

Metody Sztucznej Inteligencji  
Optymalizacja wielokryterialna.

Paweł Malczyk  
26.XII.2009

1. **Cel** – celem zadania było:

- Implementacja algorytmu optymalizacji wielokryterialnej NSGA-II
- Rozwiązaniu problemów benchmarkowych ZDT1-ZDT4
- Zaprezentowaniu wyników w postaci metryk HVR.

2. **Realizacja:**

**Ważniejsze klasy:**

**Real** – Opakowuje wartość liczby rzeczywistej ( $x_i$ ) w zmiennych decyzyjnych problemów ZDT ( $x_i$ ), z uwzględnieniem ograniczeń co o wartości definiowanych (Dla obliczania wartości funkcji w problemach ZDT używamy zbioru liczb rzeczywistych).

**ProblemVariables** – Wiąże zmienne decyzyjne problemu z samym problemem (ZDT), przechowuje odpowiednie wartości  $x_i$  ( $i=1,2,\dots,n$ ), i udostępnia ilość zmiennych decyzyjnych.

**Individual** – reprezentuje pojedynczego osobnika populacji który poddawany jest operacjom (np. krzyżowania, selekcji) oraz wykonuje (oblicza) jakieś polecone mu zadania (ZDT). Osobnik dodatkowo ma właściwości związane z samym algorytmem NSGA takie jak przynależność do frontu oraz wielkości cuboidu.

**Population** – reprezentuje zbiór osobników (Populację). Używany jako całość przez operator selekcji, do przechowania populacji potomków, ale także istnieje możliwość łączenia populacji.

**SimpleSelection** – bardzo prosta implementacja operator selekcji osobników z populacji (ogólnie działa w sposób losowy).

**SBXCrossover** – implementacja operatora krzyżowania Simulated Binary Crossover (wzory do obliczenia wartości skrzyżowanego potomka, oraz jakieś przykłady na których bazowano znaleziono w sieci).

**PolynomialMutation** – Implementacja operatora w postaci wielomianowej (wzory do obliczenia wartości zmutowanego potomka, oraz jakieś przykłady na których bazowano znaleziono w sieci).

**Problem** – abstrakcja dla problemów ZDT, przechowuje informacje na temat ograniczeń wartości zmiennych decyzyjnych w problemach ZDT (przedział do którego należy  $x_i$  [ $i=1,2,\dots,n$ ] oraz ilość ( $n$ ) tych iteracji sumowania wartości  $x_i$ ).

**ZDT1 do ZDT4** – implementacje do obliczania wartości funkcji z problemów ZDT (zwykle wstawienie do wzoru i obliczenie realizowane przez pojedynczego osobnika przy użyciu **ProblemVariables**).

**Distance** – Klasa pomocnicza odpowiadająca za obliczanie wielkości „crowding distance” dla każdego  $i$ -tego rozwiązania z frontu (z pkt. 3.2 opisu Density Estimation).

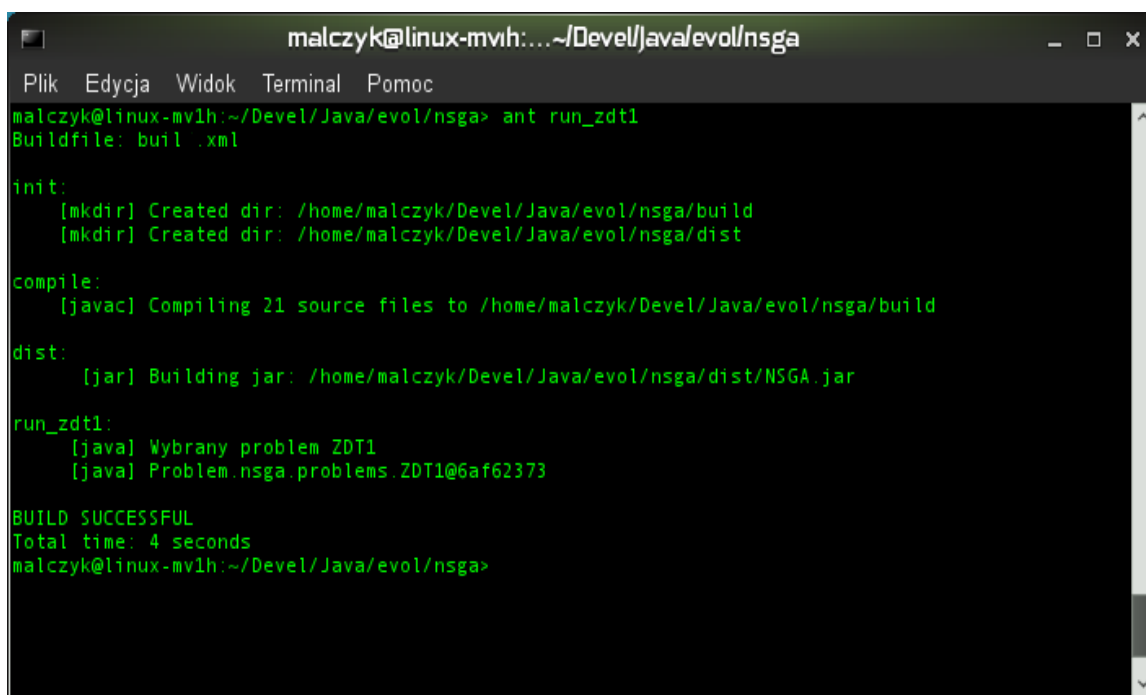
**Ranking** – Klasa zawiera właściwą implementację algorytmu „fast-nondominated-sort” na populacji, oraz w niej jest zawarta logika identyfikowania frontów.

### 3. Uruchomienie:

W celu łatwego uruchamiania stworzono prosty skrypt ANT, który automatycznie uruchamia i wykonuje odpowiedni problem zdt. W skrypcie występują następujące cele.

- dist – tworzenie uruchamialnego archiwum JAR.
- run\_zdt1 – uruchomienie problemu ZDT1
- run\_zdt2 – uruchomienie problemu ZDT2
- run\_zdt3 – uruchomienie problemu ZDT3
- run\_zdt4 – uruchomienie problemu ZDT4

Po uruchomieniu któregoś z celów z wykonaniem odpowiedniego problemu tworzony jest plik danych nazwie odpowiadającej nazwie problemu, który to plik jest podstawą do stworzenia odpowiedniego wykresu.



```
malczyk@linux-mv1h:~/Devel/Java/evol/nsga
Plik Edycja Widok Terminal Pomoc
malczyk@linux-mv1h:~/Devel/Java/evol/nsga> ant run_zdt1
Buildfile: build.xml

init:
[mkdir] Created dir: /home/malczyk/Devel/Java/evol/nsga/build
[mkdir] Created dir: /home/malczyk/Devel/Java/evol/nsga/dist

compile:
[javac] Compiling 21 source files to /home/malczyk/Devel/Java/evol/nsga/build

dist:
[jar] Building jar: /home/malczyk/Devel/Java/evol/nsga/dist/NSGA.jar

run_zdt1:
[java] Wybrany problem ZDT1
[java] Problem.nsga.problems.ZDT1@6af62373

BUILD SUCCESSFUL
Total time: 4 seconds
malczyk@linux-mv1h:~/Devel/Java/evol/nsga>
```

### 4. Zasada działania:

Na początku generowana jest losowa populacja 100 osobników.

W każdym pokoleniu (*NSGAI1.java*) (pokoleń jest 25000) na tworzona jest para osobników, która jest wynikiem krzyżowania (*SBXCrossover.java*) dwóch osobników wyselekcjonowanych (*SimpleSelection.java*) z populacji osobników rodziców.

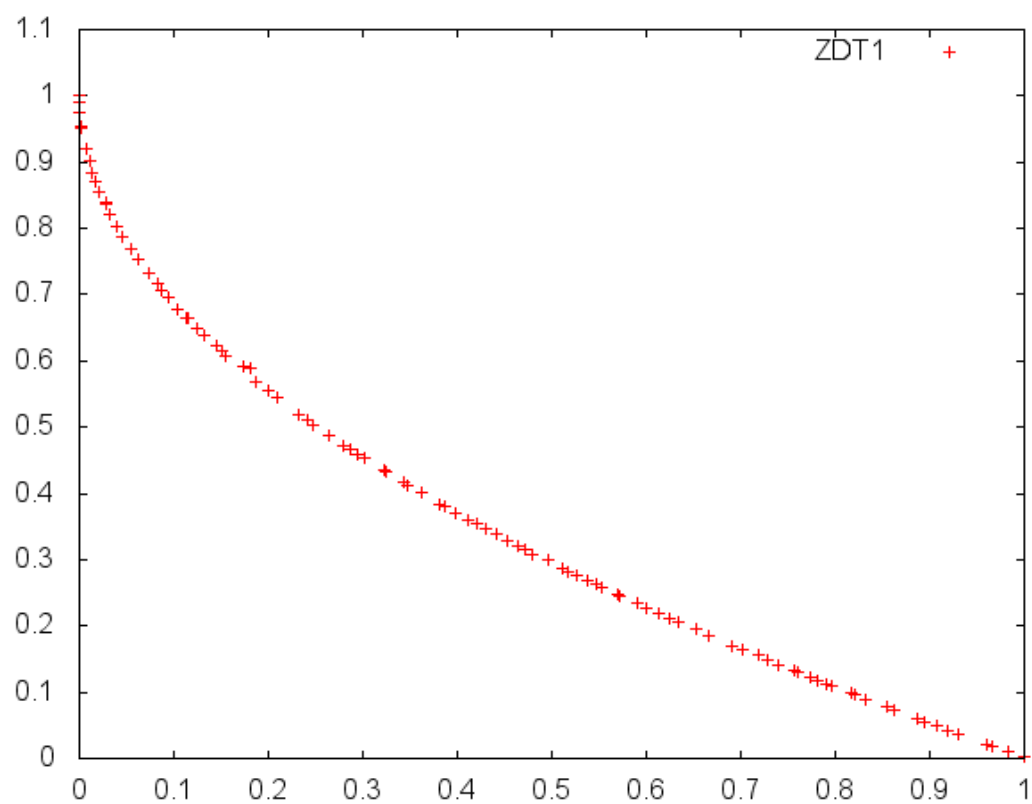
Każdy z powstałych osobników potomków poddawany jest mutacji (*PolynomialMutation.java*). Następnie każdy z tych potomków dokonuje obliczenia funkcji zdefiniowanej w odpowiednim problemie ZDT. Ostatnim krokiem w każdym pokoleniu t jest wykonanie kroków zawartych w punkcie 3.4 (Main Loop) czy trzy główne czynności. Połączenie populacji osobników rodziców oraz potomstwa.

Wykonanie algorytmu „fast non-dominated-sort” (*Ranking.java*) czyli wyznaczenie frontów niedominowanych (bazuje na obliczonych wcześniej wartościach (*DominanceComparator.java*)).

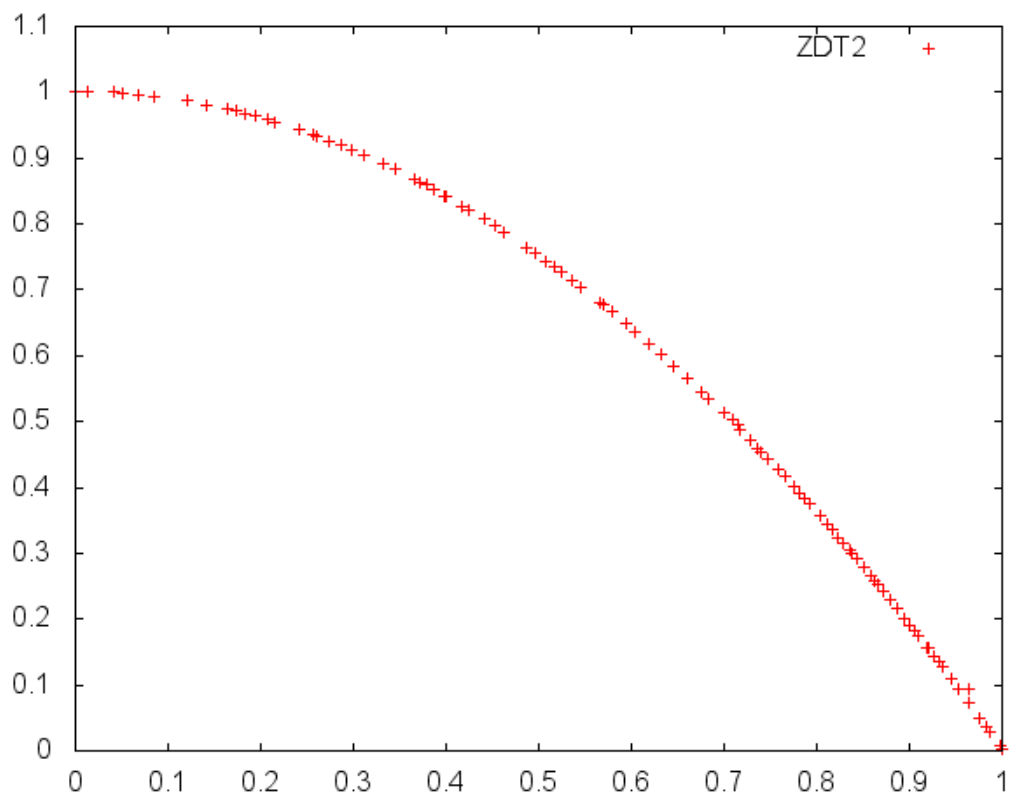
Obliczenie wielkości „crowding distance” (pkt 3.2 opisu) (*Distance.java*) dla każdego osobnika znajdującego się na froncie.

**5. Wyniki:**

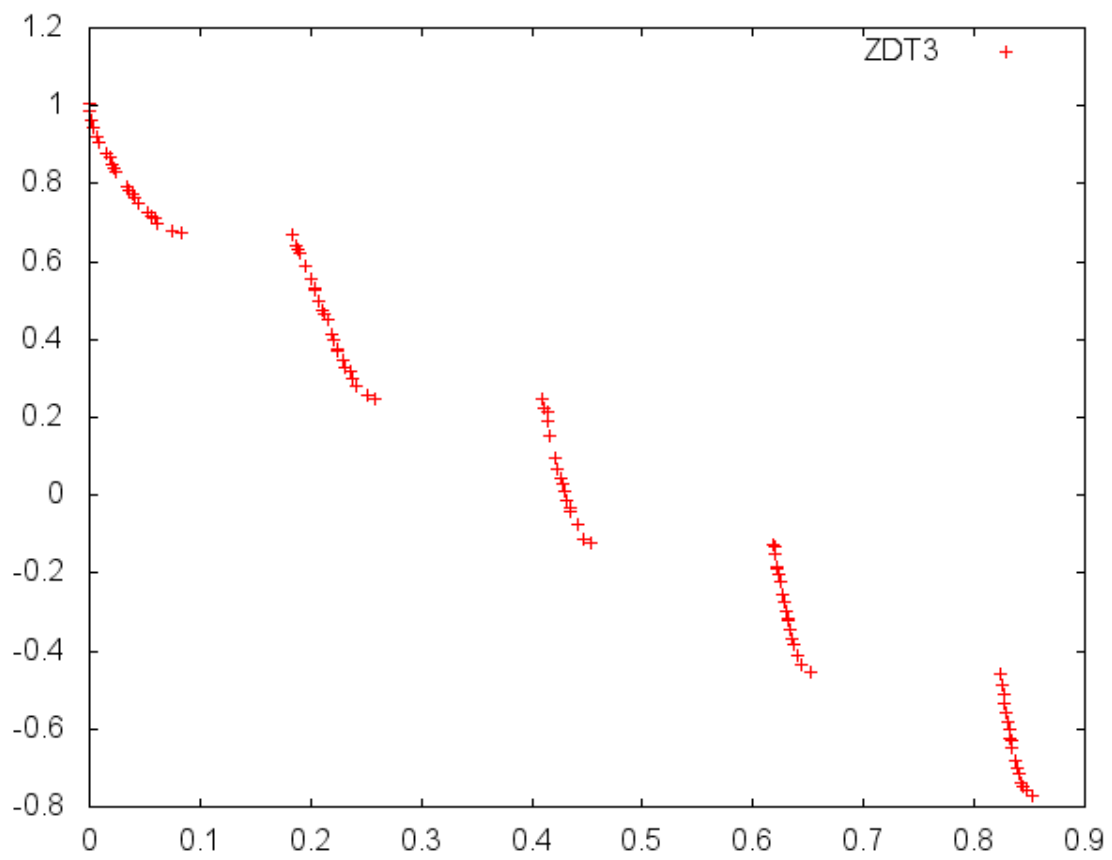
**ZDT1:**



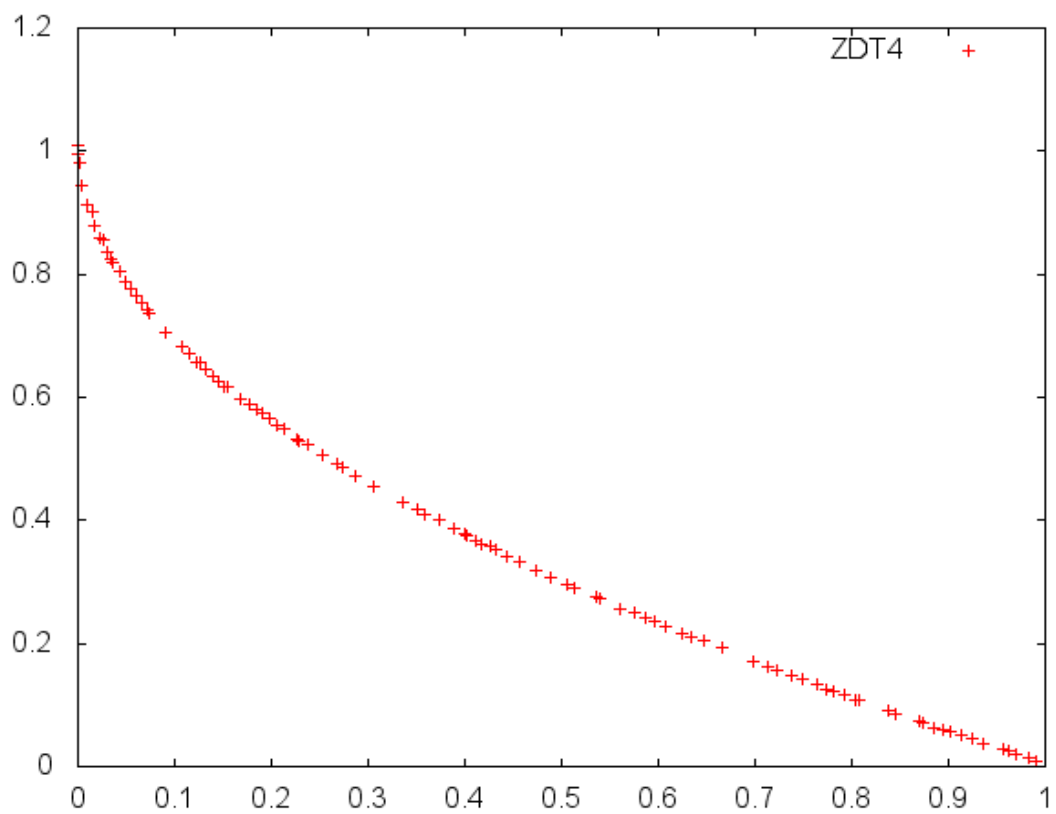
**ZDT2:**



**ZDT3:**



**ZDT4:**



## 6. Metryki HVR

Obecnie brak implementacji jak czas pozwoli zostanie to jeszcze uzupełnione, ale pewnie nie uda się tego zrobić przed 11.01.2010.