Haladó algoritmusok beadandó dokumentáció

Általánosságban:

Minden megoldó algoritmus egy interfészt vár a konstruktorban, a megoldófüggvény az interfészt megvalósító osztályok függvényeit hívja.

Hegymászó algoritmus interfésze:

Hegymászó algoritmus használatához, az adott problémának meg kell valósítania az IHillClimbingProblem<T> generitkus interfészt

```
public interface IHillClimbingProblem<T>
{
    2 references
    List<T> LoadStartState();
    2 references
    bool StopCondition(List<T> solution);
    2 references
    List<T> Distance(List<T> solution, float e);
    3 references
    float Fitness(List<T> solution);
}
```

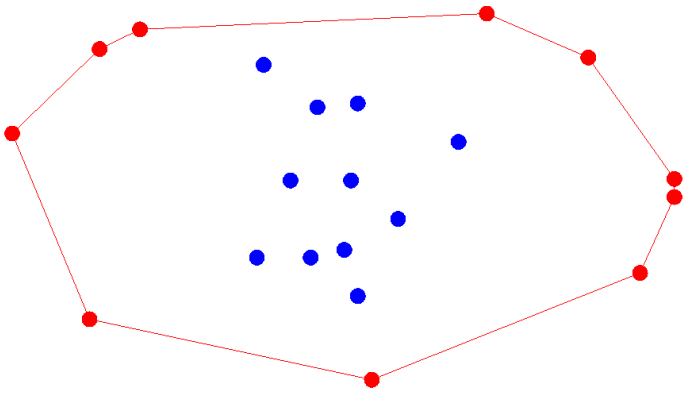
A beadandómban a SmallestBoundaryPoligon probléma valósítja meg.

Smallest Boundary Poligon:

- A helpers mappa smallest_boundary_fix.txt tartalmazza a kezdeti pontokat
- Minden iterációban 4 irányba mozdulhat el egy pont

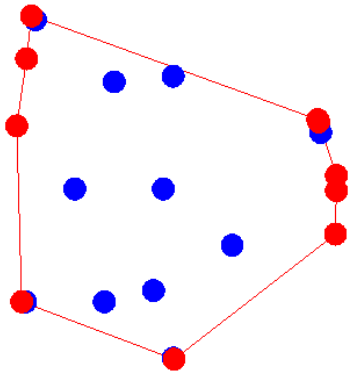
Az egyes iterációkat a LogViewerrel lehet figyelemmel kísérni.

Kezdeti állapot:



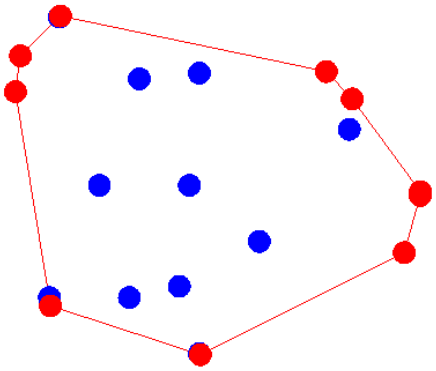
Első futásra 336 iteráció után:

```
<<  <  >  >> 336 P>  P>>  P>>>
```



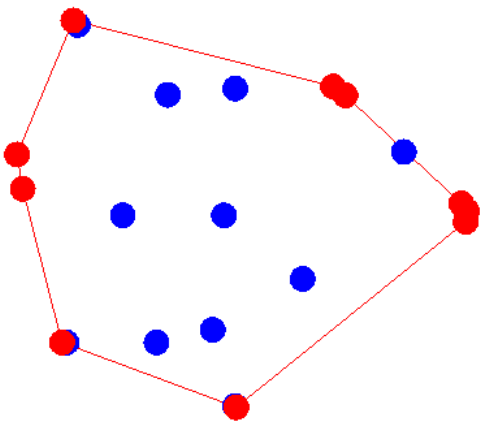
2. futás 276 iteráció:

```
<<  <  >  >> 276 P>  P>>  P>>>
```



3. futás 332 iteráció:

```
<<  <  >  >> 322 P>  P>>  P>>>
```



Jól látható, hogy a megoldás nem pontos, ezt a stopcondition függvény eltérő implementálásával és az epszilon finomításával is lehet javítani, de a hegymászó módszer nem garantál tökéletes megoldást.

Genetikus algoritmus interfész:

```
public interface IGeneticProblem<T>
{
    2 references
    int GetPopulationSize();
    2 references
    void PopultionOverWrite(List<T> newPopulation);
    2 references
    void InitializePopulations();
    3 references
    void Evaluation();
    2 references
    bool StopCondition();
    2 references
    List<T> SelectParents();
    2 references
    List<List<int>> Selection(List<T> elit);
    2 references
    T CrossOver(List<List<int>> matingPool);
    2 references
    T Mutate(T c);
    3 references
    T GetBestFitness();
}
```

Az én esetemben a függvényközelítés probléma valósítja meg, a stopcondition az, hogy a fitnesz 0 legyen, tehát egy tökéletes megoldást kapjak.

Függvényközelítés (függvényparaméterek keresése):

A függvényértékeket a Helpers mappa function_approx.txt fájl tartalmazza. Az A, B, C, D, E paraméterek kezdeti értékei egy -200 és 200 közötti random szám.

Futási eredmények (generationcounter a generációszámot mutatja):

```
A: 0, B: -14, C: 4, D: 6, E: 8, fitness: 0
GenerationCounter: 1678
END
```

```
A: 0, B: -4, C: 4, D: 6, E: 8, fitness: 0
GenerationCounter: 565
END
```

```
A: 0, B: -13, C: 4, D: 6, E: 8, fitness: 0
GenerationCounter: 853
END
```

```
A: 0, B: 0, C: 4, D: 6, E: 8, fitness: 0
GenerationCounter: 2484
END
```

```
A: 0, B: 16, C: 4, D: 6, E: 8, fitness: 0
GenerationCounter: 831
END
```

Többszöri futásra is látszik, hogy a B paraméter független a megoldástól.

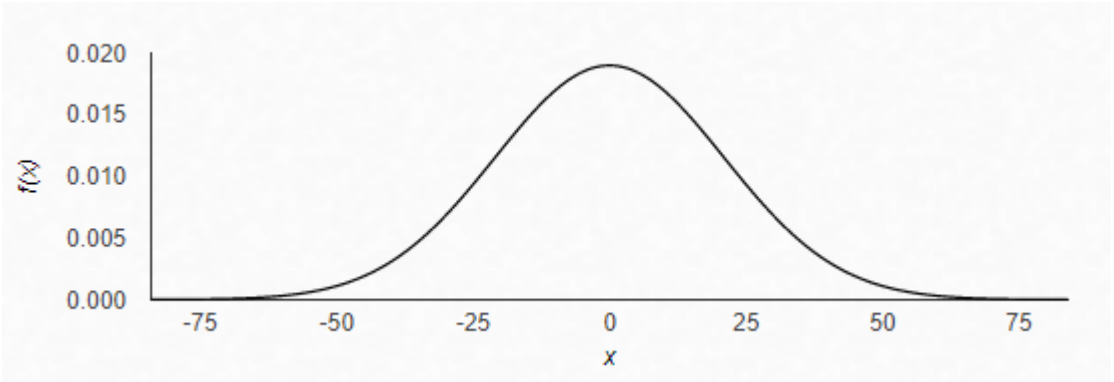
Random optimalizáció interfésze:

```
public interface IRandomOptimization<T>
{
    2 references
    List<T> Init();
    2 references
    bool StopCondition();
    2 references
    int RND();
    3 references
    double Fitness(List<T> solution);
    2 references
    void BetterSolutionHelper(List<T> oldSolution, List<T> newSolution);
    2 references
    List<T> Step(List<T> route, int step);
}
```

Utazó ügynök probléma:

A városok koordinátáit fájlból olvasom be. Mivel nincs fix epszilon érték, ami a hegymászónak nagy hátránya volt, így normál eloszlást kell beállítani. Nálam 0 körüli értékeket dobja leggyakrabban és a városok számánál nagyobb értéket jóval ritkábban. Ha negatív értéket dob, annak az abszolút értékével számolok.

Nálam így néz ki a görbe: városok száma: 64

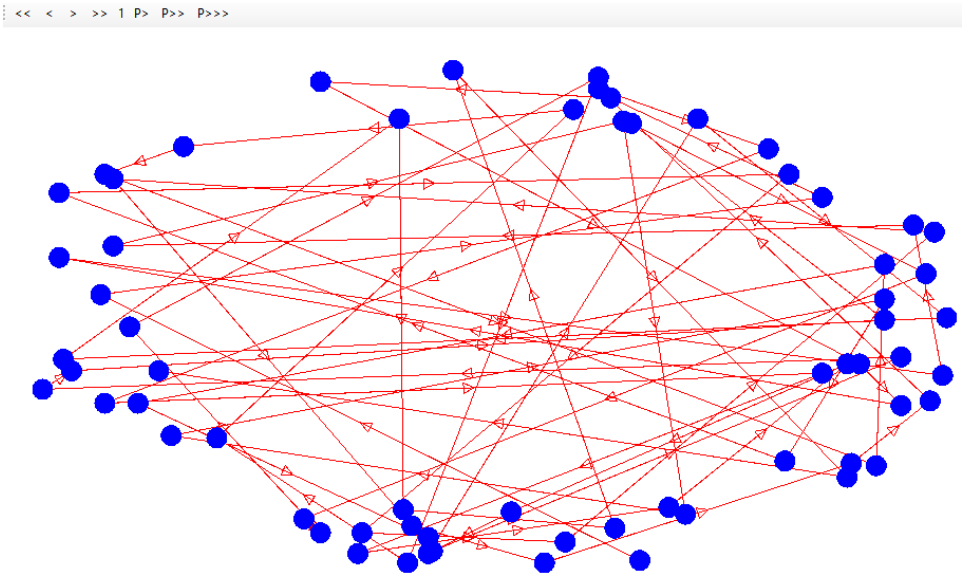


A normál eloszlás által meghatározott szám azt jelenti a feladat esetében, hogy hány várost cserélek meg. Ha a városok számánál nagyobb értéket dob, akkor a városok száma lesz a dobott érték.

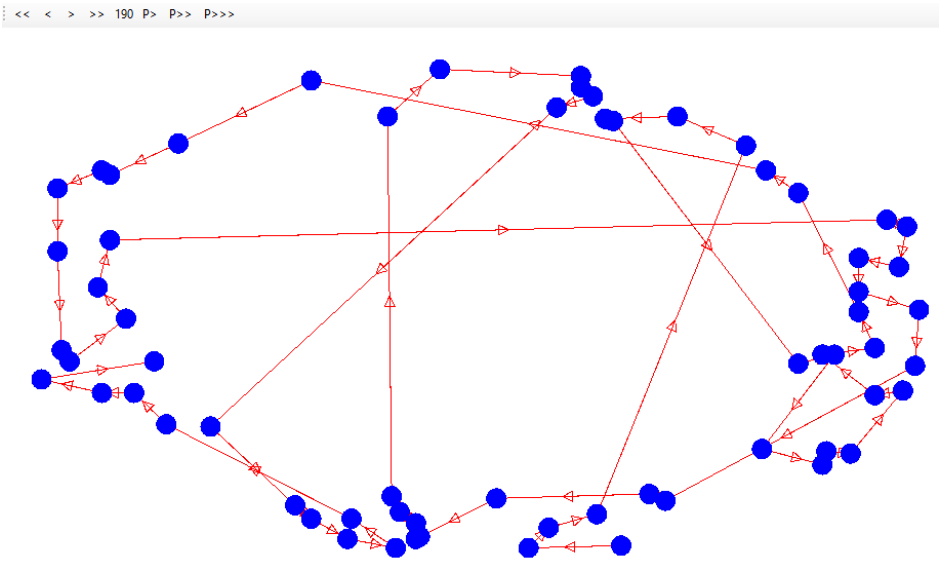
Megállási feltételem az, hogy ha a legutolsó 5 új megoldás fitneszének összege nem nagyobb, mint 10, akkor érjen véget a probléma. Ezzel természetesen előjöhet az a lehetőség, hogy nem találja meg a globális optimumot, de így gyorsabban véget ér az algoritmus futása. Ennek kalibrálásával, finomításával tudjuk elérni, hogy a globális optimumot megtalálja, viszont így a futási ideje nehezen jósolható.

Első futás eredménye:

Kezdő állapot (random):



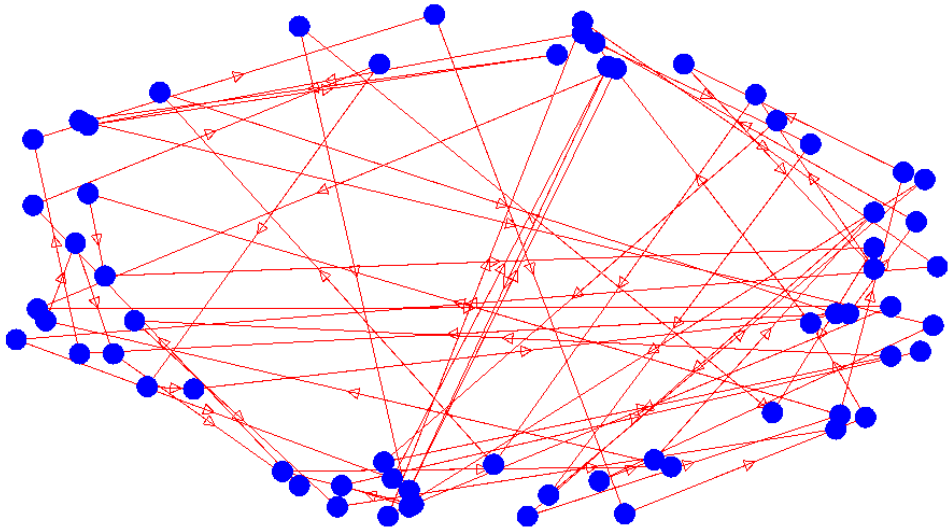
Eredmény:



Második futás:

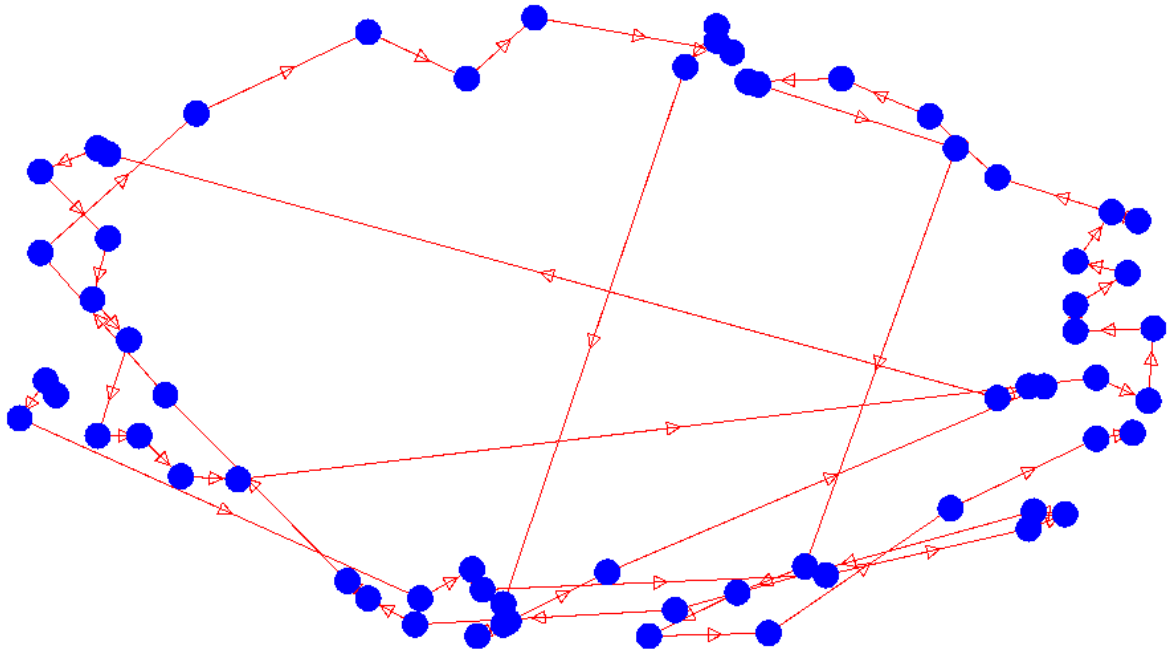
Kezdeti állapot:

```
<<  <  >  >> 1 P>  P>>  P>>>
```

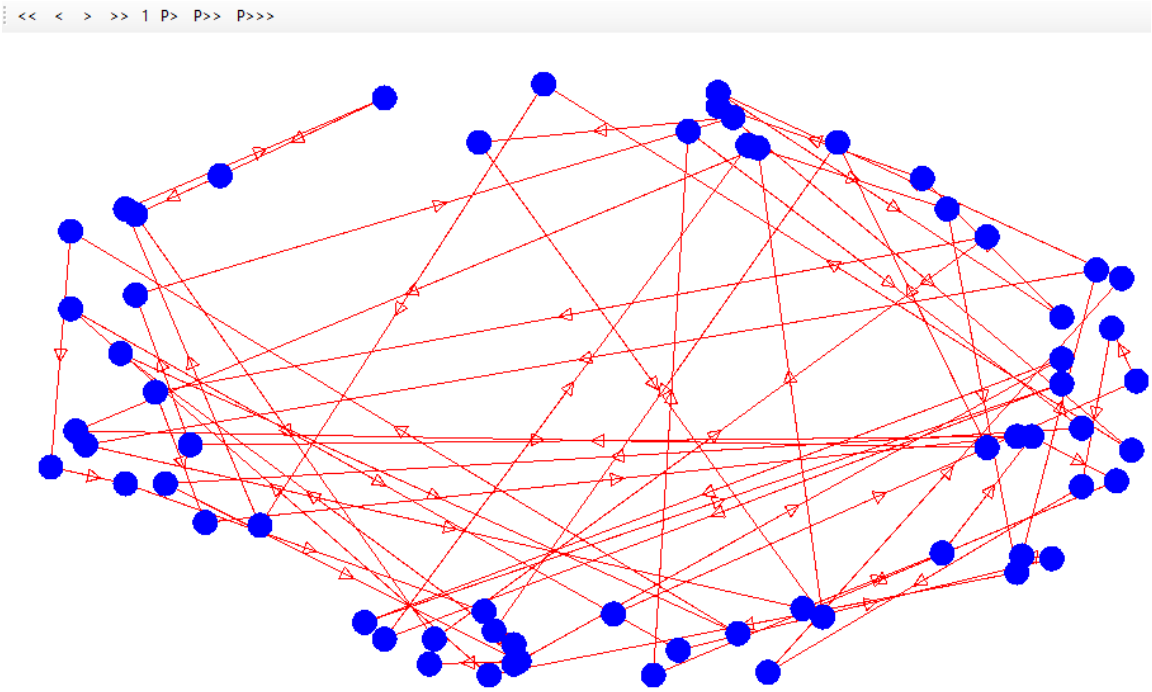


Végeredmény:

```
<<  <  >  >> 168 P>  P>>  P>>>
```



Harmadik futás:



Végeredmény: