King Saud University

College of Computer and Information Sciences

Information Technology Department

IT 426

Artificial Intelligence

SmartRehabilitation

a GA-Based Solution for Optimal Exercise Plans

Prepared by

Group member: group# 9
Lama Alamri, 439201036, Leader
Ghaida Alomran, 439200697, Member
Ghada Altamimi, 439200969, Member
Sarah Aldrees, 439200791, Member

October, 2021

Table of Contents

Solution Representation 3
Fitness Function 4

Solution Representation

We use the Python language to solve the rehabilitation problem by using genetic algorithms (GA). We transfer the given table in the PDF file to a Comma-Separated Values (CSV) file, then we load the CSV file into the Python program to get the search space.

```
Body Part,Exercise,Condition Type,Age Category
Elbow,Elbow extensor using free weights,Stroke,Adult
Elbow,Crawling,Stroke,Child
Elbow,Elbow flexor using free weights,Stroke,Adult
Elbow,Bear walking,Stroke,Child
Elbow,Elbow extensor using theraband,Spinal cord injuries,Adult
Elbow,Lifting in parallel bars,Spinal cord injuries,Child
Elbow,Elbow Flexor using theraband,Spinal cord injuries,Adult
Elbow,Lifting in sitting using scale,Spinal cord injuries,Child
Elbow,Rotating forearm,Brain injury,Adult
Elbow,Forearm supination,Brain injury,Child
Elbow,Learning forwards in a large ball,Brain injury,Adult
Elbow,Wheelbarrow walking on hands,Brain injury,Child
Upper Arm,Shoulder external rotator using free weights,Stroke,Adult
Upper Arm,Weight-bearing through one shoulder,Stroke,Child
Upper Arm,Shoulder extensor,Stroke,Adult
Upper Arm,Crawling,Stroke,Child
Upper Arm,Shoulder abductor using theraband,Spinal cord injuries,Adult
Upper Arm,Shoulder abductor Stritch in setting,Spinal cord injuries,Child
Upper Arm,Shoulder adductor using theraband,Spinal cord injuries,Adult
Upper Arm,Boxing in setting,Spinal cord injuries,Child
Upper Arm,Push-ups in prone,Brain injury,Adult
Upper Arm,Reaching in four points kneeling,Brain injury,Child
Upper Arm,Lowering and pushing-up in long setting,Brain injury,Adult
Upper Arm,Crab-walking,Brain injury,Child
Knee/Lower leg,Knee flexor using a device,Stroke,Adult
Knee/Lower leg,Squatting against a wall,Stroke,Child
Knee/Lower leg,Knee extensor using a device,Stroke,Adult
Knee/Lower leg,Seated walking,Stroke,Child
```

After that, the user will specify their optimal plan by entering age category (options include Adult and Child), condition type (options include Stroke, Spinal cord, and Brain injuries) and number of exercises (number of different exercises to be performed per body part). Then, we will represent (encode) each individual as an array of random indexes, where each index points to an exercise from the search space (the table of all exercises). For example, they may enter **1** elbow exercise, **2** upper arm exercises, **1** knee/lower leg exercise, and **1** wrist exercise. That is a total of **5** exercises .Therefore, the array will have **5** indexes of exercises. Here is an example of random encoded indexes:

[7, 14, 0, 11, 9]

So the above array of indexes will point to these exercises:

7	Elbow	Lifting in sitting using scale	Spinal cord injuries	Child
14	Upper Arm	Shoulder extensor	Stroke	Adult
0	Elbow	Elbow extensor using free weights	Stroke	Adult
11	Elbow	Wheelbarrow walking on hands	Brain injury	Child
9	Elbow	Forearm supination	Brain injury	Child

That array of indexes represents one possible solution/individua

Fitness Function

The fitness function uses the *Age Category*, the *Condition Type*, and the *Number of Exercises*, while the *Age Category* and *Number of Exercises* are equally important which means the *Age Category is 25%* and *Number of Exercises is 25%,* they are half as important as 3- the *Condition Type which has a 50%* so the they are all 100%.

$$w1 = 0.25 \text{ , } w2 = 0.25 \text{ , } w3 = 0.5$$
$$\text{where } \Sigma\, wi = 1.$$

First, it calculates each of these as weighted sums and then creates a fitness from these weighted sums between 0.0 and 1.0.

For the *Age Category* and *Condition Type*, the weighted sum is computed by adding one when they match the optimal plan, and not adding anything (zero) if they do not match. For example, if the user inputted *Adult* and *Stroke* for the optimal plan, then this is the calculation:

```
age_category_sum = 0
condition_type_sum = 0
optimal_plan.age_catogery >>Adult
optimal_plan.condition_type >>Stroke

    if exercise.age_category == optimal_plan.age_catogery:
        age_category_sum += 1

    if exercise.condition_type == optimal_plan.condition_type:
        condition_type_sum += 1
```

Now, these values need to be normalized between 0.0 and 1.0, so they are divided by the number of optimal exercises:

```
number_of_exercises = 5
age_category_sum = age_category_sum / number_of_exercises
condition_type_sum = condition_type_sum / number_of_exercises
```

For the *Number of Exercises*, the weighted sum is computed by subtracting the optimal number of each exercise by what the individual has, and then it takes the absolute value of that for example

```
optimal_num_of_elbow =2
optimal_num_of_upper_arm=1
optimal_num_of_knee_lower_leg=1
optimal_num_of_wrist=0
num_of_elbow =1
num_of_upper_arm=0
num_of_knee_lower_leg =2
num_of_wrist =1
```

```
#n (the difference between individual and optimal exercise)

n= (

  abs(optimal_num_of_elbow - num_of_elbow) +
  abs(optimal_num_of_upper_arm - num_of_upper_arm) +
  abs(optimal_num_of_knee_lower_leg - num_of_knee_lower_leg) +
  abs(optimal_num_of_wrist - num_of_wrist))
```

optimal plan has **2** elbow exercises and the individual has **0**, then that's a difference of **2** meaning that the individual is off by **2 so the sum of differences is**

```
n=2+1+1+1=5
```

The problem with this sum is that lower values are good and higher values are bad. To fix this, the sum is subtracted from the maximum sum of possible value so, we need first to compute the maximum sum of possible value by multiply the number of body parts which is 4 by number of exercises, then we subtracted the n (the difference between individual and optimal exercise) from maximum sum of possible value as shown in below :

max_sum_of_possible_value = 4 * number_of_exercises
no_of_exercises_sum = max_sum_of_possible_value - n (the difference between individual and optimal exercises.

Now, this sum also needs to be between 0.0 and 1.0, so like the other sums, this number is divided by the maximum value:

```
no_of_exercises_sum = no_of_exercises_sum / max_sum
```

Finally, the requirements state that the *Age Category* and *Number of Exercises* should be equally important, but half as important as the *Condition Type*.
which means
Therefore, they are multiplied by these percentages to compute the final fitness:

```
fitness = 0.0
fitness += 0.25 * age_category_sum          # 25%
fitness += 0.25 * no_of_exercises_sum       # 25%
fitness += 0.50 * condition_type_sum        # 50%
```

The final fitness is between 0.0 and 1.0, when it is close to 0.0 it means a worse fitness and when it is close to 1.0 it means a better. A fitness of 1.0 is the ideal, perfect goal and what the Genetic Algorithm is trying to achieve.