WARSAW UNIVERSITY OF TECHNOLOGY
Faculty of Mathematics
and Information Science

Artificial Intelligence Fundamentals Project:

Evolutionary algorithm for solving the Knapsack Problem

Aleksander Kuśmierczyk
Mikołaj Słoń
Mateusz Szperna
Kornel Żaba

Computer Science
22 March 2017

Date	**Author of change**	**Description**
22.03.2017	Mikołaj Słoń	Created documentation file
22.03.2017	Kornel Żaba	Added problem description
23.03.2017	Mateusz Szperna	Added existing solutions
29.03.2017	Aleksander Kuśmierczyk	Added detailed solution

Table of Contents

1. Description of a problem

The aim of our project is to provide a solution to so called ‘**Knapsack Problem’**. If we have a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible (Figure 1.).

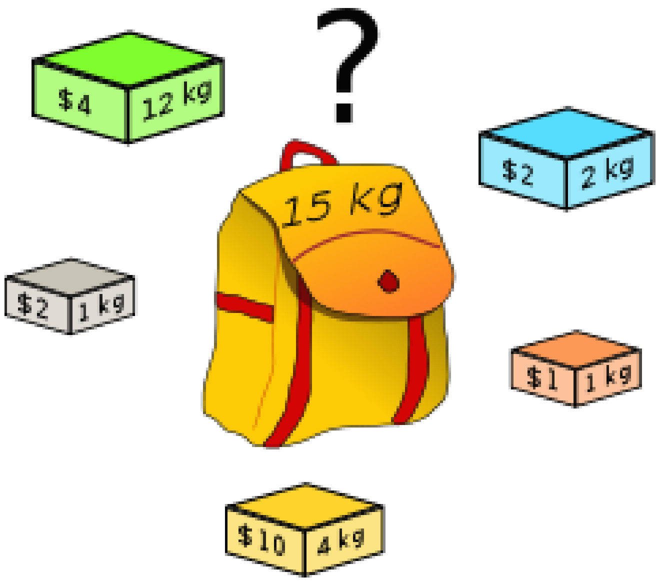


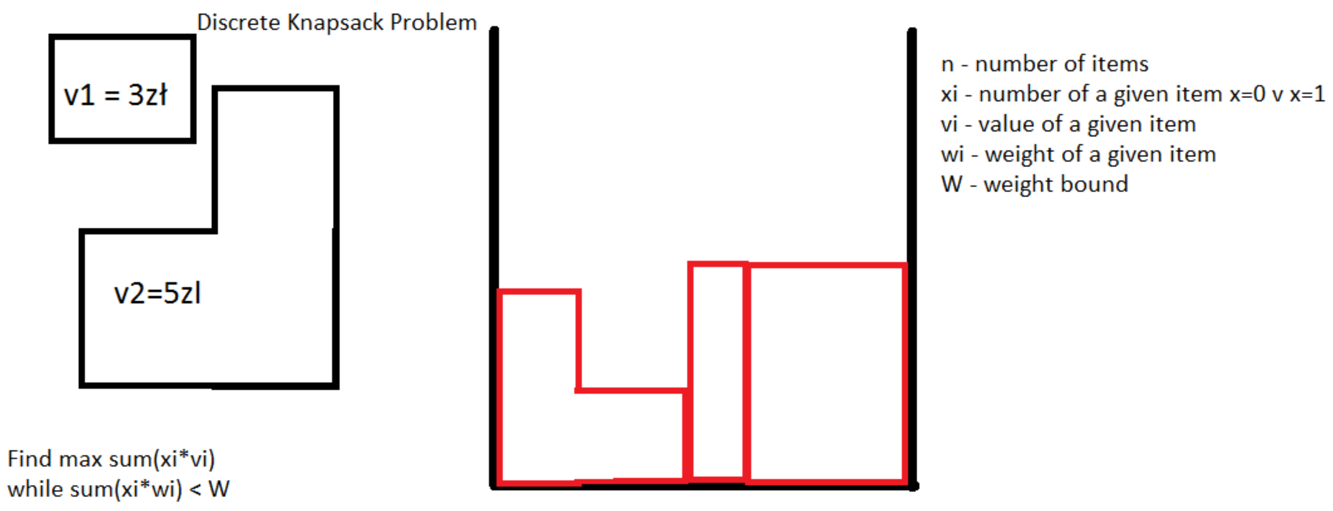
Figure 1. Visualization od Knapsack Problem

2. An analysis of a problem

Our project will discuss the **0-1 Knapsack Problem**. In which the number x_i of copies of each kind of item is restricted to zero or one. Given a set of n items numbered from 1 up to n, each with a weight w_i and a value v_i, along with a maximum weight capacity W.

Our goal will be:

Maximize $\Sigma \boldsymbol{v_i x_i}$ while $\Sigma w_i x_i \leq W$ and $x_i \in \{0, 1\}$


Discrete Knapsack Problem
v1 = 3zł
v2=5zl
n - number of items
xi - number of a given item x=0 v x=1
vi - value of a given item
wi - weight of a given item
W - weight bound
Find max sum(xi*vi)
while sum(xi*wi) < W

3. Some existing solutions

Naive solution

```
int knapSack(int W, int wt[], int val[], int n)
{
   int i, w;
   int K[n+1][W+1];

   // Build table K[][] in bottom up manner
   for (i = 0; i <= n; i++)
   {
       for (w = 0; w <= W; w++)
       {
           if (i==0 || w==0)
               K[i][w] = 0;
           else if (wt[i-1] <= w)
                 K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
           else
                 K[i][w] = K[i-1][w];
       }
   }

   return K[n][W];
}
```

Improved solution using Dynamic programming

```
// Returns the maximum value that can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{
   // Base Case
   if (n == 0 || W == 0)
       return 0;

   // If weight of the nth item is more than Knapsack capacity W, then
   // this item cannot be included in the optimal solution
   if (wt[n-1] > W)
       return knapSack(W, wt, val, n-1);

   // Return the maximum of two cases:
   // (1) nth item included
   // (2) not included
   else return max( val[n-1] + knapSack(W-wt[n-1], wt, val, n-1),
                    knapSack(W, wt, val, n-1)
                  );
}
```

All existing solutions presented here are taken from the article **“Dynamic Programming (0-1 Knapsack Problem)”** on the website www.geeksforgeeks.org.
“

An idea for solving it using evolutionary algorithm.

- We create a set of random items with sum of weights less than W. We will call this set teams.
- We define a team as good if the sum of values of its elements is high.
- Each of the teams should be able to mutate. Mutation is used to randomly shake up some teams. This prevents us from getting stuck at a local maximum.
- We should be able to breed. A function that takes two teams and creates a better one.
- At each iteration of the algorithm Weakest teams will be discarded and strongest ones bread
- At each iteration we randomly add other individuals to promote genetic diversity.

This solution gets better results the more iterations we make.

4. Description of preferred solution

We have chosen **C#** programming language for development of our project, as the whole group has already had experience with it, additionally it provides useful functionalities. As our project is focused on solving the problem rather than visualizing it, our presentation layer will not include additional frameworks. Main IDE for the project is set to be **Visual Studio 2015**. It is a very powerful C# development environment, with integrated **Visual Studio Unit Testing Framework**, that will be used for testing phase of the project.

As we chose the Evolutionary approach, the solutions of the initial problem will take the role of the candidate solutions and subsequent computations using the fitness function and with the application of some random changes will gradually evolve the population reaching the optimal solution.

Our problem is to be data driven therefore we will use the pre prepared player datasets available at (for testing).Testing and running the program shall be performed using this sample, as it provide all necessary data for our problem. Additionally we will use a random knapsack problem generator (self implemented) for the sake of testing. The results will be compared with the expected ones.

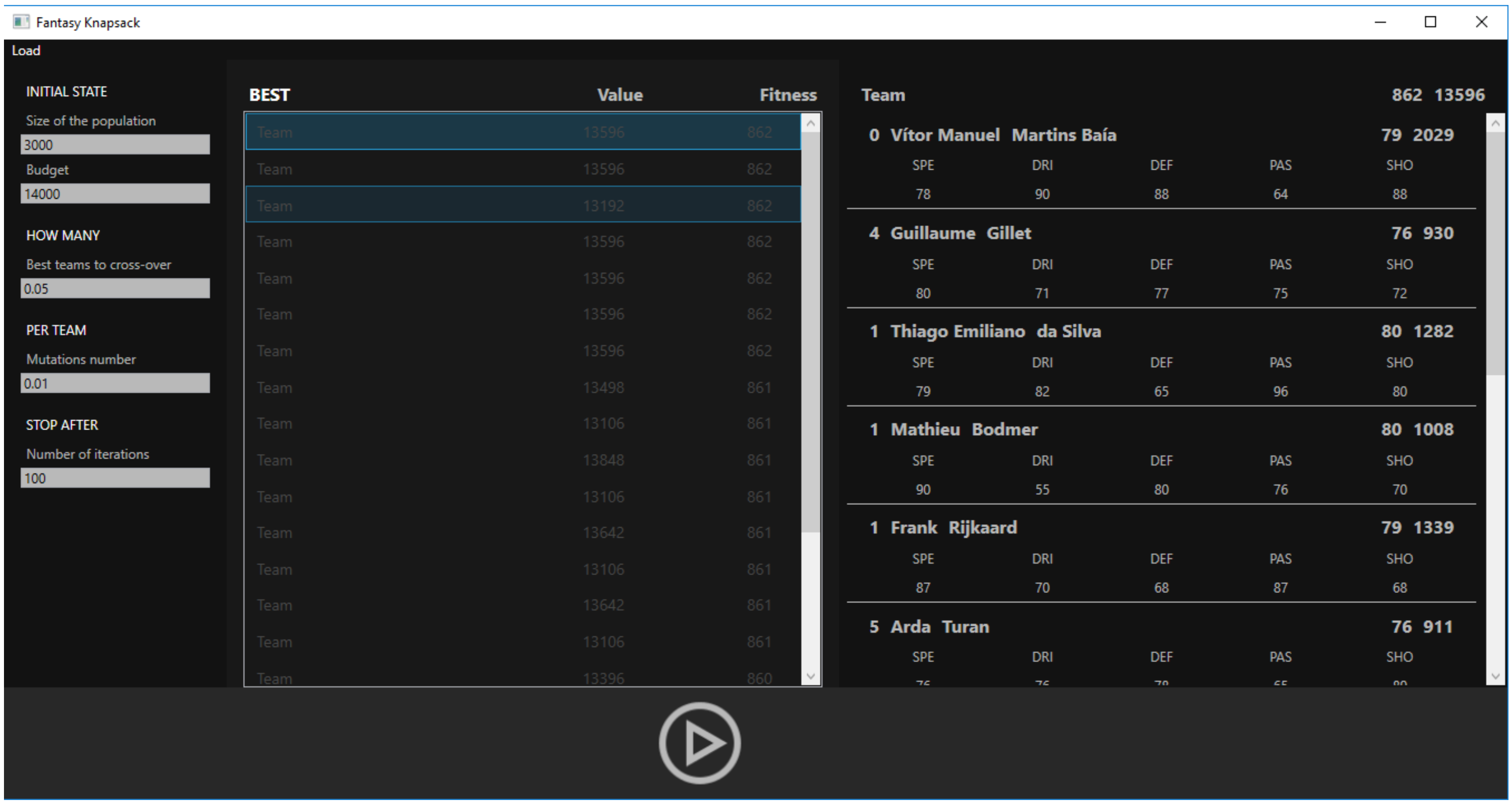
Project will be structured into 3 distinctive parts:

1. Back-end application:
    - back-end functionalities.
2. Front-end application:
    - User friendly interface.
    - Adjustable settings.
3. Evolutionary algorithm for Knapsack problem.
    - Problem is adjusted to create the best soccer team within the budget constraints.

1) All of the backend functionalities are accessible through the code manipulation, as most of them are already strapped into the user interface operating in the program. Some of them are: Use of the random sample, change the number of players, modification of weighting algorithm.

2)User friendly interface provides the simple and intuitive panels and controls in order to adjust the problem to the needs. Player database is imported from the .csv file in the directory and transformed for our needs in order to support the running of the program. The number of iterations as well as the population of teams is also possible to adjust through the user interface.

3) The approach we took in order to solve this problem was the evolutionary approach, therefore we allow to set both the mutation and crossover rate between the teams created randomly. When finished the program will return the “winning” team meaning the team with the best average score of the players on their respective roles (all weighted differently) and with the cumulative price (of players) within the budget constraints. All of the players are split



Fantasy Knapsack

Load

INITIAL STATE

Size of the population: 3000

Budget: 14000

HOW MANY

Best teams to cross-over: 0.05

PER TEAM

Mutations number: 0.01

STOP AFTER

Number of iterations: 100

BEST	Value	Fitness
Team	13596	862
Team	13596	862
Team	13192	862
Team	13596	862
Team	13596	862
Team	13596	862
Team	13596	862
Team	13498	861
Team	13106	861
Team	13848	861
Team	13106	861
Team	13642	861
Team	13106	861
Team	13642	861
Team	13106	861
Team	13396	860

Team 862 13596

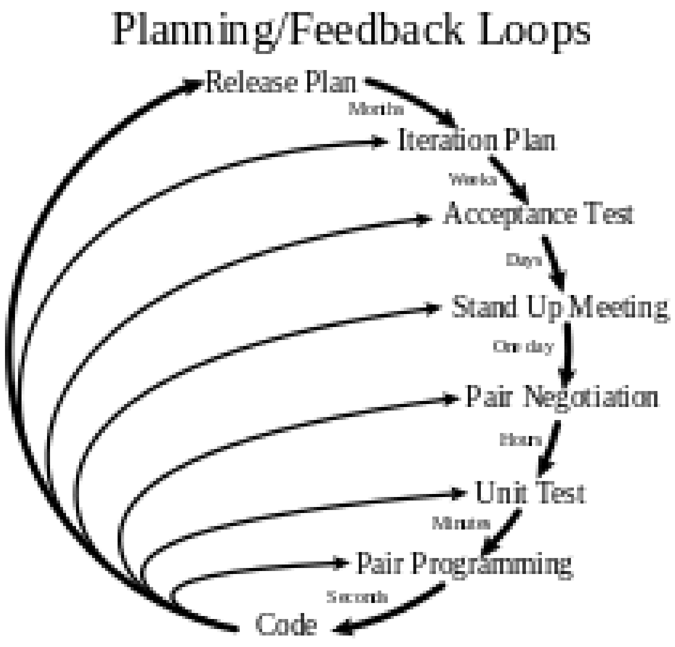
Player	SPE	DRI	DEF	PAS	SHO	
0 Vítor Manuel Martins Baía	78	90	88	64	88	79 2029
4 Guillaume Gillet	80	71	77	75	72	76 930
1 Thiago Emiliano da Silva	79	82	65	96	80	80 1282
1 Mathieu Bodmer	90	55	80	76	70	80 1008
1 Frank Rijkaard	87	70	68	87	68	79 1339
5 Arda Turan						76 911

into the categories – Goalkeeper, defense, midfield, attack. All of the respective roles have different weighting function determining their fitness.

Screenshot of the Fantasy Knapsack program running.

5. Some implementation problems

To avoid implementation problems we chose the extreme programming approach.

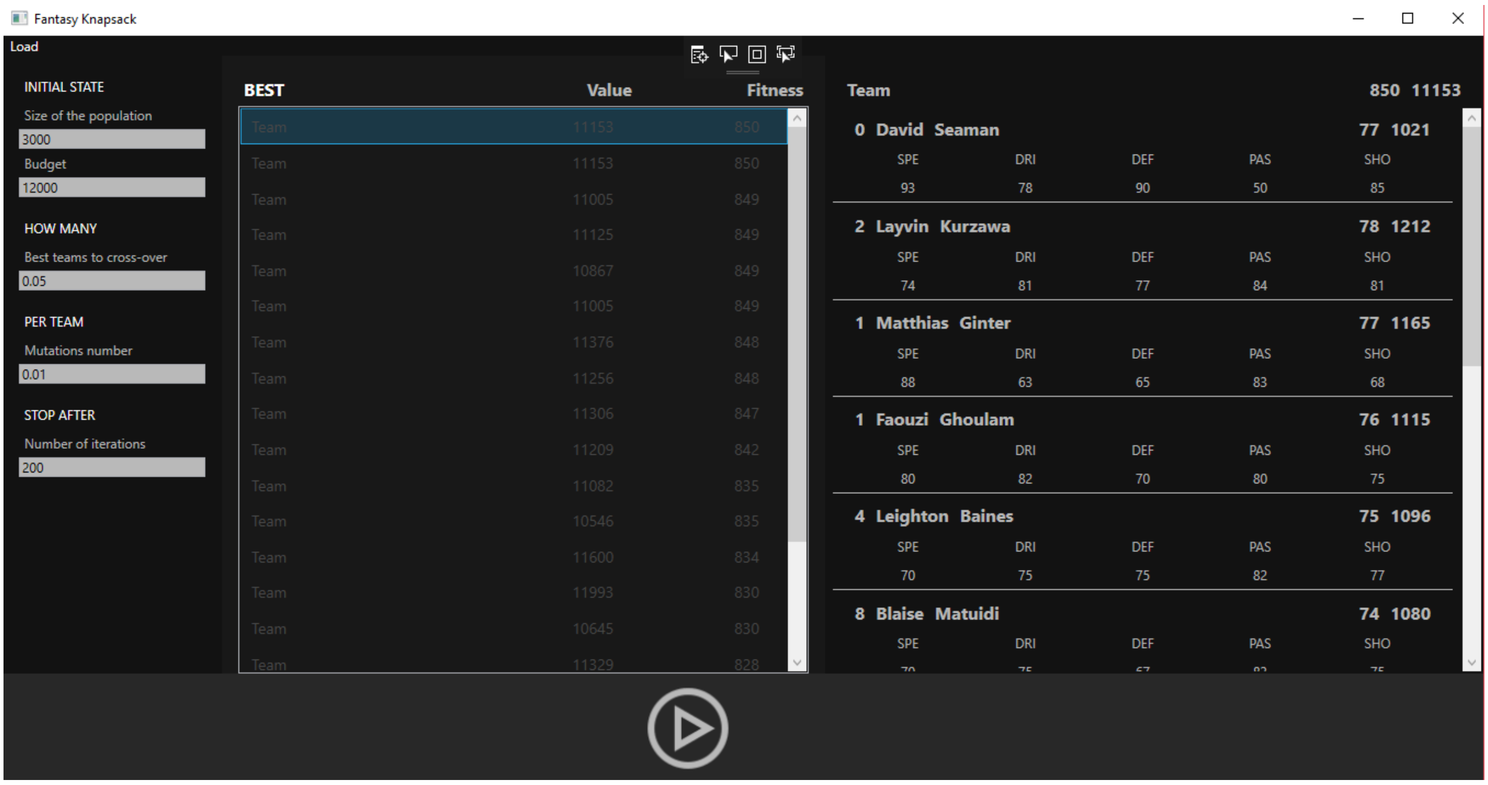


Workflow represented by extreme programming methodology.

During the implementation we have encountered the following problems:

- Integration of User interface with the backend algorithm
- Initial design of the application had minor flaws
- We didn't make satisfactory number of unit tests.
- Adaptation of player statistics to our problem has proven difficult due to the share number of them.
- Adjusting the weighting function for the algorithm.

6. End results of the application.



INITIAL STATE
Size of the population: 3000
Budget: 12000

HOW MANY
Best teams to cross-over: 0.05

PER TEAM
Mutations number: 0.01

STOP AFTER
Number of iterations: 200

BEST	Value	Fitness
Team	11153	850
Team	11153	850
Team	11005	849
Team	11125	849
Team	10867	849
Team	11005	849
Team	11376	848
Team	11256	848
Team	11306	847
Team	11209	842
Team	11082	835
Team	10546	835
Team	11600	834
Team	11993	830
Team	10645	830
Team	11329	828

Team 850 11153

Player	SPE	DRI	DEF	PAS	SHO		
0 David Seaman	93	78	90	50	85	77	1021
2 Layvin Kurzawa	74	81	77	84	81	78	1212
1 Matthias Ginter	88	63	65	83	68	77	1165
1 Faouzi Ghoulam	80	82	70	80	75	76	1115
4 Leighton Baines	70	75	75	82	77	75	1096
8 Blaise Matuidi						74	1080

Screenshot of the running application listing the best team achieved with 200 iterations, mutation rate of 0.01, cross-over rate of 0.05 budget of 12000 and the initial population of 3000.

7. Conclusion

With the application as the end result of this project we have achieved a usable tool that can be used for the "fantasy league" championships. Such possibility gives not only interesting conclusions about the creation of football teams but also allows the user to compete with other players in the "fantasy league" for monetary rewards. Further expansion of the algorithm may improve the results by reducing the speed of the computation as well as the better adjustment of the program to the real life sports matches scores.