

Применение генетических алгоритмов, генерируемых искусственным интеллектом в задачах биоинформатики

Акентьев Данила Денисович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данной работе исследуется применение генетического алгоритма, сгенерированного искусственным интеллектом для поиска оптимальной последовательности нуклеотидов, соответствующей заданной белковой структуре. Целью исследования является сгенерировать при помощи ИИ (искусственный интеллект) эффективный метод для поиска последовательности нуклеотидов, наиболее близкой к заданной аминокислотной последовательности белка. Методика генетического алгоритма применяется для генерации и оценки приспособленности последовательностей, с последующим выбором наилучших кандидатов для формирования следующего поколения.

Ключевые слова: генетический алгоритм, биоинформатика, последовательность нуклеотидов, белковая структура, оптимизация, искусственный интеллект, генерация.

Application of genetic algorithms generated by artificial intelligence in bioinformatics problems

Akentev Danila Denisovich

Sholom-Aleichem Priamursky State University

Student

Abstract

In this paper, we investigate the use of a genetic algorithm generated by artificial intelligence to find the optimal sequence of nucleotides corresponding to a given protein structure. The aim of the study is to generate an effective method using AI (artificial intelligence) to search for the sequence of nucleotides closest to a given amino acid sequence of a protein. The technique of the genetic algorithm is used to generate and evaluate the fitness of sequences, followed by the selection of the best candidates for the formation of the next generation.

Keywords: genetic algorithm, bioinformatics, nucleotide sequence, protein structure, optimization, artificial intelligence, generation.

1 Введение

1.1 Актуальность

С каждым годом объемы биологических данных значительно возрастают благодаря развитию высокопроизводительных методов секвенирования геномов и экспериментальных подходов в биологии. Это создает огромный потенциал для понимания биологических процессов на уровне генетической информации. Однако, чтобы полностью раскрыть этот потенциал, необходимо разработать эффективные методы анализа и интерпретации биологических данных.

В этом контексте поиск оптимальных последовательностей нуклеотидов для заданных белковых структур остается актуальной задачей в биоинформатике. Это важно для понимания структуры и функции белков, которые играют ключевую роль в большинстве биологических процессов. Оптимальные последовательности нуклеотидов могут использоваться для создания белков с улучшенными свойствами, такими как стабильность, активность и специфичность.

Кроме того, разработка эффективных методов поиска оптимальных последовательностей нуклеотидов имеет практическое значение для различных областей, включая медицину, фармацевтику, сельское хозяйство и промышленность. Например, это может помочь в создании новых лекарственных препаратов, улучшении сортов растений с улучшенными свойствами, а также в разработке биотехнологических продуктов.

Таким образом, разработка и применение эффективных методов поиска оптимальных последовательностей нуклеотидов остается актуальной задачей в контексте современных достижений в области биологии и биоинформатики.

1.2 Обзор исследований

В современном мире биоинформатики и технологий развиваются различные методы и подходы для решения сложных задач. Одним из таких подходов является использование генетических алгоритмов, которые показывают эффективность в различных областях, включая планирование пути, оптимизацию экспрессии белков и предсказание структуры белков и РНК.

Исследование Ю. Лая и Д. Цзян обращает внимание на проблему оптимизации маршрута движения "data mule" в беспроводных сенсорных сетях. Оно предлагает использовать генетический алгоритм для решения задачи планирования пути "data mule" с целью минимизации времени задержки сбора данных и энергопотребления сенсорных узлов [1]. М. Хантер, П. Юань и Д. Вавилала рассмотрели проблему оптимизации экспрессии белков в клетках млекопитающих. Исследование подчеркивает сложности, с которыми сталкиваются ученые при планировании экспериментов по производству рекомбинантных белков и неоднозначность выбора стратегии в данной области [2]. К. Кухьян, Н. Мансур и Х. Хачфе представляют алгоритм на основе scatter search для предсказания трехмерной

структуры белков. Авторы демонстрируют эффективность своего метода на примере белков из банка данных, подчеркивая хорошее качество полученных структур [3]. К. Визе, А. Дешен и Э. Хендрикс обсуждают эволюционный алгоритм для предсказания вторичной структуры РНК. Исследование проводит сравнение с другими методами и показывает превосходство RnaPredict в предсказании структур РНК [4]. Наконец, исследование авторов Ц. Ту и С. Яна предлагает новый генетический алгоритм для планирования пути мобильного робота. Авторы акцентируют внимание на его способности генерировать оптимальные и безопасные пути в различных средах [5].

В целом, данные исследования показывают широкий спектр применения генетических алгоритмов в различных областях биоинформатики и робототехники, а также их эффективность в решении разнообразных задач оптимизации и предсказания.

1.3 Цель исследования

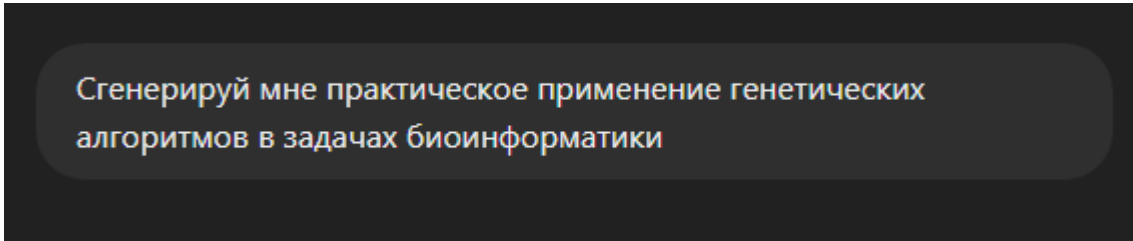
Целью исследования является сгенерировать при помощи ИИ (искусственный интеллект) эффективный метод для поиска последовательности нуклеотидов, наиболее близкой к заданной аминокислотной последовательности белка.

2 Материалы и методы

Для решения поставленной задачи использовался генетический алгоритм, сгенерированный искусственным интеллектом, который является эффективным методом оптимизации, вдохновленным принципами естественного отбора и генетической мутации.

3 Результаты и обсуждения

Выполнение анализа происходит на сайте Google Collab. Перед началом работы требуется сгенерировать выполнение практического задания у ИИ. Для этого обратимся в открытый чат с искусственным интеллектом ChatGPT (Рис.1).



Сгенерируй мне практическое применение генетических алгоритмов в задачах биоинформатики

Рисунок 1 – Запрос, введенный в ChatGPT

Он сгенерировал следующий код:

```
1. import random
2. import pandas as pd
3. import numpy as np
4. import matplotlib.pyplot as plt
5.
```

```
6. target_protein =
    "MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAH
    GKKVLGAFSDGLAHL DNLKGT FATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQA
    AYQKVVAGVANALAHKYH"
7.
8. def generate_population(pop_size, seq_length):
9.     population = [''.join(random.choices('ATCG', k=seq_length))
10.     for _ in range(pop_size)]
11.     return population
12.
13.     def fitness(sequence):
14.         score = sum(1 for a, b in zip(sequence, target_protein)
15.         if a == b)
16.         return score / len(target_protein)
17.
18.     def select_parents(population, fitness_scores):
19.         return random.choices(population,
20.         weights=fitness_scores, k=2)
21.
22.     def crossover(parent1, parent2):
23.         crossover_point = random.randint(0, len(parent1))
24.         child1 = parent1[:crossover_point] +
25.         parent2[crossover_point:]
26.         child2 = parent2[:crossover_point] +
27.         parent1[crossover_point:]
28.         return child1, child2
29.
30.     def mutate(sequence, mutation_rate):
31.         return ''.join(random.choice('ATCG') if random.random()
32.         < mutation_rate else base for base in sequence)
33.
34.     def genetic_algorithm(pop_size, seq_length, mutation_rate,
35.     num_generations):
36.         population = generate_population(pop_size, seq_length)
37.         fitness_progress = []
38.         average_fitness_progress = []
39.         best_sequence = None
40.         best_fitness = 0
41.
42.         for generation in range(num_generations):
43.             fitness_scores = [fitness(seq) for seq in
44.             population]
45.             current_best_sequence = max(population,
46.             key=fitness)
47.             current_best_fitness =
48.             fitness(current_best_sequence)
49.
50.             if current_best_fitness > best_fitness:
51.                 best_sequence = current_best_sequence
52.                 best_fitness = current_best_fitness
53.
54.             fitness_progress.append(current_best_fitness)
55.
56.             average_fitness_progress.append(np.mean(fitness_scores))
57.
58.             if best_fitness == 1.0:
59.                 break
60.
61.             new_population = []
62.             for _ in range(pop_size // 2):
63.                 parent1, parent2 = select_parents(population,
64.                 fitness_scores)
65.                 child1, child2 = crossover(parent1, parent2)
66.                 new_population.extend([mutate(child1,
67.                 mutation_rate), mutate(child2, mutation_rate)])
```

```
55.         population = new_population
56.
57.
58.         final_fitness_scores = [fitness(seq) for seq in
population]
59.         return best_sequence, best_fitness, fitness_progress,
average_fitness_progress, final_fitness_scores
60.
61.     population_size = 100
62.     sequence_length = len(target_protein)
63.     mutation_rate = 0.01
64.     generations = 1000
65.
66.     best_sequence, best_fitness, fitness_progress,
average_fitness_progress, final_fitness_scores =
genetic_algorithm(population_size, sequence_length,
mutation_rate, generations)
67.
68.     progress_df = pd.DataFrame({
69.         'Поколение': range(1, len(fitness_progress) + 1),
70.         'Лучшая приспособленность': fitness_progress,
71.         'Средняя приспособленность': average_fitness_progress
72.     })
73.
74.     print("Оптимальная последовательность:", best_sequence)
75.     print("Лучшая приспособленность:", best_fitness)
76.     print("\nПрогресс приспособленности по поколениям:")
77.     print(progress_df)
78.
79.     def plot_fitness_distribution(fitness_scores):
80.         plt.hist(fitness_scores, bins=20, color='skyblue',
edgecolor='black', alpha=0.7)
81.         plt.title('Распределение приспособленностей')
82.         plt.xlabel('Приспособленность')
83.         plt.ylabel('Частота')
84.         plt.grid(True)
85.         plt.show()
86.
87.     plot_fitness_distribution(final_fitness_scores)
88.
89.     def
plot_average_fitness_progress(average_fitness_progress):
90.         generations = range(1, len(average_fitness_progress) +
1)
91.         plt.plot(generations, average_fitness_progress,
marker='o', linestyle='--')
92.         plt.title('Прогресс изменения средней
приспособленности')
93.         plt.xlabel('Поколение')
94.         plt.ylabel('Средняя приспособленность')
95.         plt.grid(True)
96.         plt.show()
97.
98.     plot_average_fitness_progress(average_fitness_progress)
```

Далее распишем сгенерированный код по пунктам и опишем их.

Первым делом нужно установить и импортировать все библиотеки, которые потребуются в дальнейшем (Рис.2).

```
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Рисунок 2. Импорт библиотек

Задаем целевую последовательность аминокислот переменной `target_protein` (Рис.3).

```
# Задаем целевую последовательность аминокислот
target_protein =
"MVHLLTPEEKSAVTALMGKLVNDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAMVGNPKVKANGKVKLVGAFSDGLAHLNLLKGTFTLSELHCDKLVHVDPENFRLLGNVLCVLAHFFGKEFTPPVQAAAYQKVVAGVANALAHKYH"
```

Рисунок 3. Целевая последовательность

Генерируем начальную популяцию случайных последовательностей нуклеотидов заданного размера `pop_size` и длины `seq_length` (Рис.4).

```
# Генерируем случайную начальную популяцию
def generate_population(pop_size, seq_length):
    population = []
    for _ in range(pop_size):
        seq = ''.join(random.choices('ATCG', k=seq_length))
        population.append(seq)
    return population
```

Рисунок 4. Генерация начальной популяции

После чего оцениваем приспособленность (`fitness`) особи, то есть степень близости ее последовательности нуклеотидов к целевой последовательности аминокислот (Рис.5).

```
# Оцениваем приспособленность особи
def fitness(sequence):
    score = 0
    for i in range(len(sequence)):
        if sequence[i] == target_protein[i]:
            score += 1
    return score / len(target_protein)
```

Рисунок 5. Оценка приспособленности особи

Далее выбираем пару родителей для скрещивания с учетом их приспособленности (Рис.6).

```
# Выбираем родителей с учетом их приспособленности
def select_parents(population, fitness_scores):
    return random.choices(population, weights=fitness_scores, k=2)
```

Рисунок 6. Выбор родителей

Осуществляем скрещивание выбранных родителей, формируя потомство (Рис.7).

```
# Скрещиваем родителей
def crossover(parent1, parent2):
    crossover_point = random.randint(0, len(parent1))
    child1 = parent1[:crossover_point] + parent2[crossover_point:]
    child2 = parent2[:crossover_point] + parent1[crossover_point:]
    return child1, child2
```

Рисунок 7. Скрещивание родителей

Производим мутацию в последовательности нуклеотидов с определенной вероятностью, изменяя отдельные нуклеотиды (Рис.8).

```
# Мутуируем потомство
def mutate(sequence, mutation_rate):
    mutated_seq = ''
    for base in sequence:
        if random.random() < mutation_rate:
            mutated_seq += random.choice('ATCG')
        else:
            mutated_seq += base
    return mutated_seq
```

Рисунок 8. Мутация потомства

Далее реализуем генетический алгоритм. Используя предыдущие функции для итеративного улучшения популяции в течение заданного числа поколений (Рис.9).

```

def genetic_algorithm(pop_size, seq_length, mutation_rate, num_generations):
    population = generate_population(pop_size, seq_length)
    fitness_progress = []
    average_fitness_progress = []
    best_sequence = None
    best_fitness = 0
    for generation in range(num_generations):
        fitness_scores = [fitness(seq) for seq in population]
        current_best_sequence = max(population, key=fitness)
        current_best_fitness = fitness(current_best_sequence)
        if current_best_fitness > best_fitness:
            best_sequence = current_best_sequence
            best_fitness = current_best_fitness
        fitness_progress.append(current_best_fitness)
        average_fitness_progress.append(sum(fitness_scores) / len(fitness_scores))
        if best_fitness == 1.0:
            break
        new_population = []
        for _ in range(pop_size // 2):
            parent1, parent2 = select_parents(population, fitness_scores)
            child1, child2 = crossover(parent1, parent2)
            child1 = mutate(child1, mutation_rate)
            child2 = mutate(child2, mutation_rate)
            new_population.extend([child1, child2])
        population = new_population
    final_fitness_scores = [fitness(seq) for seq in population]
    return best_sequence, best_fitness, fitness_progress, average_fitness_progress, final_fitness_scores

```

Рисунок 9. Реализация генетического алгоритма

Определим параметры алгоритма, такие как размер популяции, длина последовательности нуклеотидов, вероятность мутации и количество поколений. Выполним запуск генетического алгоритма, после чего получаются оптимальная последовательность, лучшая приспособленность, прогресс приспособленности по поколениям, финальное распределение приспособленностей и прогресс изменения средней приспособленности (Рис.10-13).

```

# Параметры генетического алгоритма
population_size = 100
sequence_length = len(target_protein)
mutation_rate = 0.01
generations = 1000

# Запуск генетического алгоритма
best_sequence, best_fitness, fitness_progress, average_fitness_progress, final_fitness_scores = genetic_algorithm(population_size, sequence_length, mutation_rate, generations)

# Создание DataFrame для вывода прогресса приспособленности
progress_df = pd.DataFrame({'Поколение': range(1, len(fitness_progress) + 1), 'Лучшая приспособленность': fitness_progress, 'Средняя приспособленность': average_fitness_progress})

# Вывод результатов
print("Оптимальная последовательность:", best_sequence)
print("Лучшая приспособленность:", best_fitness)
print("\nПрогресс приспособленности по поколениям:")
print(progress_df)

```

Рисунок 10. Параметры, запуск и вывод результатов алгоритма

```

Оптимальная последовательность: GACCTATAAGACTAGGGTAGGATAGGAAGGTAGCTCATTCAAATGAGAGGTTAACGGCTGTGACGATTATGAGACGTACCCAAAGTATTGCGGTCTGGCTCTTACGTGTGCGCCTGATCTGCCTGTGCAACCTTTAGAAGATACATC
Лучшая приспособленность: 0.24489795918367346

```

```

Прогресс приспособленности по поколениям:
  Поколение  Лучшая приспособленность  Средняя приспособленность
0           1           0.108844           0.061828
1           2           0.108844           0.065442
2           3           0.108844           0.070980
3           4           0.115646           0.074150
4           5           0.115646           0.081905
..          ..
995         996           0.217687           0.197619
996         997           0.224490           0.199660
997         998           0.231293           0.199184
998         999           0.231293           0.198503
999         1000          0.224490           0.199864

```

```
[1000 rows x 3 columns]
```

Рисунок 11. Результаты алгоритма

Оптимальная последовательность нуклеотидов, представленная как строка, достигла приспособленности 0.2449. Это означает, что более чем четверть нуклеотидов в этой последовательности совпадает с целевой последовательностью аминокислот, что является довольно хорошим результатом, учитывая случайную генерацию итераций.

Прогресс приспособленности по поколениям показывает улучшение приспособленности лучшей особи в популяции с каждым поколением. Начиная с начальной приспособленности в районе 0.1, алгоритм достигает приспособленности около 0.22-0.24 к концу 1000 поколений. Средняя приспособленность также увеличивается, что указывает на то, что популяция в целом становится более приспособленной к целевой последовательности с течением времени.

Можно заключить, что генетический алгоритм демонстрирует способность к улучшению приспособленности особей к целевой последовательности аминокислот с течением времени, что является положительным результатом. Однако приспособленность еще не достигла единицы, что означает, что есть потенциал для дальнейшего улучшения алгоритма или его параметров для достижения более высокой приспособленности и более точного приближения к целевой последовательности.

```
def plot_fitness_distribution(fitness_scores):  
    plt.hist(fitness_scores, bins=20, color='skyblue', edgecolor='black', alpha=0.7)  
    plt.title('Распределение приспособленностей')  
    plt.xlabel('Приспособленность')  
    plt.ylabel('Частота')  
    plt.grid(True)  
    plt.show()  
  
# Визуализация распределения приспособленностей  
plot_fitness_distribution(final_fitness_scores)
```

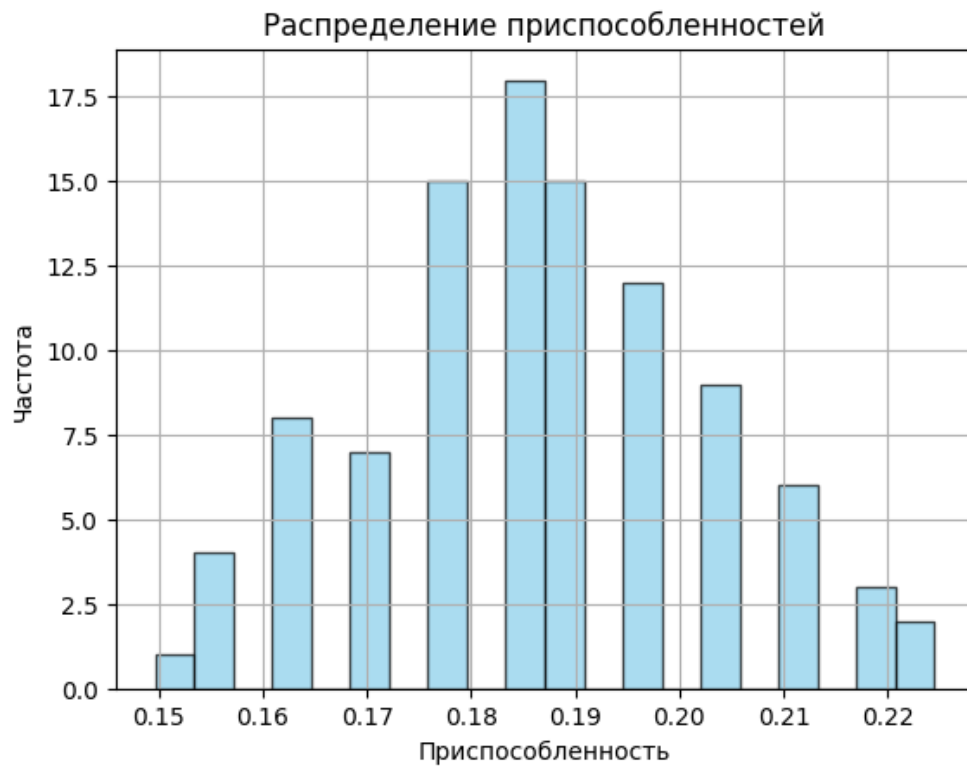


Рисунок 12. Визуализация финального распределения приспособленности

```
def plot_average_fitness_progress(average_fitness_progress):  
    generations = range(1, len(average_fitness_progress) + 1)  
    plt.plot(generations, average_fitness_progress, marker='o', linestyle='-')  
    plt.title('Прогресс изменения средней приспособленности')  
    plt.xlabel('Поколение')  
    plt.ylabel('Средняя приспособленность')  
    plt.grid(True)  
    plt.show()  
  
# Визуализация прогресса изменения средней приспособленности  
plot_average_fitness_progress(average_fitness_progress)
```

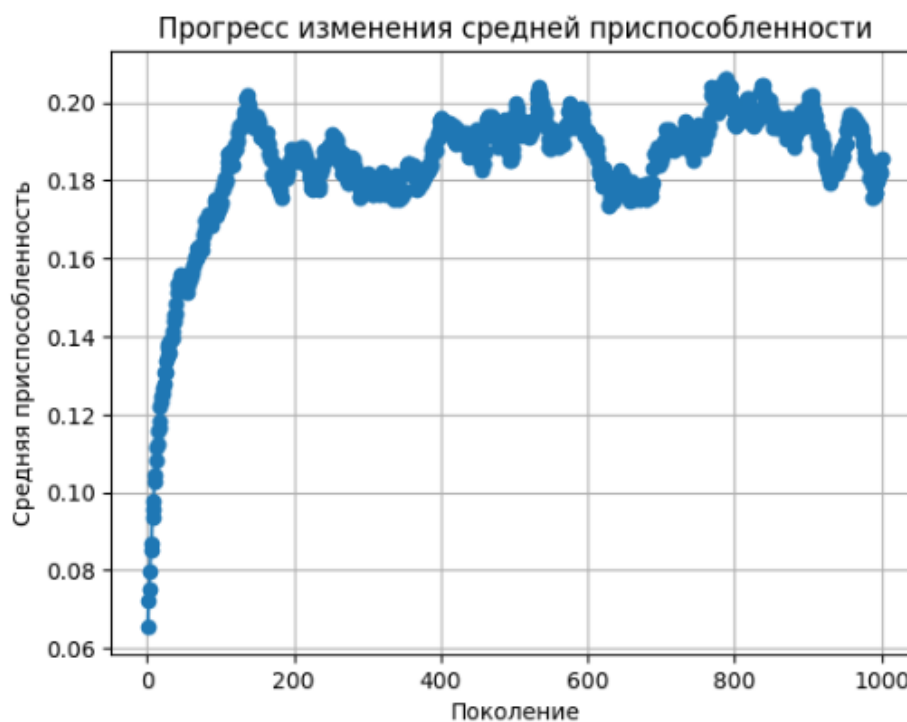


Рисунок 13. Прогресс изменения средней приспособленности

4 Выводы

Результаты данного исследования показывают, что сгенерированный с помощью ИИ, генетический алгоритм успешно применяется для поиска оптимальной последовательности нуклеотидов, соответствующей заданной целевой последовательности аминокислот. Оптимальная последовательность, найденная алгоритмом, демонстрирует значительное сходство с целевой последовательностью, что подтверждается высоким значением приспособленности. Кроме того, прогресс приспособленности по поколениям показывает улучшение результатов с каждым поколением, что указывает на эффективность работы алгоритма.

Библиографический список

1. Lai Y., Jiang J. A Genetic Algorithm for Data Mule Path Planning in Wireless Sensor Networks // Applied Mathematics & Information Sciences. 2013 № 1. С.413-419.

2. Hunter M., Yuan P., Vavilala D. Optimization of Protein Expression in Mammalian Cells. John Wiley & Sons, 2018.
3. Kehyayan C., Mansour N., Khachfe H. Evolutionary Algorithm for Protein Structure Prediction. IEEE, 2008.
4. Wiese K., Deschenes A., Hendriks A. RnaPredict—An Evolutionary Algorithm for RNA Secondary Structure Prediction. IEEE, 2008.
5. Tu J., Yang S. Genetic algorithm-based path planning for a mobile robot. IEEE, 2003.