10, 8, 22, 20, 4, 16, 21, 17, 23]
2	3894	[4, 19, 24, 1, 11, 12, 14, 22, 3, 16, 21, 9, 8, 7, 10, 18, 5, 13, 2, 17, 0, 6, 15, 20, 23]
3	3860	[23, 17, 10, 1, 4, 20, 2, 7, 24, 16, 15, 5, 8, 13, 11, 6, 9, 21, 22, 14, 0, 18, 19, 3, 12]
4	3786	[4, 1, 24, 16, 11, 23, 17, 15, 2, 3, 10, 7, 5, 6, 13, 20, 8, 18, 9, 22, 14, 21, 0, 19, 12]
5	3894	[12, 4, 14, 10, 11, 22, 7, 2, 24, 16, 21, 8, 5, 13, 17, 0, 18, 6, 15, 3, 1, 19, 9, 20, 23]
6	3908	[12, 3, 22, 9, 18, 16, 11, 2, 6, 0, 23, 13, 20, 5, 15, 17, 24, 10, 7, 8, 4, 1, 19, 14, 21]
7	3876	[4, 19, 21, 7, 11, 1, 22, 8, 24, 16, 14, 9, 2, 10, 3, 0, 18, 5, 13, 17, 12, 6, 15, 20, 23]
8	3892	[12, 6, 15, 18, 0, 13, 20, 5, 9, 2, 22, 23, 8, 7, 17, 3, 10, 24, 19, 1, 11, 16, 14, 21, 4]
9	3862	[3, 22, 16, 11, 4, 13, 2, 7, 17, 1, 23, 9, 5, 19, 10, 20, 6, 18, 15, 24, 12, 0, 8, 21, 14]
10	3876	[4, 1, 24, 14, 12, 10, 7, 8, 21, 22, 13, 5, 15, 6, 0, 20, 2, 9, 19, 18, 23, 17, 3, 16, 11]

با فرض مقادیر ثابت 250 برای اندازه جمعیت و 0.1 برای احتمال جهش، با در نظر گرفتن مقادیر مختلف برای احتمال تقاطع، نتیجه به شکل زیر می‌باشد:

p_crossover	best_fitness_score	best_solution
0.5	3836	[4, 1, 10, 0, 23, 3, 24, 7, 13, 11, 21, 2, 8, 22, 16, 19, 9, 5, 6, 20, 14, 17, 18, 15, 12]
0.7	3834	[4, 2, 17, 1, 10, 3, 23, 13, 24, 7, 11, 21, 5, 15, 8, 19, 9, 6, 18, 0, 12, 16, 22, 14, 20]
0.85	3920	[4, 1, 10, 20, 23, 19, 24, 7, 13, 3, 14, 15, 5, 2, 17, 0, 8, 6, 9, 11, 21, 18, 22, 12, 16]

در ادامه، با فرض مقادیر ثابت 0.7 برای احتمال تقاطع و 250 برای اندازه جمعیت، با در نظر گرفتن مقادیر مختلف برای احتمال جهش، نتیجه به شکل زیر می‌باشد:

p_mutation	best_fitness_score	best_solution
0.05	3870	[11, 4, 1, 2, 17, 16, 22, 24, 7, 10, 21, 8, 15, 5, 13, 12, 18, 6, 9, 3, 14, 0, 19, 20, 23]
0.1	3860	[4, 1, 2, 17, 10, 14, 7, 5, 13, 23, 0, 8, 6, 15, 20, 19, 18, 9, 24, 3, 12, 21, 22, 16, 11]
0.25	3928	[17, 20, 16, 22, 12, 1, 24, 2, 13, 7, 4, 3, 6, 5, 8, 10, 15, 18, 0, 14, 23, 11, 19, 9, 21]

در نهایت، با فرض مقادیر ثابت 0.7 برای احتمال تقاطع و 0.1 برای احتمال جهش، با در نظر گرفتن مقادیر مختلف برای احتمال جمعیت، نتیجه به شکل زیر می‌باشد:

N_Population	best_fitness_score	best_solution
250	3874	[4, 1, 10, 17, 23, 11, 16, 24, 15, 20, 2, 22, 8, 7, 13, 3, 9, 18, 5, 6, 19, 21, 14, 0, 12]
370	3826	[12, 0, 18, 14, 21, 9, 5, 6, 15, 19, 20, 13, 8, 3, 16, 22, 10, 7, 24, 11, 23, 17, 2, 1, 4]
500	3870	[4, 11, 16, 17, 23, 1, 24, 15, 10, 20, 19, 2, 6, 13, 3, 21, 7, 8, 5, 9, 14, 22, 18, 0, 12]
750	3890	[4, 1, 17, 10, 23, 11, 3, 24, 20, 16, 2, 6, 15, 12, 13, 22, 18, 8, 7, 9, 5, 0, 14, 19, 21]

با توجه به اجراهایی که از کد با مقادیر مختلف پارامترها گرفته شد، بنظر می‌آید که نتایج بدست آمده نزدیک به مقدار بهینه هستند، و بهترین نتایج در حالتی بدست آمده‌اند که مقادیر احتمال تقاطع برابر 0.7، احتمال جهش برابر 0.1، و اندازه جمعیت برابر 250 بوده‌اند.

بهترین نتیجه در اجرا ششم از ده اجرا ابتدای این قسمت بدست آمده است. که مقدار امتیاز در این اجرا برابر 3786 بوده.

قابل بیان است که اگر به جای استفاده از روش انتخاب tournament، از روش انتخاب roulette wheel استفاده شده بود، میانگین امتیازهای اجرا حدودا برابر 4100 می‌شد، که این نتیجه ضعیفتر از نتایج بدست آمده با روش tournament می‌باشد.

ضمیمه (کد) :

Github Repo

پیاده سازی تابع هدف:

```python
def obj_func(v):
    matrices_size = distance_matrix.shape[0]
    fitness_sum = 0
    for x in range(matrices_size):
        for y in range(matrices_size):
            fitness_sum += distance_matrix[x, y] * flow_matrix[v[x], v[y]]
    return fitness_sum
```

پیاده سازی GA برای مسئله QAP:

```python
def _tournament_selection(pop, scores, k=3):
    # first random selection
    selection_ix = randint(len(pop))
    for ix in randint(0, len(pop), k - 1):
        # check if better (e.g. perform a tournament)
        if scores[ix] < scores[selection_ix]:
            selection_ix = ix
    # print(pop[selection_ix])
    return pop[selection_ix]


def _roulette_selection(pop, scores):
    if np.min(scores) < 0:
        scores = scores - np.min(scores)

    scores = np.max(
        scores) - scores  # this makes smaller scores mean better instead of traditional higher score better.
    # first random selection
    div_scores = scores / np.sum(scores)
    rnd_num = random.uniform(0, 1)
    idx = -1
    for i in range(len(scores)):
        rnd_num = rnd_num - div_scores[i]
        if (rnd_num < 0):
            idx = i
            break
    return pop[idx]
```

```
def _erx_crossover(p1, p2, r_cross):
    if np.random.rand() > r_cross:
        return [p1, p2]

    c1, c2 = [], []

    for idx, cur_child in enumerate([c1, c2]):
        neighbourhoods = _get_neighbourhoods(p1, p2)
        chosen = eval(f"p{idx + 1}[0]")
        cur_child.append(chosen)
        while len(cur_child) != len(p1):

            neighbourhoods = remove_values_from_lists(neighbourhoods, chosen)
            d = {x: len(neighbourhoods[x]) for x in neighbourhoods[chosen]}

            if bool(d):
                min_val = min(d.values())
                mn1 = list(filter(lambda x: d[x] == min_val, d))
                chosen = random.choice(mn1)
                cur_child.append(chosen)
            else:
                for num in p1:
                    if num not in cur_child:
                        cur_child.append(num)
    return [c1, c2]


# mutation operator
def _swap_mutation(child, r_mut):
    # check for a mutation
    if rand() < r_mut:
        swap_random(child)

    return child
```

```python
def _get_neighbourhoods(list1, list2):
    """This function takes as input two parents which has only unique values 0 to N,
     and output the 2D neighbourhood list.  0's neighbours will be the first list,
      1's neighbours will be the second list, and so on...
    """
    N = len(list1)
    arr = [[] for _ in range(len(list1))]
    for i in range(N):
        idx1 = list1.index(i)
        idx2 = list2.index(i)

        arr[i].extend({list1[idx1 - 1], list1[(idx1 + 1) % N], list2[idx2 - 1], list2[(idx2 + 1) % N]})

    return arr
```

```python
# genetic algorithm
def genetic_algorithm(objective, perm_n, n_iter, n_pop, r_cross, r_mut):
    # initial population of random numbers in permutation
    pop = [np.random.permutation(perm_n).tolist() for _ in range(n_pop)]
    # keep track of best solution

    best, best_eval = 0, objective(pop[0])

    # enumerate generations
    for gen in range(n_iter):
        # evaluate all candidates in the population
        scores = [objective(d) for d in pop]
        # check for new best solution
        for i in range(n_pop):
            if scores[i] < best_eval:
                best, best_eval = pop[i], scores[i]
                print(">%d, new best f(%s) = %f" % (gen, pop[i], scores[i]))
        # select parents
        selected = [_roulette_selection(pop, scores) for _ in range(n_pop)]
        # create the next generation
        children = list()
        for i in range(0, n_pop, 2):
            # get selected parents in pairs
            p1, p2 = selected[i], selected[i + 1]
            # crossover and mutation
            for c in _erx_crossover(p1, p2, r_cross):
                # mutation
                chd = _swap_mutation(c, r_mut)
                # store for next generation

                children.append(chd)
        # replace population
        pop = children
    return [best, best_eval]
```

توابع کمکی:

```python
def read_lines_into_arr(file_name, line_begin, line_end):
    with open(f"{file_name}") as lines:
        array = np.genfromtxt(islice(lines, line_begin - 1, line_end))
        return array


def remove_values_from_lists(lists, val):
    # this function removes values from list of lists, and also returns the index list
    return [[value for value in i if value != val] for i in lists]


def swap_random(seq, repetition=1):
    # swap two random numbers in array
    idx = range(len(seq))
    for i in range(repetition):
        i1, i2 = random.sample(idx, 2)
        seq[i1], seq[i2] = seq[i2], seq[i1]
```

اسکریپت اصلی:

```python
def GA_main(population_size, n_perm, p_crossover, p_mutation):
    # define the total iterations
    n_iter = 1000
    # define the population size
    n_pop = population_size
    # crossover rate
    r_cross = p_crossover
    # mutation rate
    r_mut = p_mutation

    best, score = genetic_algorithm(obj_func, n_perm, n_iter, n_pop, r_cross, r_mut)

    return best, score


def main():

    best_solution, objective_function_score = GA_main(population_size=370,
                                                      n_perm=25, p_crossover=0.7, p_mutation=0.1
    print(f"best_solution:\n {best_solution}")
    print(f"objective_function:\n {objective_function_score}")
    print("------------")


if __name__ == '__main__':
    main()
```