



# *TP: Benchmark de résolutions de Sudokus*

Résolution par algorithme  
génétique avec GeneticSharp

# SOMMAIRE

- Processus de recherche
- Explication du code
- Bibliothèques à disposition





# THÉORIE DE L'ÉVOLUTION

- . La théorie synthétique de l'évolution (ou TSE) est une théorie darwinienne de l'évolution basée sur la sélection naturelle de variations aléatoires du génome
- . La théorie de l'évolution permet d'expliquer la diversité des formes de vie rencontrées dans la nature, en partant du principe que chaque espèce vivante se transforme progressivement au cours des générations, tant sur un plan morphologique que génétique.
- . Les mécanismes principaux par lesquels cette évolution se produit sont la sélection naturelle et la mutation
  - > l'une et l'autre étant intimement reliées aux processus qui permettent aux parents de transmettre à leurs descendants certaines caractéristiques



# SÉLECTION NATURELLE

## 3 Principes :

- Première itération: choix de l'individu Elite, celui qui passera à l'étape suivante, mais n'est pas assuré d'être le meilleur pour le développement de la population.
- Lors de l'itération suivante toute la population subit des mutations, on parle de crossover. (mutations aléatoires).
- On répète l'opération et on affine jusqu'à trouver la population idéale qui résout le sudoku avec le meilleur fitness possible (le moins d'erreur)

# PROCESSUS DE RECHERCHE

1

Comprendre la  
méthode ?



2

Étape: Github  
(recupération du code  
arbitre)



3

Analyse de solution  
possible

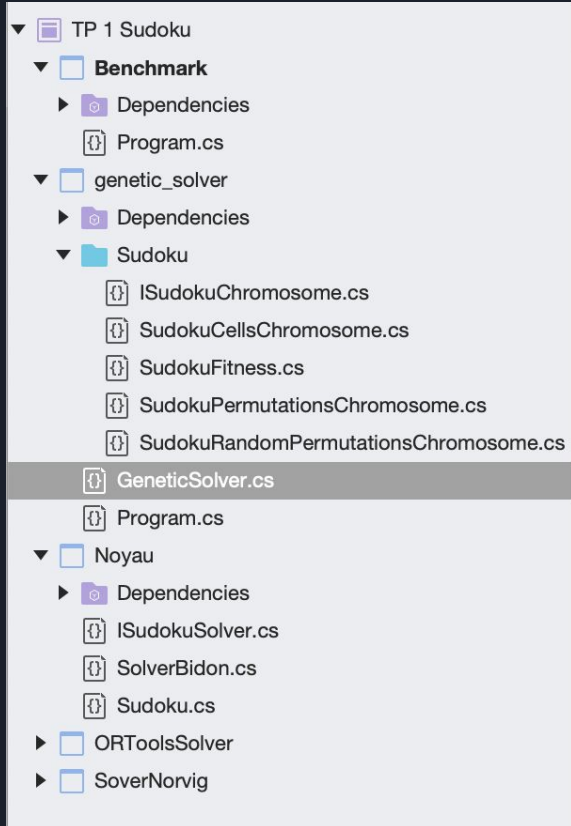



4

Résolution du  
sudoku

4	1	5	6	3	8	9	7	2
3	6	2	4	7	9	1	8	5
7	8	9	2	1	5	3	6	4
9	2	6	3	4	1	7	5	8
1	3	8	7	5	6	4	2	9
5	7	4	9	8	2	6	3	1
2	5	7	1	6	4	8	9	3
8	4	3	5	9	7	2	1	6
6	9	1	8	2	3	5	4	7

# EXPLICATION DU CODE





```
public static Sudoku Eval( Sudoku sudoku, int populationSize, double fitnessThreshold, int generationNb)
{
    //creation du chromosome
    IChromosome chromosome = new SudokuPermutationsChromosome(sudoku);

    //variable qui indique l'erreur
    var fitness = new SudokuFitness(sudoku);

    //choix de la selection : ici elite
    var selection = new EliteSelection();

    //Choix du crossover : ici uniform
    var crossover = new UniformCrossover();

    //choix de la mutation : ici uniform
    var mutation = new UniformMutation();

    //creation de la population
    var population = new Population(populationSize, populationSize, chromosome);

    //création de l'algo génétique
    var ga = new GeneticAlgorithm(population, fitness, selection, crossover, mutation)
    {
        //pour mettre fin a l'exec si on atteint le threshold ou le nombre de rep fixé
        Termination = new OrTermination(new ITermination[]
        {
            new FitnessThresholdTermination(fitnessThreshold),
            new GenerationNumberTermination(generationNb)
        })
    };

    ga.Start();
}
```



# RÉSOLUTION DU SUDOKU

```
//recupétaion de la meilleure solution  
var bestIndividual = ((ISudokuChromosome)ga.Population.BestChromosome);  
var solutions = bestIndividual.GetSudokus();  
  
return solutions[0];
```





# BIBLIOTHÈQUES UTILISÉES

GeneticSharp

```
1 using System;  
2 using System.Linq;  
3 using GeneticSharp;  
4 using GeneticSharp.Domain;  
5 using GeneticSharp.Domain.Chromosomes;  
6 using GeneticSharp.Domain.Crossovers;  
7 using GeneticSharp.Domain.Mutations;  
8 using GeneticSharp.Domain.Populations;  
9 using GeneticSharp.Domain.Selections;  
10 using GeneticSharp.Domain.Terminations;  
11 using genetic_solver;  
12 using Noyau;
```



# BILAN & TEST

## TEST :

Exécution de programme:	Temps d'exé:	Nombre d'erreurs:
pop=1000 it=100	8,6 s	-17
pop=5000 it=100	36s	-4
pop= 5000 it= 500	154s	-5

## BILAN D'ÉQUIPE :

Travail d'équipe de classe réalisé en temps dans une ambiance à la fois agréable et studieuse.