Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 125 (2018) 68–76

Procedia Computer Science

www.elsevier.com/locate/procedia

6th International Conference on Smart Computing and Communications, ICSCC 2017, 7-8 December 2017, Kurukshetra, India

A Genetic Algorithm Approach to Autonomous Smart Vehicle Parking system

Diya Thomas*[a], Binsu C. Kovoor[b]

[a]Department of Computer Science and Engineering, RSET, Kochi, India
[b]Department of Information Technology, CUSAT, Kochi, India

Abstract

in parking bay of shopping mall is a usual scenario witnessed at those peak times. Customer precious time and fuel is wasted and they get only few time for shopping. Authorities find it difficult to cope up with this situation even after appointing more employees to manage the traffic experienced in the bay. A smart car parking system that could elevate this problem is an urgent requirement for the shopping mall. This paper falls light on this issue by proposing a new prototype for the smart vehicle parking system. A genetic algorithm approach has been taken to address the issue of scheduling the vehicle to the parking bay.

Chromosome structure

The problem is to find out the best parking region.Each parking region is represented as a chromosome. In the chro-mosome structure as shown in Fig. 2.,each bit in the chromosome represent slot in the parking region. If the slot is occupied the value of the slot is 0 and if it not occupied the value is 1. Each bit is represented by C_i where 1 _ *i* _ *N* gives information about slot occupancy.

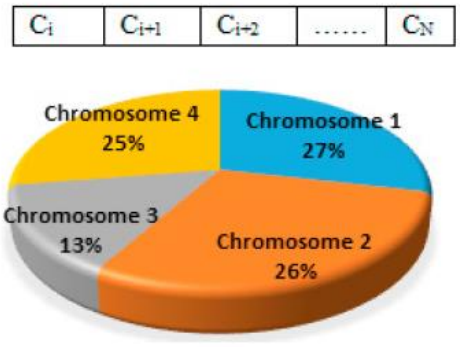

Fig. 2. Chromosome Structure and Roulette wheel selection

```
def roulette_wheel(wheel):
    m = sum(wheel.values())
    pick = random.uniform(0, m)
    current = 0
    for key, value in wheel.items():
        current += value
        if current > pick:
            return key
```

Initial Population

Generation The initial population size is set to 4 which is equal to number of parking region. If a parking region consist of 8 slots then the number of bits needed to represent the chromosome is 8. In general of there are ´M´ parking region with ´N´ number of parking slot in each region then our population consist of ´M´ chromosome with chromosome length equal to ´N´.The encoded chromosome structure is shown in Table 2.

Table 2. Encoding-Initial Population

Chromosome Number	chromosome
1	00011111
2	01111111
3	00000011
4	00001111

```
def initial(population_size, num_of_slots):
    chromosomes = []
    for i in range(population_size):
        chromosomes.append([random.randint(0, 1) for j in range(num_of_slots)])

    return chromosomes
```

Selection

The selection operation select the best solution from the population of solution based on the fitness function *F* (*s*). The *F* (*s*) of solution ´s´ is defined in Equation (12).Roulette wheel selection strategy is employed.

$$F(s) = \frac{\sum \alpha_i C_i}{N}$$

where $\alpha_i = 2^{i+1}$ and $1 \leq i \leq N$

Alpha is the weight assigned to each bit in the solution or chromosome As an example, consider the chromosome 3 (00000001) listed in table 3. The fitness F(chromosome 3), probability of selection P(chromosome 3), Percentage probability PP(chromosome 3) and expected count E(chromosome 3) are calculated as follows:

$F(00000001)=2^0 0 + 2^1 0 + 2^2 0 + 2^3 0 + 2^4 0 + 2^5 0 + 2^6 0 + 2^7 1=\frac{128}{8}=16$

Probability of selection P(s) is as follows:

$$P(s) = \frac{F(s)_i}{\sum F(s)_i} \tag{13}$$

where $1 \leq i \leq N$

Example:$P(00000001) = \frac{16}{117.75} = 0.13$

Percentage probability PP(s) is as follows:

$$PP(s) = P(s)(100) \tag{14}$$

Example:$PP(00000001) = (0.13)100=13\%$

Expected Count E(s) is defined below:

$$E(s) = \frac{F(s)_i}{Average\,(F(s)_i)} \tag{15}$$

$$Average\,(F(s)) = \frac{\sum F(s)_i}{N} \tag{16}$$

where $1 \leq i \leq N$

Example:$E(00000001)= \frac{16}{14.71}=1.08$

Similarly fitness, probability of selection, percentage probability and expected count of the remaining chromosomes are calculated. The obtained results for each chromosome are listed in table 3. The roulette wheel constructed after above PP(s) calculation of all the chromosome is shown in Fig.2. After the roulette wheel selection, the actual count of the chromosomes is also obtained and is listed as the last column in table 3. Actual count of the chromosome will determine the chromosome from the population that will be moved to the mating pool. The actual count of the chromosome ´s´ A(s) equal to ´i´ means that the chromosome ´s´ will appear in the mating pool ´i´ times. For example A(chromosome 2) is equal to 2 as shown in table 3. So chromosome 2 (01111111) will appear in the mating pool twice. The mating pool constructed based on the actual count of the chromosome is shown in the table 4. Based on the actual count A(s), chromosome 2 is chosen twice and chromosome 1 and 4 is chosen once to be in the mating pool as shown in table 4. The crossover and mutation operation is applied to these chromosomes in the mating pool to generate offspring. Since actual count of the chromosome 3 (least fit) is zero, it is not taken to the mating pool and will be completely removed from the current population.

```
def selection(chromosomes, num_of_slots):
    # print(chromosomes)
    fitness = []
    P = []
    PP = []
    E = []
    A = []
    for i in chromosomes:
        counter = 0
        f_tmp = 0.0
        for j in i:
            f_tmp += j*(2**counter)
            counter += 1
        fitness.append(f_tmp/num_of_slots)

    for i in fitness:
        P.append(i/sum(fitness))
        PP.append(i/sum(fitness)*100)
        E.append(i/statistics.mean(fitness))
    return fitness, P, PP, E
```

Crossover

The cross over operation used to solve the problem is three parent crossover. Bit-wise comparison of the first two parents are done. The bit in each location of offspring is chosen based on the following condition. If the bit in the same location of the first two parent chromosomes are same then that bit will be chosen for offspring. But if the bit differ then the bit specified at the same location in third parent chromosome will be chosen. The crossover probability is 100%. For Example, consider the chromosome 2 (01111111) in the mating pool as shown in second column of the table 4. As in the third column of the table 4, the parents randomly chosen for crossover of chromosome 2 is chromosome 2, chromosome 3 and chromosome 4. Cross over operation applied on chromosome 2 is shown below. Similarly crossover operation is applied to remaining chromosome in the mating pool. The offspring's obtained after crossover is listed in the fourth column of the table 4. Fitness value of each offspring is calculated and is listed as fifth column in table 4. The fitness function is calculated using the same procedure described under selection operation.
Example:

Before Crossover

Parent 1(Chromosome2)- 01111111

Parent 2(chromosome 3)- 00011111

Parent 3(chromosome 4)- 00001111

After crossover Offspring-00011111

```
def crossover(chromosomes_dict, wheel, population_size, num_of_slots):
    chromosomes_dict_tmp = chromosomes_dict.copy()
    offspring = chromosomes_dict.copy()
    for i in range(population_size):
        chromosomes_dict[i] = chromosomes_dict_tmp[roulette_wheel(
            wheel)].copy()
    print(chromosomes_dict)
    for i in range(population_size):
        first_parent, second_parent, third_parent = random.sample(
            range(0, population_size), 3)
        # print(i)
        # print(first_parent, second_parent, third_parent)
        for j in range(num_of_slots):
            if (chromosomes_dict[first_parent][j] ==
chromosomes_dict[second_parent][j]):
                offspring[i][j] = chromosomes_dict[first_parent][j]
            else:
                offspring[i][j] = chromosomes_dict[third_parent][j]
    # print(offspring)
    return offspring
```

Mutation

Mutation is done on the offspring listed in column four of table 4. The bits at the random position are flipped. For example, consider the offspring 1 (01111111) listed in second column of table 5. The random position chosen to flip the bit in offspring 1 is the first position. Mutation probability is 0.12. The bit in first position of offspring 1 was originally 0 and is flipped to 1 after mutation as shown below. Offspring 1 after crossover: 01111111 Offspring 1

after mutation: 11111111 Remaining offspring's generated after crossover are all mutated like the offspring 1. The mutated offspring's are shown in column 4 of table 5. After crossover and mutation, there is an increase in the total fitness value of the offspring compared to their parents. The fitness value of the mutated offspring are calculated and is shown as the last column in table 5. These fitness value of the individual chromosome will decide their existence to the next generation of population. The next generation of population obtained is shown in table 6. To obtain an optimal solution, all the stages discussed above is repeated until a stopping condition is met. Algorithm Step 1: Encode the solution using binary encoding

```
def mutation(offspring, population_size, num_of_slots, rate):
    offspring_tmp = offspring.copy()
    for i in range(population_size):
        for j in range(num_of_slots):
            if random.random() < rate:
                offspring_tmp[i][j] = 1-offspring_tmp[i][j]
    return offspring_tmp
```

Table 3. Selection

Chromosome Number	chromosome	Fitness [F(s)$_i$]	P(s)	PP(s)	E(s)	A(s)
1	00011111	31	0.26	26%	2.10	1
2	01111111	31.75	0.27	27%	2.22	2
3	00000011	16	0.13	13%	1.08	0
4	00001111	30	0.25	25%	2.03	1
Sum [$\sum F(s)_i$]		117.75	0.91	100%	7.43	4
Average [$\frac{(\sum F(s)_i)}{N}$]		14.71	0.22	25%	1.85	1
Maximum		32.75	0.27	27%	2.22	2

Table 4. Crossover

Chromosome number	Mating pool	Chromosome number of the chosen parent	Offspring	F(s)
1	01111111	1,2,3	01111111	31.7
2	01111111	2,3,4	00011111	31
3	00011111	1,2,4	01111111	31.7
4	00001111	1,4,3	00011111	31
Sum [$\sum F(s)_i$]				125.4

Table 5. Mutation

Chromosome Number	Offspring after crossover	Random Position	Offspring after mutation	F(s)
1	01111111	1	11111111	31.7
2	00011111	3	00111111	31.5
3	01111111	2	00111111	31.5
4	00011111	3	00111111	31.5
Sum [$\sum F(s)_i$]				126.3