Sztuczna inteligencja i inżynieria wiedzy laboratorium

Ćwiczenie 1. Algorytmy genetyczne

opracowanie: P.Myszkowski. M.Laszczyk, H.Kwaśnicka

Cel ćwiczenia

Zapoznanie się z metaheurystyką algorytmów genetycznych w praktyczny sposób poprzez samodzielną implementację.

Realizacja ćwiczenia

- Zapoznanie się z metaheurystyką algorytmów genetycznych
- Określenie problemu optymalizacyjnego do rozwiązania – minimalizacja czasu realizacji harmonogramu w problemie MS-RCPSP
- Zbudowanie algorytmu genetycznego: osobnik, funkcja oceny, krzyżowanie, mutacja, selekcja, inicjalizacja
- Implementacja modelu w dowolnym języku obiektowym (sugerowana Java)
- Zbadanie wpływu różnych parametrów (prawd. mutacji, krzyżowania, selekcji, rozmiar populacji, liczba pokoleń) na efektywność i skuteczność metody
- Sporządzenie sprawozdania z ćwiczenia
- Pokazanie na wykresach zmianę wartości przystosowania (wartość optymalizowanej funkcji celu) w poszczególnych pokoleniach: najlepszy osobnik, średnia wartość w populacji i najgorszy osobnik
- Porównaj działanie algorytmu genetycznego z wybraną przez siebie, nieewolucyjną metodą optymalizacji (wybierz jedną: metoda losowego przeszukiwania, greedy). Zwróć uwagę na uzyskany wynik, czas działania oraz liczbę wartościowań optymalizowanej funkcji celu
- Pokaż i omów najciekawsze wyniki
- Raport z ćwiczenia powinien zawierać wszystkie punkty
- do pomocy w realizacji zadania udostępnione są: pliki testowe (*.def), biblioteka w Javie z funkcją oceny i algorytmem zachłannym, oraz zewnętrzne narzędzie walidacji rozwiązań (validator.jar).

Zasada działania algorytmu genetycznego

Algorytm genetyczny jest metaheurystyką, która naśladuje ewolucję naturalną metodą ciśnienia selekcyjnego i doboru naturalnego. Aby ją zastosować, należy zdefiniować potencjalne rozwiązanie (osobnika), sposoby jego zmiany (mutacja), łączenia (krzyżowania) oraz oceny jakości rozwiązania (funkcja oceny). Algorytm genetyczny opiera się na schemacie przedstawionym jako Pseudokod 1.

```
begin
t:=0;
initialise(pop_0);
evaluate(pop_0);
        while (not stop_condition) do
        begin
           pop_t+1:= selection(pop_t);
           pop_t+1:= crossover(pop_t+1);
           pop_t+1:= mutation(pop_t+1);
           evaluate(pop_t+1);
           t:=t+1;
        end
end
```

Pseudokod 1. Pseudokod Algorytmu Genetycznego

GA wstępnie inicjalizuje (zwykle losowo) populację rozwiązań. Następnie ocenia jakość poszczególnych osobników. W kolejnym kroku sprawdzane są warunki zatrzymania: czy osiągnięto akceptowalne rozwiązanie i/lub limit pokoleń został przekroczony. Wybór osobników (sugerowana metoda ruletki lub turnieju) do następnej populacji następuje przed działaniem operatora krzyżowania i mutacji, które budują osobniki nowego pokolenia. Dalej, następuje ocena nowych rozwiązań i cykl GA się zamyka przy sprawdzaniu warunków zatrzymania.

Problem harmonogramowania projektu

MS-RCPSP (skrót od ang. *Multi-Skill Resource Constrained Project Scheduling Problem*) jest wariantem problemu harmonogramowania. Problem jest zdefiniowany poprzez zbiór zadań do wykonania oraz zbiór zasobów, które mogą wykonywać zadania. Dodatkowo z każdym zadaniem skojarzony jest zbiór umiejętności, z której każda określona jest poprzez rodzaj oraz poziom biegłości. Zasoby mają także powiązany z nim koszt użycia, który potem pozwala na obliczenie kosztu realizacji całego harmonogramu. Rozwiązanie problemu polega na przyporządkowaniu zasobów (wraz z czasem startu) do zadań tak, aby nie łamały poniższych **ograniczeń**:

- tylko jeden zasób może pracować nad zadaniem w jednym czasie,
- zasób może wykonywać tylko jedno zadanie w danym czasie,
- zasób może rozpocząć pracę, tylko jeśli posiada wszystkie potrzebne umiejętności na wymaganym bądź wyższym poziomie,
- wszystkie zadania muszą zostać wykonanie, tj. muszą mieć przyporządkowany zasób,
- zadania posiadają zbiór poprzedników – aby nie naruszyć ograniczeń, wszystkie zadania z tego zbioru muszą być wykonane,

Uzyskane rozwiązanie to **harmonogram**, dla którego możemy określić czas realizacji (czas końca pracy nad ostatnim zadaniem) oraz koszt harmonogramu. Dla ułatwienia realizacji zadania ocena jakość harmonogramu zostanie zawężona tylko do czasu realizacji harmonogramu.

Algorytm Genetyczny (GA) jest metodą ukierunkowanego przeszukiwania przestrzeni rozwiązań. Punktem tej przestrzeń jest w naszym wypadku harmonogram, który interpretujemy jako przydziały <zadanie,zasób,czas> przy uwzględnieniu ograniczeń przedstawionych w poprzednim punkcie. Rozwiązanie problemu harmonogramowania sprowadza się do zdefiniowana przestrzeni przeszukiwań.

Problem może być rozwiązany przez GA na wiele sposobów i zależy to tylko od inwencji projektanta. Najważniejsze jednak jest, aby w efekcie uzyskać harmonogram oparty na jednoznacznych przydziałach <zadanie,zasób,czas>. Poniżej przedstawiono dwa przykładowe sposoby reprezentacji. Najprostszy (macierzowy) jest nadmiarowy, jednak nie wymaga dodatkowych metod układania harmonogramu. W przypadku drugiej propozycji (wyspecjalizowane) wykorzystuje się *de facto* podział problemu na dwa podproblemy <zadanie,zasób> i <zadanie,czas>. Taka reprezentacja jest prostsza dla GA, jednak wymaga użycia innej metody – tutaj zaproponowano algorytm zachłanny.

Naturalnym rozwiązaniem może być **chromosom jako macierz**, która przypisuje dla każdego zadania zasób i czas realizacji. Przykładowy chromosom (patrz Tab.1.) składa się z macierzy zawierającej czasy (reprezentujemy jako dyskretne wartości) rozpoczęcia zadania przez każdy zasób nad każdym zadaniem.

Tab. 1. Chromosom jako macierz przypisań

Zadanie / zasób	Zadanie 1	Zadanie 2	Zadanie 3	Zadanie 4
zasób 1	czas11	czas12	czas13	czas14
zasób 2	czas21	czas22	czas32	czas24
zasób 3	czas31	czas32	czas33	czas34
zasób 4	czas41	czas42	czas43	czas44

Takie najprostsze podejście to ma jedną dużą zaletę – właśnie prostotę. Niestety nie jest wolne od wad, bo z jednej strony jest nadmiarowe (każde z zadań ma przypisane wiele zasobów i czasów) oraz potencjalnie może łamać ograniczenia. Przy takiej nadmiarowej reprezentacji, w przypadku niejednoznaczności, należy zdecydować, który z zasobów jest wybierany (np. najtańszy).

Inne podejście, już **wyspecjalizowane**, może sprowadzać się do podziału problemu właściwego na dwa podproblemy: (1) przypisanie zasobów i (2) przypisanie terminów realizacji. Przykładowo, (1) przypisanie zasobów może realizować GA, a (2) może być oparte na algorytmie zachłannym. W takim wypadku chromosom jest zbudowany jako wektor interpretowany jako przypisanie zadanie-zasób (patrz Tab.2). W tej reprezentacji powinno się pilnować, aby przydzielone zasoby były poprawnie przydzielone do zadań. Drugim wyjściem jest „naprawianie" osobnika, jeśli zadanie w wyniku działania operatorów genetycznych uzyskał niepoprawne przypisanie.

Tab.2. Przykładowy chromosom jako przypisanie zadanie-zasób

Zadanie	1	2	3	4
zasób	zasób 5	zasób 2	zasób 1	zasób 2

W powyższej reprezentacji wektorowej (dla przykładu z Tab.2. chromosomem jest wektor <5,2,1,2>) rozwiązanie pomija ustalenia czasów rozpoczęcia zadań. Do tego można wykorzystać **algorytm zachłanny**, który dla wszystkich zadań po kolei przyporządkowuje najwcześniejszy możliwy czas, rozważając czas pracy nad poprzednim zadaniem oraz koniec pracy nad wszystkimi poprzednikami. Dla ułatwienia algorytm jest zaimplementowany w udostępnionej bibliotece.

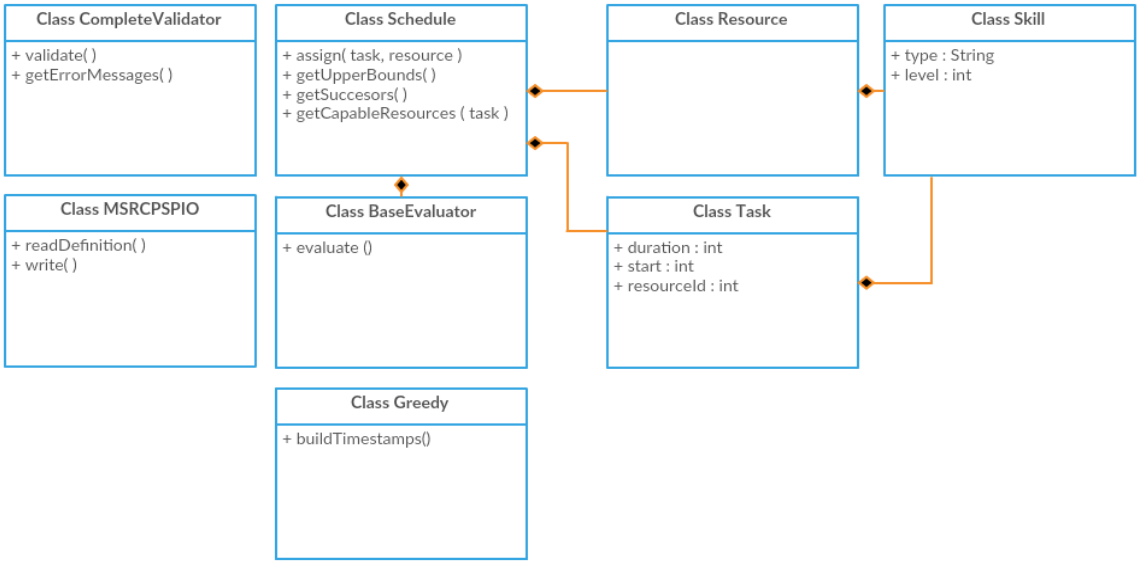
Algorytm genetyczny do działania zwykle potrzebuje dwóch **operatorów genetycznych**: krzyżowania i mutacji. Krzyżowanie polega na wymianie materiały genetycznego pomiędzy osobnikami, zwykle w najprostszym przypadku wyznaczając losowy punkt „cięcia".
Np. mając dwa osobniki: P1:=<**1,2,3,4,5**> i P2:=<6,7,8,9,10> i wyznaczając punkt cięcia na 3 pozycji, możemy otrzymać dwa osobniki potomne: O1:=<**1,2,3,**9,10> i O2:=<6,7,8,**4,5**>.
W przypadku mutacji operator działa tylko na jednym osobniku i w najprostszej wersji zmianie ulec losowo (z prawdopodobieństwem *Pm*) wartość „genu". Np. mając osobnik O1:=<1,2,**3**,9,10> i wyznaczeniu trzeciego genu do zmiany osobnik O1 zmienić się może O1:=<1,2,**10**,9,10>.

Inicjalizacja w GA zwykle jest losowa, tj. populacja pierwsza jest generowana losowo. W przypadku selekcji proponuje się stosowanie metody ruletki lub metody turniejowej. Metoda ruletki określa prawdopodobieństwo wyboru osobnika do rozrodu proporcjonalnego do jego jakości przystosowania. Druga metoda (turniejowa) zakłada, że z populacji wybierane jest losowo *Tn* (parametr metody) osobników i spośród nich wybierany jest osobnik o najlepszej jakości. Dla *Tn=1* mamy losowy wybór i brak ciśnienia selekcyjnego i GA „błądzi", dla *Tn* równemu rozmiarowi populacji zwykle jest wybierany najlepszy osobnik i powoduje to szybkie utykanie w lokalnych optimach.

Proszę pamiętać o parametrach GA: rozmiar populacji (*pop_size*), liczbie pokoleń (g*en*), prawdopodobieństwo mutacji (*Pm*) i prawdopodobieństwo krzyżowania (*Px*). Ew. parametrem może być też rozmiar turnieju. Proponuje się rozpoczęcie badań dla następujących (**domyślnych**) wartości parametrów: *pop_size*=100, *gen*=100, *Pm*=0,01, *Px*=0.1 i *Tn*=5. Uwaga! Nie są to wartości optymalne, które dla różnych implementacji mogą różnie się zachowywać.

Biblioteka

Na potrzeby realizacji zadania została udostępniona biblioteka, która realizuje najważniejsze funkcjonalności rozwiązywanego problemu MS-RCPSP.



Rys. 1. Uproszczony diagram klas udostępnionej biblioteki

Zaimplementowana jest metoda zachłanna, która umożliwia ustawienie zadań na osi czasu. Polega ona na iteracji po wszystkich zadaniach i każdemu z nich przyporządkowywaniu najwcześniejszego możliwego czasu.

```
// create greedy algorithm to set timestamps
Greedy greedy = new Greedy(schedule.getSuccesors());
// set starts of each task using greedy algorithm
greedy.buildTimestamps(schedule);
```

Pseudokod 2. Pseudokod algorytmu zachłannego do ustawienia zadań na osi czasu

Dostępna klasa *BaseIndividual* korzysta z evaluatora, który umożliwia policzenie czasu harmonogramu. Istnieje również funkcja zwracająca wartości znormalizowane. Czas harmonogramu reprezentuje dopasowanie osobnika.

```
BaseValidator validator = new CompleteValidator();
System.out.println(validator.validate(schedule));
System.out.println(validator.getErrorMessages());
```

Pseudokod 3. Uruchomienie funkcji oceny oraz uzyskanie czasu harmonogramu

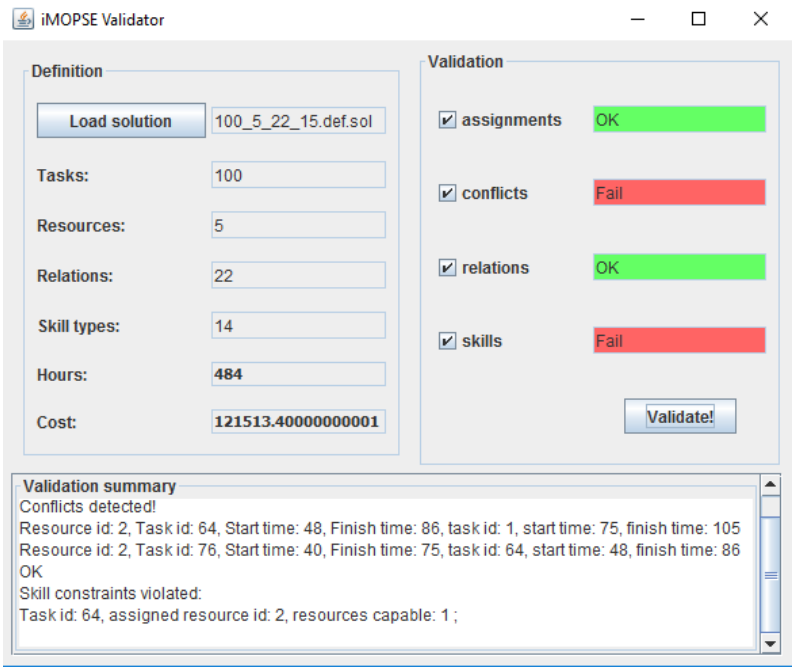
Zapewnienia ograniczeń ułatwić może funkcja sprawdzająca, czy dane zadanie może zostać wykonane przez dany zasób oraz metoda zwracająca wszystkie zasoby mogące pracować nad danym zadaniem.

```
schedule.canDoTask(task, resource); //pasuje zasób?
schedule.getCapableResources(task) //lista wszystkich popr. zasobów
```

Pseudokod 4. Sprawdzenia czy zasób pasuje do zadania / pobranie wszystkich zasobów pasujących do zadania

Dostępne jest także narzędzie zewnętrze **Validator(.jar)** sprawdzający poprawność utworzonych harmonogramów (link poniżej, zapis harmonogramu do pliku jest zaimplementowany w bibliotece). Po wygenerowaniu wynikowego harmonogramu do pliku i podaniu pliku definicji validator wskaże czy harmonogram jest poprawny i jeśli nie określi szczegóły złamanych ograniczeń – patrz Rys.2.

Narzędzie do walidacji może być szczególnie pomocne dla osób chcących napisać rozwiązanie zadania w innym języku niż sugerowany (Java) lub bez wykorzystania podanej biblioteki. Takie zewnętrze narzędzie walidacji pozwala na określenie czy podane rozwiązanie jest poprawne.



Rys. 2. Przykład prezentacji szczegółów niepoprawnego harmonogramu.

Celem zadania jest sprawdzenie jak algorytmy genetyczne sprawdzają się w problemie optymalizacyjnym, tj. minimalizacji czasu realizacji harmonogramu projektu.

Do niniejszej treści zadania dołączony został plik **ga-ms-rcpsp.zip** zawierający:

- dane testowe do pracy (def_small.zip). Dla chętnych istnieje możliwość sprawdzenia swojego algorytmu na danych bardziej złożonych (do ściągnięcia ze strony projektu iMOPSE).
- kod źródłowy biblioteki (AI.zip) opisanej wyżej, która zawiera zaimplementowaną definicję problemu, wczytywanie/zapis do pliku
- plik biblioteki validatora (validator.jar) wraz z dwoma dokumentami opisu użycia oraz przykładami działania.

Więcej informacji o problemie MS-RCPSP i opis metod rozwiązywania dostępne są pod adresem z materiałami projektu iMOPSE: ***http://imopse.ii.pwr.edu.pl/***

Ocena realizacji ćwiczenia

1pkt Zbudowanie modelu algorytmu genetycznego
1pkt Implementacja algorytmu genetycznego
2pkt Zbadanie działania na 6 małych plikach testowych (zbiór small_defs)
2pkt Zbadanie wpływu prawd. krzyżowania i mutacji na wyniki działania GA
2pkt Zbadanie wpływu rozmiaru populacji i liczby pokoleń na wyniki działania GA
1pkt Zbadanie wpływu selekcji na skuteczność GA
1pkt Porównanie skuteczności GA z wynikami innej metody nieewolucyjnej

Uwaga! Przy testach należy brać pod uwagę przynajmniej 10 uruchomień i uśrednianie wyniku (podajemy wartość średniej i odchylenie standardowe).

Literatura

- Materiały do ćwiczenia: dokument na Board'zie w katalogu /Kwasnicka/Sztuczna_Inteligencja/ Calosc_ZeszytNaukowyNr1_Ostatni.pdf
- Mitchell M., *An introduction to Genetic Algorithms:* http://www.boente.eti.br/fuzzy/ebook-fuzzy-mitchell.pdf
- Arabas J. Wykłady z algorytmów ewolucyjnych http://staff.elka.pw.edu.pl/~jarabas/ksiazka.html
- Goldberg D. Algorytmy genetyczne i ich zastosowanie
- Michalewicz Z. Algorytmy genetyczne + struktury danych = programy ewolucyjne
- Michalewicz Z., Fogel D.B. Jak to rozwiązać, czyli nowoczesna heurystyka
- Strona projektu iMOPSE MS-RCPSP: http://imopse.ii.pwr.edu.pl

Opracowano: 06.03.2017