BADANIA OPERACYJNE

Skład zespołu:

Tomasz Lichoń

Tomasz Pałys

Jakub Sawicki

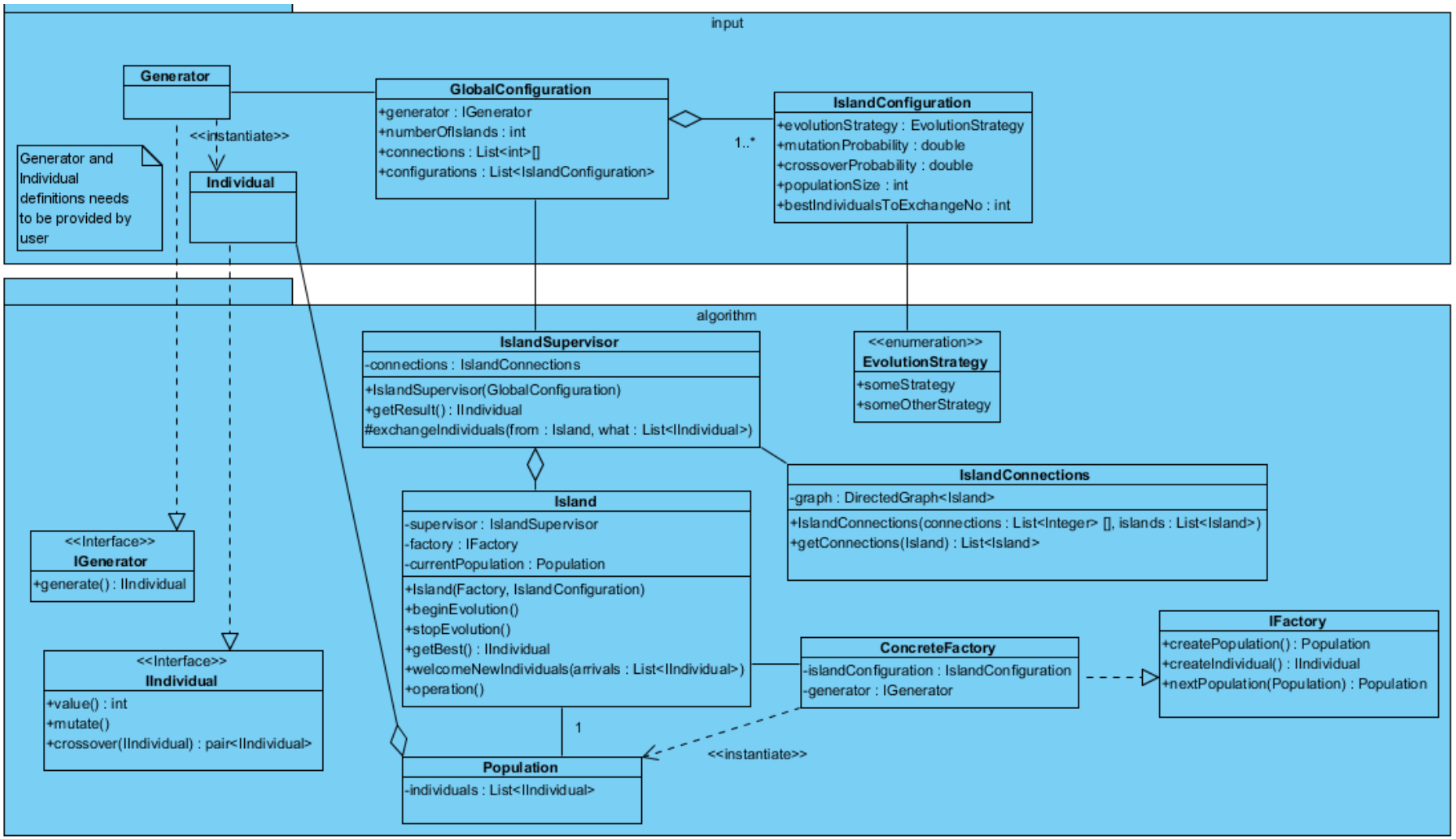
Natalia Potoczny

Mateusz Radko

Konrad Zuchowicz

Łukasz Cieśla

1. Diagram UML



2. Architektura systemu

Ogólny zarys systemu możemy zobaczyć na załączonym diagramie klas UML. Widać wyraźny podział między częścią konfiguracyjną zależną od użytkownika (input) i częścią algorytmiczną która jest parametryzowana przez konkretne konfigurację.

Input:

W celu skorzystania z systemu użytkownik musi dostarczyć klasy implementujące interfejsy IIndividual i IGenerator. Podstawą jest IIndividual w którym użytkownik powinien zdefiniować sposób reprezentacji konkretnego argumentu, metodę która pozwala wyliczyć wartość funkcji której ekstremum mamy badać, oraz dostarczyć podstawowe operacje takie jak mutacja, krzyżowanie czy zduplikowanie osobnika.

Klasy te sąbezpośrednio związane z problemem, oprócz tego wymagamy wyspecyfikowania pełnej konfiguracji działania systemu, co obejmuje:

- instancję obiektu generatora zdolnego generować losowe osobniki
- zmienną logiczną maximize, informującą czy mamy szukać minimum czy maksimum
- czas ewolucji (jak długo algorytm ma szukać rozwiązania)
- lista sąsiedztwa dla grafu połączeń między wyspami
- konfigurację dla każdej z wysp, która obejmuje:
    - Strategię ewolucyjną wyspy (którąś z tych udostępnianych przez system)
    - prawdopodobieństwo mutacji
    - prawdopodobieństwo krzyżowania
    - rozmiar populacji
    - ilość osobników wybieranych co ewolucję w celu rozmnożenia
    - ilość osobników wymienianych z innymi wyspami

Algorithm:

Sercem systemu jest IslandSupervisor, który ustala graf połączeń na podstawie konfiguracji użytkownika, tworzy poszczególne wyspy z odpowiednimi konfiguracjami, mówi kiedy mają rozpocząć i kończyć pracę, oraz pośredniczy w komunikacji między nimi.

W naszym przypadku wyspy implementowane są przy pomocy odrębnych wątków, można jednak w prosty sposób wydzielić interfejsy wykorzystane przy komunikacji poziomu slave – supervisor i przenieść je na osobne maszyny tworząc system rozproszony.

Na każdej z wysp ewolucja postępuje według tego samego, prostego algorytmu:

1. Dodaj do populacji przybywające na wyspę osobniki
2. Wyślij swoje najlepsze osobniki do supervisora w celu przekazania ich innym wyspą
3. Przeprowadź ewolucję przechodząc do następnej populacji
4. Zaktualizuj informację o najlepszym osobniku

Całość informacji o konkretnym sposobie prowadzenia ewolucji posiada supervisor który każdej z wysp przyporządkowuje konkretną fabrykę odpowiedzialną za generację nowych

populacji. Wyspy natomiast operują na takim poziomie abstrakcji, że nie są świadome istnienia różnych strategii ewolucyjnych, korzystają tylko ze znanego sobie interfejsu Ifactory.

Kluczowym z punktu widzenia całości systemu jest algorytm fabryki. Musi ona być w stanie wygenerować populację początkową, oraz na podstawie zdefiniowanej konfiguracji przeewoluować jedną populację w drugą. Trzon każdej z fabryk jest nakreślony w klasie AbstractFactory, szczególnie istotna jest metoda nextPopulation(Population parent), która definiuje schemat każdej z przeprowadanych ewolucji, jest on następujący:

1. Wybierz osobników ojców
2. Krzyżuj ojców aż uzyskasz pełną populację
3. Mutuj wybrane osobniki
4. Zwróć tak stworzone osobniki jako nową generację

Metoda selekcji zależna jest od konkretnej strategii ewolucyjnej, aktualnie wspieramy pięć strategii: Roulette, Stochastic, Tournament, LinearRanked, Truncation. Każdej z nich odpowiada osobna fabryka, która jako jednostka przekazywana wyspie kryje w sobie całą logikę związaną z czystym algorytmem ewolucyjnym.

Nowe strategie mogą być w prosty sposób dodawane poprzez zdefiniowanie konkretnej fabryki. Można sobie nawet wyobrazić sytuację w której użytkownik system w zależności od potrzeb definiuje własny sposób ewolucji dostosowany do charakterystyki problemu.

W następnym rozdziale znajdują się opisy zdefiniowanych przez nas strategii

3. Algorytmy

Selekcja ma na celu wyłonienie osobników, które będą podlegać krzyżowaniu. Zaimplementowane metody selekcji:

Metoda ruletki

Wybieramy n osobników n razy losując liczbę z zakresu od 0 do sumy wskaźników fitness dla całej populacji.

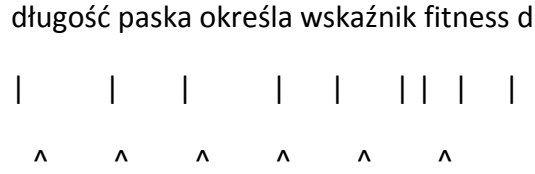
Prawdopodobieństwo, czy wybierzemy danego osobnika zależy liniowo od jego wskaźnika fitness. Może się zdarzyć, że

dominujący osobnik zostanie wybrany kilka razy.

Metoda stochastyczna

W celu zmniejszenia dominacji najlepszych osobników w wynikowych populacjach losujemy tylko jedną

liczbę, natomiast wybieramy n osobników w następujący sposób:



Losowo wybieramy pierwszy wskaźnik, a następnie akceptujemy osobniki w odpowiednich odstępach.

Dzieki temu zmniejszony jest elityzm w wynikowych populacjach.

Metoda turniejowa

N razy dokonujemy losowania dwóch osobników z populacji, a następnie porównujemy ich fitness.

Z prawdopodobieństwem równym presji środowiska (liczba od 0 do 1, praktycznie od 0.5) wybieramy lepszego z nich.

Metoda Rankingowa

Algorytm selekcji opiera się na wyznaczeniu liniowej funkcji przypisującej prawdopodobieństwo wyboru kandydata. Sortujemy populację od najlepszego do najgorszego kandydata na rozwiązanie, prawdobodonieństwo wyboru poszczególnego rozwiązania liczymy według wzoru:

$$P(x) = \frac{N + 1 - x}{N(N + 1)}$$

gdzie x to pozycja w rankingu, a N opisuje liczność zbioru kandydatów.

Następnie przedział [0,1] dzielimy według wzoru otrzymujac N przedziałów dla każdego kandydata $[0, \frac{N}{N(N+1)})$, $[\frac{N}{N(N+1)}, \frac{N}{N(N+1)} + \frac{N-1}{N(N+1)})$, … , $[\sum_{i=2}^{N} \frac{i}{N(N+1)}, 1]$.

Losujemy k-krotnie liczbę z przedziału [0,1] i do zbioru dodajemy kandydata w przedziale, którego znajdzie się randomowa liczba. (Podobnie jak w Roulette Selection)

Truncation method

Algorytm działa analogicznie do Linear Ranked Selection jednak odrzucamy od 10 do 50% procent najgorszych kandydatów.

Krzyżowanie

Osobniki są losowo dobierane w pary i wykonywane jest na nich krzyżowanie. Operacja ta dokonywana jest

na już wyselekcjonowanych osobnikach, do których na koniec całej operacji dodawana jest nowa pula ich dzieci.

Tzn. osobiniki wyselekcjonowane pozostają w populacji, a nie tylko ich dzieci.

Mutacja

Z określonym w dokumentacji prawdopodobieństwem każdy z osobników mutuje.

4. Testy

Do przetestowania naszego systemu wykorzystaliśmy bibliotekę QAPLib. Zawiera ona ponad 100 problemów QAP – quadratic assigment problem, wraz z rozwiązaniami optymalnymi – jeśli znaleziono; w przeciwnym przypadku najlepszymi do tej pory znalezionymi.

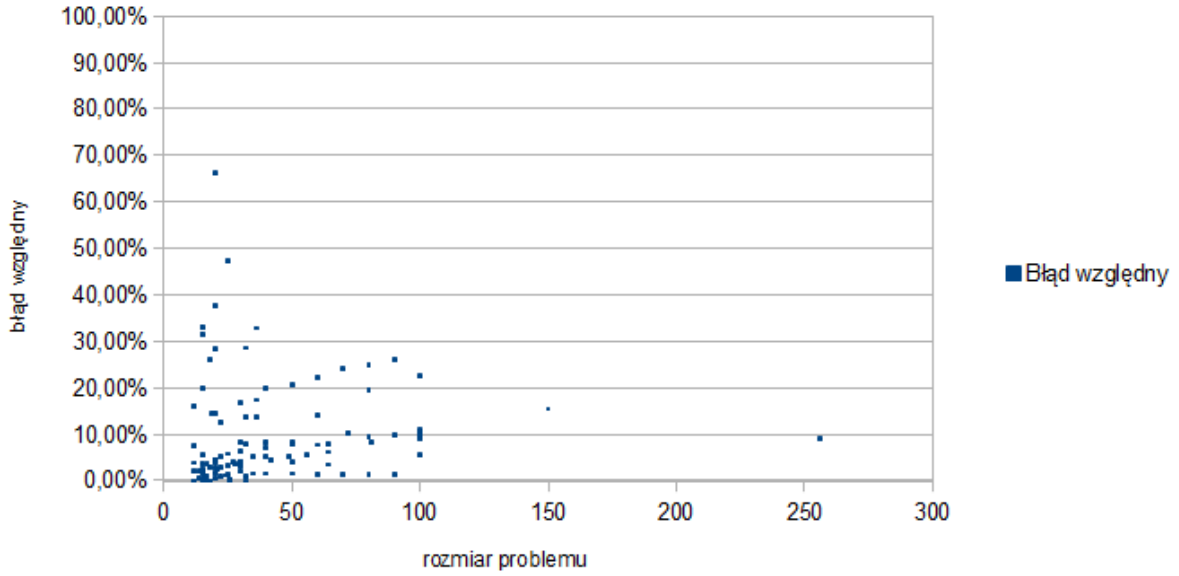
W problemie QAP mamy dane dwie macierze kwadratowe – w i d.w_{ij} oznacza przepływ z punktu i do j, natomiast d_{ij} oznacza odległość między punktami i i j. Zadanie polega na znalezieniu takiej permutacji p, że wyrażenie $\sum_{i=0}^{n} w_{ij} d_{p(i)p(j)}$ ma wartość najmniejszą.

Testy przesiewowe

Przeprowadziliśmy przesiewowe testy na wszystkich dostępnych problemach. Dla każdego problemu algorytm działał przez 60 sekund. W każdym przypadku było osiem wysp, dla testów przesiewowych było pięć wysp o różnych parametrach które wysyłają osobniki do dwóch wysp, z których jedna bardzo często mutuje osobniki a druga często dokonuje krzyżowania. One z kolei przesyłają osobniki do ostatniej wyspy.

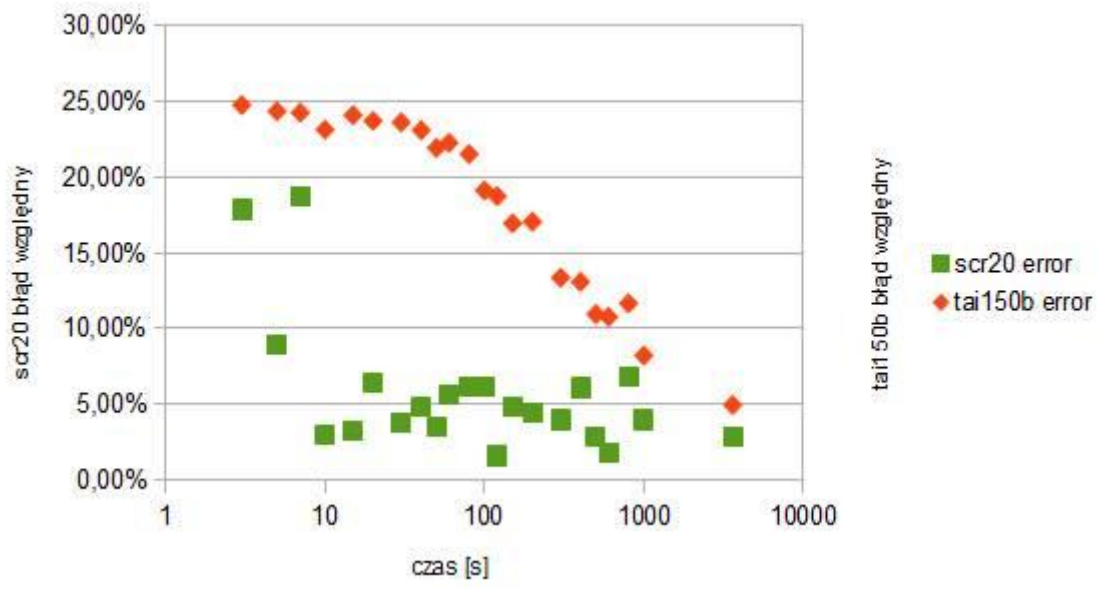
Udało się uzyskać wiele rozwiązań optymalnych (np. esc16, esc32, hadx) jak i suboptymalnych z różnicą rzędu procenta (np. bur26, lipxa, nugx, też mniejsze taix). Dla

wielu większych problemów również osiągnęliśmy akceptowalne wyniki (np. tho150 - 16%, wil100 - 6%, tai100a - 11%, tai100b - 23%).



Jak widać nie ma wyraźnej ogólnej zależności pomiędzy wielkością problemu a możliwością jego dokładnego rozwiązania. Takie zależności widać dla poszczególnych klas problemów, ale również niewielkie problemy mogą okazać się trudne do optymalizacji.

Zależności czasowe



Dla niewielkiego problemu scr20 błąd oscyluje w granicach kilku procent i nie widać żadnej zależności między nim a czasem uruchomienia algorytmu. Natomiast dla dużego problemu

tai150b błąd wyraźnie maleje wraz ze wzrostem czasu obliczeń. Ostatecznie dając akceptowalne wyniki.

Operator selekcji

Dobór operatora selekcji ma znaczący wpływ na jakość otrzymanego rozwiązania. Dla scr20 najefektywniejszy okazał się operator linear rank natomiast dla tai150b najlepiej sprawdził się operator turniejowy.

selekcja	tai150b	scr20
roulette	22,89%	2,82%
stochastic	10,49%	3,11%
tournament	9,12%	7,33%
truncation	11,88%	1,85%
linear rank	12,42%	0,88%

Wartości w tabeli to wartości względnego błędu.

Topologia

Testy przeprowadziliśmy dla różnych topologii. Pod uwagę wzięliśmy topologię kliki, cyklu i słońca (jedna wyspa centralna).

topologia	tai150b	scr20
klika	7,84%	7,67%
słońce	10,91%	1,63%
cykl	9,91%	2,92%

Jak widać wyniki nie są jednoznaczne i dla każdego zagadnienia lepsza może być inna topologia.

W dużym problemie lepiej sprawdziła się topologia kliki, natomiast dla niewielkiego scr20 zdecydowanie lepsze wyniki uzyskaliśmy przy użyciu topologii słońca niż kliki.