به نام خدا

درس مبانی هوش محاسباتی

تمرین سری سوم

مدرس درس:
جناب آقای دکتر مزینی

تهیه شده توسط:
الناز رضایی ۹۸۴۱۱۳۸۷

تاریخ ارسال: ۱۴۰۱/۱۱/۰۱

سوال ۱ :

در این سوال به طور کامل الگوریتم کلوني مورچه‌ها و نحوه همگرايي آن را توضیح دهید. يك مثال کوچك بزنید و مراحل همگرايي را توضیح دهید. (۲۰ نمره)

پاسخ ۱:

- الگوریتم و نحوه همگرایی: در الگوریتم‌های بهینه‌سازی کلونی مورچه‌ها، یک مورچه مصنوعی یک عامل محاسباتی ساده است که به دنبال راه‌حل‌های خوب برای یک مسئله بهینه‌سازی معین می‌گردد. برای اعمال الگوریتم کلونی مورچه‌ها، مسئله بهینه‌سازی باید به مسئله یافتن کوتاه‌ترین مسیر در یک نمودار وزن‌دار تبدیل شود. در مرحله اول هر تکرار، هر مورچه به طور تصادفی یک راه‌حل می‌سازد، یعنی به ترتیبی که لبه‌های نمودار باید دنبال شود. در مرحله دوم، مسیرهای پیدا شده توسط مورچه‌های مختلف با هم مقایسه می‌شوند. آخرین مرحله شامل به روز رسانی سطوح pheromone در هر لبه است.

```
procedure ACO_MetaHeuristic is
    while not terminated do
        generateSolutions()
        daemonActions()
        pheromoneUpdate()
    repeat
end procedure
```

پروسهٔ پیدا کردن کوتاه‌ترین مسیر توسط مورچه‌ها، ویژگی‌های بسیار جالبی دارد، اول از همه قابلیت تعمیم زیاد و خودـ سازمانده بودن آن است. در ضمن هیچ مکانیزم کنترل مرکزی ای وجود ندارد. ویژگی دوم قدرت زیاد آن است. سیستم شامل تعداد زیادی از عواملی است که به تنهایی بی‌اهمیت هستند بنابراین حتی تلفات یک عامل مهم، تأثیر زیادی روی کارایی سیستم ندارد. سومین ویژگی این است که، پروسه یک فرایند تطبیقی است. از آنجا که رفتار هیچ‌کدام از مورچه‌ها معین نیست و تعدادی از مورچه‌ها همچنان مسیر طولانی‌تر را انتخاب می‌کنند، سیستم می‌تواند خود را با تغییرات محیط منطبق کند و ویژگی آخر اینکه این پروسه قابل توسعه

است و می‌تواند به اندازهٔ دلخواه بزرگ شود. همین ویژگی‌ها الهام بخش طراحی الگوریتم‌هایی شده‌اند که در مسائلی که نیازمند این ویژگی‌ها هستند کاربرد دارند. اولین الگوریتمی که بر این اساس معرفی شد، الگوریتم ABC بود. چند نمونه دیگر از این الگوریتم‌ها عبارتند از: AntNet,ARA,PERA,AntHocNet.

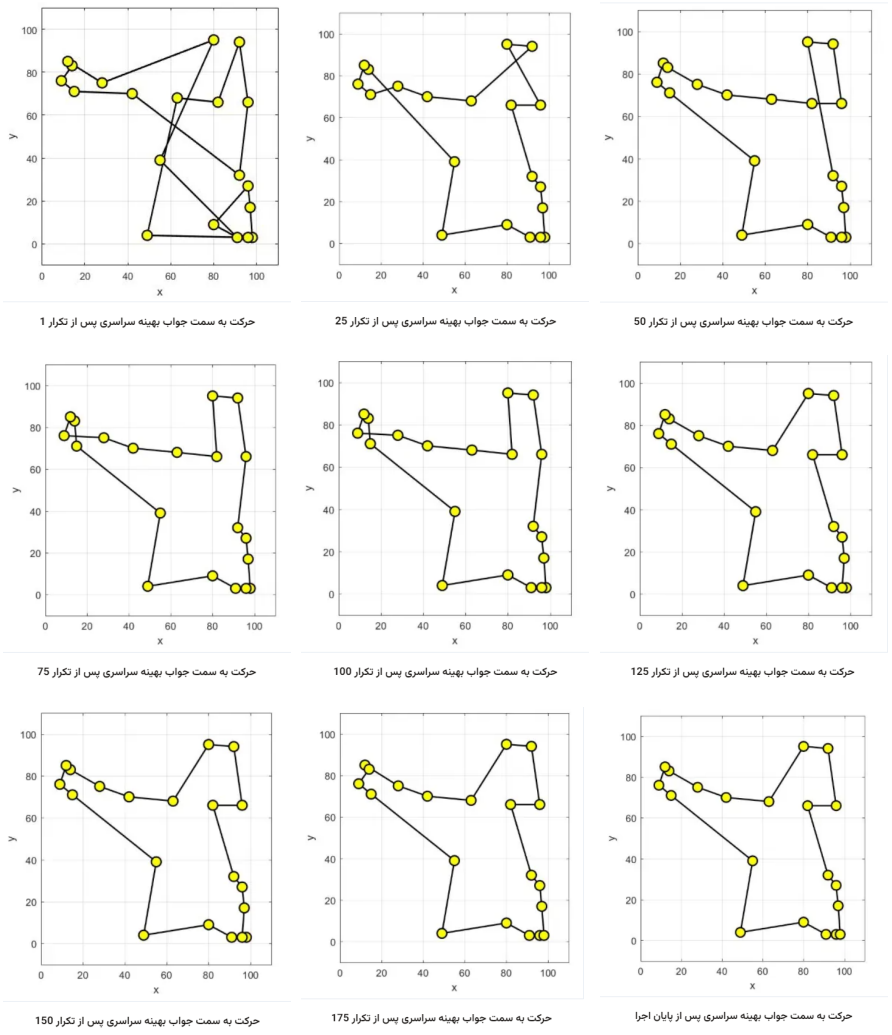
الگوریتم کلونی مورچگان از دو بخش تشکیل شده است. در بخش اول، مقادیر پارامترهای مسأله تنظیم و متغیرهای جمعیت اولیه مقداردهی می‌شوند. بخش دو شامل یک حلقه تکرار است که سه مرحله الگوریتم کلونی مورچگان را اجرا می‌کند. در هر حلقه از الگوریتم کلونی مورچگان، جواب‌های کاندید توسط تمامی مورچه‌های مصنوعی ساخته می‌شوند. جواب‌های تولید شده، از طریق یک الگوریتم جستجوی محلی بهبود می‌یابند. در مرحله آخر، pheromoneها به روز رسانی می‌شوند.

- مثال TSP: کاربرد بهینه‌سازی کلونی مورچگان برای حل مسأله فروشنده دوره‌گرد ساده و صریح است. حرکت میان شهرها (مکان‌ها)، مؤلفه‌های جواب کاندید است. از آنجا که هیچ محدودیتی در امکان حرکت از یک شهر به هر شهر دیگری وجود ندارد، گراف ساختاری تشکیل شده یک گراف کاملا متصل است و تعداد رأس‌های موجود در گراف برابر تعداد شهرهای تعریف شده در مسأله خواهد بود. همچنین، اندازه یال‌های گراف متناسب با فاصله شهرها (با رأس‌ها نمایش داده می‌شوند) از یکدیگر است. pheromone نیز متناظر با مجموعه یال‌ها E در گراف ساختاری خواهد بود.

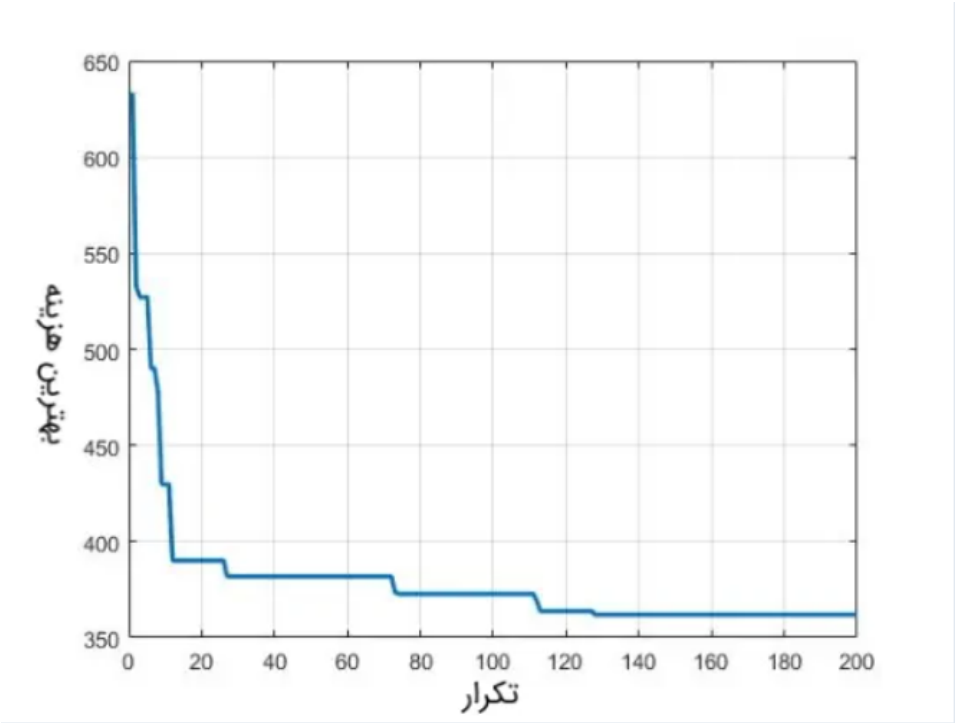
  هر کدام از مورچه‌ها، از یک شهر (یک رأس در گراف) کاملا تصادفی شروع می‌کنند. سپس، در هر گام از فرآیند تولید جواب، در راستای یال‌های گراف به حرکت می‌پردازند. هر مورچه، مسیر پیموده شده در گراف را به خاطر می‌سپارد و در گام‌های بعدی، یال‌هایی را برای حرکت در گراف انتخاب می‌کند که به مکان‌های (رأس‌های) از پیش پیموده شده منتهی نشوند. به محض اینکه تمامی رأس‌های گراف توسط یک مورچه پیمایش شد، یک جواب کاندید تولید می‌شود.

  در هر گام از فرآیند تولید جواب، مورچه‌ها به طور احتمالی، از میان یال‌های در دسترس (یال‌های پیموده نشده و منتهی به رأس‌هایی که از آن‌ها گذر نکرده)، یک یال را برای پیمایش انتخاب می‌کنند. نحوه محاسبه احتمال انتخاب یال‌ها، به پیاده‌سازی انجام شده از الگوریتم

کلونی مورچگان بستگی دارد. پس از اینکه تمامی مورچه‌ها یک جواب کاندید تولید کردند، فرومون روی یال‌ها، براساس «قانون به روز رسانی pheromone» به روز رسانی می‌شود.
شکل‌های زیر نحوه همگرایی روش بهینه‌سازی کلونی مورچگان به جواب بهینه سراسری، برای حل مسأله فروشنده دوره‌گرد را نمایش می‌دهند.



الگوریتم کلونی مورچگان، در تکرار ۱۲۸ به جواب بهینه سراسری (مقدار هزینه برابر با ۰۳۸.۳۶۲) همگرا شده است. شکل زیر، نحوه همگرایی به هزینه بهینه را نشان می‌دهد.



همگرایی به جواب بهینه سراسری در تکرار 128

لینک‌های استفاده شده در حل این سوال:

https://blog.faradars.org/ant-colony-optimization/

https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

سوال ۲:

يك دزد مي خواهد از جواهرفروشي تعدادي سنگ قيمتي سرقت كند. كوله پشتي او ۲۵ كيلوگرم ظرفيت دارد. جواهرات اين مغازه، قيمت و وزن سنگ‌ها به صورت زير است:

نام	وزن	ارزش
زمرد	۲	۳۰
نقره	۴	۱۰
یاقوت	۱	۲۰
الماس	۳	۵۰
برلیان	۵	۷۰
فیروزه	۱	۱۵
عقیق	۷	۴۰
کهربا	۴	۲۵

شکل ۱: اطلاعات سنگ‌ها

به او کمك كنيد تا كوله پشتي خود را به گونه‌اي بچيند كه مجموع وزن سنگ‌ها از ظرفيت كوله پشتي بيشتر نشود و در عين حال بيشترين ارزش را داشته باشد. او مي‌تواند از هر سنگ حداكثر يك عدد بردارد. كد اين بخش را با استفاده از الگوريتم ژنتيك پياده سازي كنيد.

پاسخ ۲:

برای حل این مسئله، باید مراحل زیر را انجام دهیم:

۱. initialize کردن جمعیت اولیه

۲. تعریف تابع fitness

۳. Selection

۴. Crossover

۵. Mutation

این چرخه از state ۲ تکرار می‌شود تا به پاسخ بهینه برسیم.
حال به پیاده‌سازی این الگوریتم می‌پردازیم.

۱. ابتدا تعداد، وزن و هزینه هر آیتم را تعریف می‌کنیم.

Items

```
[ ]  item_number = np.array([1, 2, 3, 4, 5, 6, 7, 8])
     weights = np.array([2, 4, 1, 3, 5, 1, 7, 4])
     costs = np.array([30, 10, 20, 50, 70, 15, 40, 25])
     weight_limit = 25
```

۲. initialize کردن جمعیت: در این بخش، جمعیت خود را به صورت یک ماتریس تعریف می‌کنیم که سطرهای آن، مربوط به هر عضو (کروموزوم)، و ستون‌های آن نشان دهنده ژن‌های کروموزوم می‌باشد.

1. Initialize Population

```
[11] pop_size = (8, item_number.shape[0])
     initial_population = np.random.randint(2, size = pop_size)
     initial_population = initial_population.astype(int)
     num_generations = 50
```

۳. تعریف تابع fitness: فرض کنیم بردار وزن w، بردار هزینه c، g کروموزوم و L حد وزن باشد. تابع fitness به شکل زیر تعریف می‌شود.

$$fitness(g) = \begin{cases} 0 & iff \quad \sum_{i=0}^{N-1} w_i g_i > L \\ \sum_{i=0}^{N-1} c_i g_i & otherwhise \end{cases}$$

2. Define fitness function

```
def fitness_calculator(w, c, L, pop):
  score1, score2 = 0
  fitness = np.empty(pop.shape[0])
  for i in range(pop.shape[0]):
    score1 = np.sum(pop[i] * c)
    score2 = np.sum(pop[i] * w)
    if score2 <= L:
      fitness[i] = score1
    else:
      fitness[i] = 0
  return fitness.astype(int)
```

۴. Selection: ایده این مرحله انتخاب این است که شایسته‌ترین افراد را انتخاب کنیم و اجازه دهیم ژن‌های خود را به نسل بعدی منتقل کنند.

3. Roulette Selection

```
[15] def selection(fitness, parents, pop):
       fitness = list(fitness)
       parents = np.empty((parents, pop.shape[1]))
       for i in range(parents):
         max_fitness_idx = np.where(fitness == np.max(fitness))
         parents[i,:] = pop[max_fitness_idx[0][0], :]
         fitness[max_fitness_idx[0][0]] = -999999
       return parents
```

۵. Crossover: از کد زیر برای crossover استفاده می‌کنیم.

4. Crossover

```
def crossover(parents, num_offsprings):
  offsprings = np.empty((num_offsprings, parents.shape[1]))
  crossover_point = int(parents.shape[1]/2)
  crossover_rate = 0.8
  i=0
  while (parents.shape[0] < num_offsprings):
    parent1_index = i%parents.shape[0]
    parent2_index = (i+1)%parents.shape[0]
    x = rd.random()
    if x > crossover_rate:
      continue
    parent1_index = i%parents.shape[0]
    parent2_index = (i+1)%parents.shape[0]
    offsprings[i,0:crossover_point] = parents[parent1_index,0:crossover_point]
    offsprings[i,crossover_point:] = parents[parent2_index,crossover_point:]
    i=+1
  return offsprings
```

۶. Mutation: برای mutation از رابطه زیر بهره می‌بریم.

$$r_i = \begin{cases} 0 & g_i = 1 \wedge \psi < p \\ 1 & g_i = 0 \wedge \psi < p \\ g_i & otherwise \end{cases}$$

5. Mutation

```
def mutation(offsprings):
  mutants = np.empty((offsprings.shape))
  mutation_rate = 0.4
  for i in range(mutants.shape[0]):
    random_value = rd.random()
    mutants[i,:] = offsprings[i,:]
    if random_value > mutation_rate:
      continue
    int_random_value = randint(0,offsprings.shape[1]-1)
    if mutants[i,int_random_value] == 0 :
      mutants[i,int_random_value] = 1
    else :
      mutants[i,int_random_value] = 0
    return mutants
```

حال با ترکیب تابع‌های بالا، GA خود را می‌سازیم تا به پاسخ بهینه برسیم.

Optimum Solution

```
def optimize(w, c, pop, pop_size, N, L):
    parameters, fitness_history = [], []
    num_parents = int(pop_size[0]/2)
    num_offsprings = pop_size[0] - num_parents
    for i in range(N):
      fitness = fitness_calculator(w, c, pop, L)
      fitness_history.append(fitness)
      parents = selection(fitness, num_parents, pop)
      offsprings = crossover(parents, num_offsprings)
      mutants = mutation(offsprings)
      pop[0:parents.shape[0], :] = parents
      pop[parents.shape[0]:, :] = mutants

    print('Last generation: \n{}\n'.format(pop))
    fitness_last_gen = fitness_calculator(w, c, pop, L)
    print('Fitness of the last generation: \n{}\n'.format(fitness_last_gen))
    max_fitness = np.where(fitness_last_gen == np.max(fitness_last_gen))
    parameters.append(pop[max_fitness[0][0],:])
    return parameters, fitness_history
```

آیتم‌های انتخاب شده ما توسط GA، زمرد، نقره، الماس، برلیان، عقیق و کهربا هستند که در مجموع ۲۳ کیلوگرم وزن دارند. خروجی الگوریتم، در شکل زیر قابل مشاهده است.

The Order Of Items

```
parameters, fitness_history = optimize(weights, costs, initial_population, pop_size, num_generations, weight_limit)
selected_items = item_number * parameters
print('\nSelected items that will maximize the knapsack without breaking it:')
for i in range(selected_items.shape[1]):
  if selected_items[0][i] != 0:
     print('{}\n'.format(selected_items[0][i]))
```

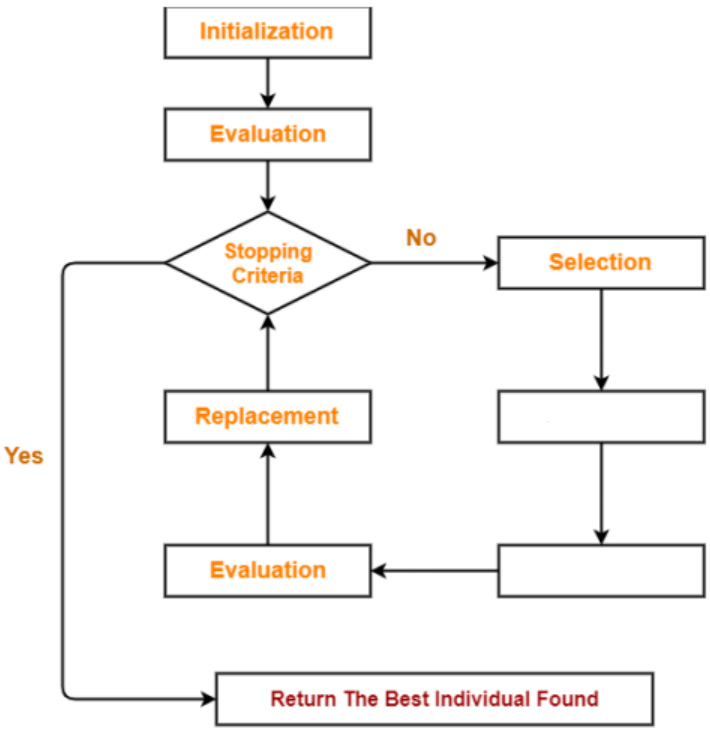
```
Selected items that will maximize the knapsack without breaking it:
1

2

4

5

7

8
```

لینک استفاده شده در حل این سوال:

https://medium.com/koderunners/genetic-algorithm-part-3-knapsack-problem-b59035ddd1d6

سوال ۳:

با توجه به اطلاعاتي از الگوريتم ژنتيك داريد فلوچارت مربوط به اين الگوريتم را كامل كنيد.



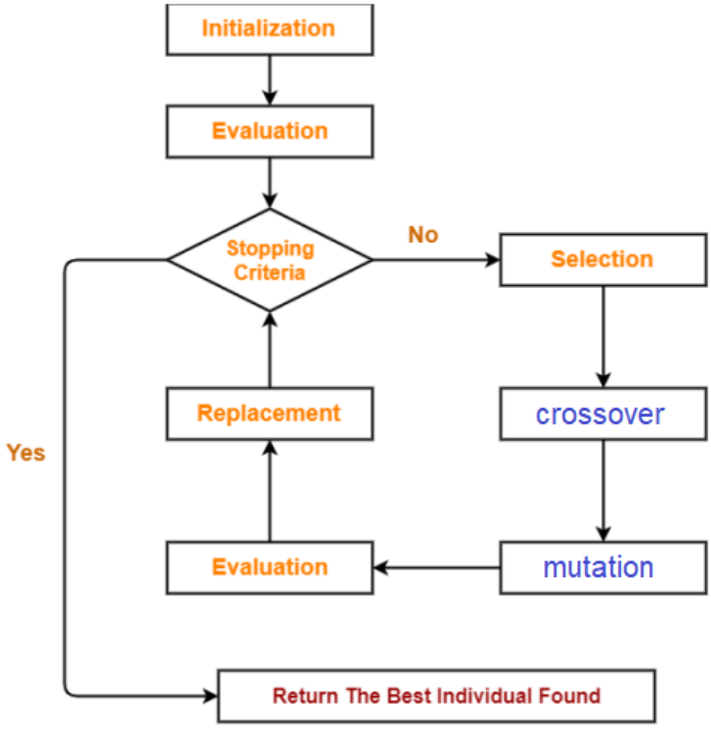
شكل ۲: فلوچارت الگوريتم ژنتيك

حال با توجه به اين مراحل يك چرخه از اين الگوريتم را براي ماكزيمم كردن رابطه

$$f(x) = x^2$$

وقتي كه مقدار x بين ۰ و ۵۰ باشد را به طور كامل بنويسيد.
دقت كنيد كه براي پاسخ اين سوال نياز نيست كه به جواب پاياني برسيد و تنها يك چرخه كافي است
اما تمامي مراحل را مانند نحوه cross over و ... را به طور واضح توضيح دهيد.
(براي راحتي كار با ۴ كروموزم اوليهي ۱۳و۲۴و۸و۱۹ شروع كنيد)

پاسخ ۳:



شکل ۲: فلوچارت الگوریتم ژنتیک

۱. هر راه‌حل را با یک عدد ۶ بیتی می‌توانیم نشان دهیم.

۲. جمعیت اولیه را با ۴ مقدار داده شده در سوال، شروع کرده و آن‌ها را به صورت باینری درمی‌آوریم. (۰۱۰۰۱۱، ۰۰۱۰۰۰، ۰۱۱۰۰۰، ۰۰۱۱۰۱)

۳. مقدار fitness را برای هر یک از اعضای جمعیت به دست می‌آوریم. برای ۴ عضو اولیه داریم: ۳۶۱، ۶۴، ۵۷۶ و ۳۶۱. میانگین آن‌ها ۲۹۳ و ماکزیمم مقدار، ۵۷۶ می‌باشد.

۴. احتمال selection هر عضو را با تقسیم بر مجموع جمعیت (۱۱۷۰) به دست می‌آوریم.

احتمال هر کدام به شکل زیر در می‌آید:

$$P(361) = 30.9\% \quad P(576) = 49.2\% \quad P(64) = 5.5\% \quad P(169) = 14.4\%$$

۵. یک عدد رندوم بین ۱ تا ۱۰۰۰ انتخاب می‌کنیم. اگر عدد کوچک‌تر از ۱۴۴ بود ۰۰۱۱۰۱، بین ۱۴۴ و ۶۳۶ بود ۰۱۱۰۰۰، بین ۶۳۷ و ۶۹۲ بود ۰۰۱۰۰۰ و در غیر این صورت ۰۱۰۰۱۱ را انتخاب می‌کنیم.

۶. فرض می‌کنیم با انتخاب مقادیر رندوم، رشته‌های ۰۰۱۱۰۱، ۰۱۱۰۰۰، ۰۱۱۰۰۰ و ۰۱۰۰۱۱ انتخاب شوند و ۰۰۱۰۰۰ به علت انتخاب نشدن از بین برود.

۷. با crossover تک نقطه‌ای، رشته‌های بالا را دو به دو ترکیب می‌کنیم. برای رشته اول و دوم، در بیت پنجم crossover را انجام می‌دهیم که رشته‌های ۰۱۱۰۰۱ و ۰۰۱۱۰۰ را تولید می‌کنند. برای دو رشته دیگر نیز بیت‌های سوم را جابه‌جا می‌کنیم که رشته‌های ۰۱۱۰۱۱ و ۰۱۰۰۰۰ را به وجود می‌آورند.

۸. با یک احتمال کوچک (0.001) هر بیت را تغییر می‌دهیم. با توجه به اینکه در اینجا ۲۴ بیت داریم، توقع داریم 0.024 بیت‌ها تغییر کنند. فرض می‌کنیم در این مثال تغییری نداشته باشیم.

۹. مقدار fitness را برای رشته‌های جدید به دست می‌آوریم که مقادیر ۱۴۴، ۶۲۵، ۷۲۹ و ۲۵۶ را با میانگین ۴۳۹ و ماکزیمم مقدار ۷۲۹ به ما می‌دهد. بنابراین به جواب خوبی رسیدیم و الگوریتم را متوقف می‌کنیم.

لینک استفاده شده در حل این سوال:

https://www.uobabylon.edu.iq/eprints/paper_2_27286_124.pdf