Inteligencja obliczeniowa - Algorytm genetyczny cz.2

Grzegorz Madejski

Część 1

Część 1: Trudne problemy, niebinarne chromosomy, egzotyczne krzyżowania

Labirynt

Problemy w wersji decyzyjnej odpowiadają TAK/NIE, a w optymalizacyjnej wyszukują najlepsze rozwiązania wg jakiegoś parametru.

- Parametr: długość drogi + minimalizacja

Labirynt (optymalizacyjny)

Podaj *jak najkrótszą* drogę dojścia do wyjścia (o ile istnieje).

- Parametr: odległość od wyjścia (sensowna wersja, gdy podejrzewamy, że rozwiązanie nie istnieje) + minimalizacja

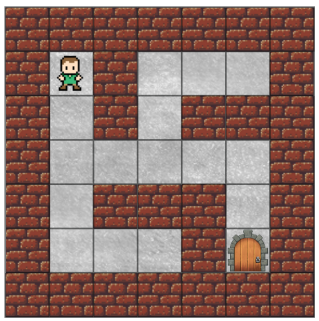
Labirynt (optymalizacyjny)

Podaj drogę dla człowieka, tak aby znalazł się *jak najbliżej* wyjścia.

Labirynt

Zastanów się, jak rozwiązać problem labiryntu w wersji decyzyjnej (szukamy drogi o maks. 10 krokach).

- Jak zakodować rozwiązanie jako chromosom?
- Jak oceniać chromosomy za pomocą funkcji fitness? Jakie oceny ona wystawia?
- Zastanów się nad różnymi sposobami. Być może problem można rozwiązać na wiele sposobów?



Czas do namysłu: kilka minut!

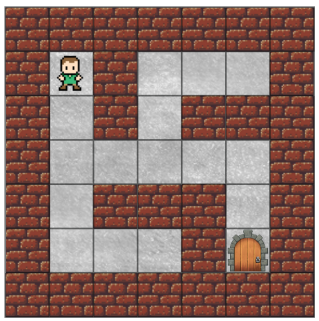
Labirynt

Sposób 1: rozwiązania jako sekwencje ruchów

Labirynt

Sposób 1: rozwiązania jako sekwencje ruchów

- *Chromosom* koduje 10 ruchów (bo tyle jest dozwolonych) dla człowieka.
- Każdy ruch ma swój znak (np. lewo $= 0$, góra $= 1$, prawo $= 2$, dół $= 3$).
- Przykładowy chromosom $X = [3, 3, 2, 2, 2, 2, 1, 2, 2, 2]$
- Gdzie dojdzie chromosom w labiryncie?



Labirynt

Sposób 1: fitness jako odległość od wyjścia

Labirynt

Sposób 1: fitness jako odległość od wyjścia

- *Fitness* bierze ruchy z chromosomu, wykonuje i patrzy gdzie się zatrzymamy. Uwaga: są dwa podejścia!

Labirynt

Sposób 1: fitness jako odległość od wyjścia

- *Fitness* bierze ruchy z chromosomu, wykonuje i patrzy gdzie się zatrzymamy. Uwaga: są dwa podejścia!
  - W przypadku ruchu na ścianę, przerywamy symulację. Nie rozpatrujemy dalszych ruchów. Takie podejście jest surowe, ale daje czyste rozwiązania.
  - W przypadku ruchu na ścianę, ignorujemy ten ruch, i przechodzimy do następnego. Takie podejście jest luźnniejsze, ale dostarczy rozwiązanie zanieczyszczone uderzeniami w ścianę (trzeba je oczyścić).

Labirynt

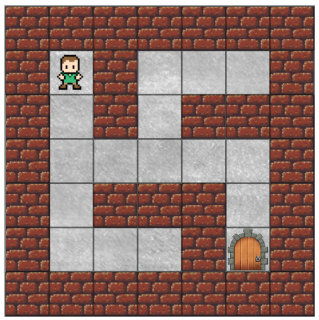
Sposób 1: fitness jako odległość od wyjścia

- *Fitness* bierze ruchy z chromosomu, wykonuje i patrzy gdzie się zatrzymamy. Uwaga: są dwa podejścia!
  - W przypadku ruchu na ścianę, przerywamy symulację. Nie rozpatrujemy dalszych ruchów. Takie podejście jest surowe, ale daje czyste rozwiązania.
  - W przypadku ruchu na ścianę, ignorujemy ten ruch, i przechodzimy do następnego. Takie podejście jest luźnniejsze, ale dostarczy rozwiązanie zanieczyszczone uderzeniami w ścianę (trzeba je oczyścić).
- *Fitness* to odległość końca symulacji chromosomu od wyjścia. Im dalej zajdzie tym lepiej. Można to zrobić metryką taksówkową:

$$d(stop, exit) = |x_{stop} - x_{exit}| + |y_{stop} - y_{exit}|$$

Labirynt

Sposób 1: fitness jako odległość od wyjścia



- Policz fitness w wersji "hard" (uderzenie w ścianę powoduje przerwanie symulacji) dla chromosomu $X = [3, 3, 2, 2, 2, 2, 1, 2, 2, 2]$ i $Y = [1, 3, 3, 0, 2, 1, 3, 1, 3, 1]$ korzystając ze wzoru:

$$d(stop, exit) = |x_{stop} - x_{exit}| + |y_{stop} - y_{exit}|$$

- Policz fitness dla powyższych chromosomów w wersji "light" (uderzenie w ścianę jest ignorowane, przechodzimy do kolejnego ruchu)

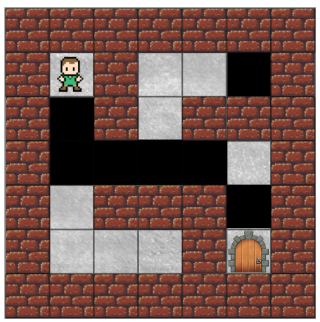
Labirynt

Sposób 2: chromosomy to zamalowanie pustych pól labiryntu

- W labiryncie jest 15 pustych pól, które mogą być częścią ścieżki do wyjścia. Numerujemy je rzędami i tworzymy chromosom binarny. Jeśli pole jest cześcią ścieżki, to 1, jeśli nie, to 0. Przykład:

$$X = [0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0]$$

Pola wskazane przez X jedynkami tworzą ścieżkę:



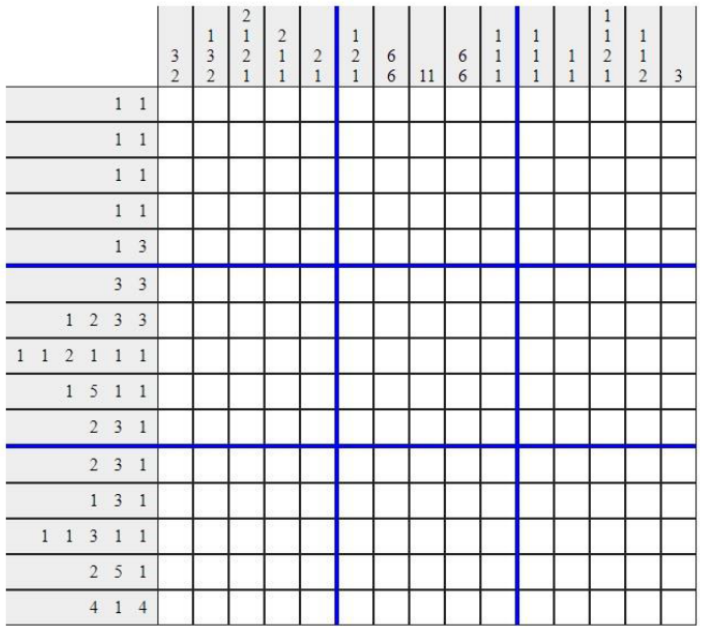
- Tworzenie funckji fitness jest skomplikowane. Musi ona premiować ściezki, które: są spójne, łączą człowieka z wyjściem, nie maja rozgałęzień, nie przekraczają limitu pól.

Labirynt

Dodatkowe pytanie do zastanowienia się:
Jak zmienić funckję fitness ze sposobu 1, aby premiowała najkrótsze ścieżki? (problem optymalizacyjny)

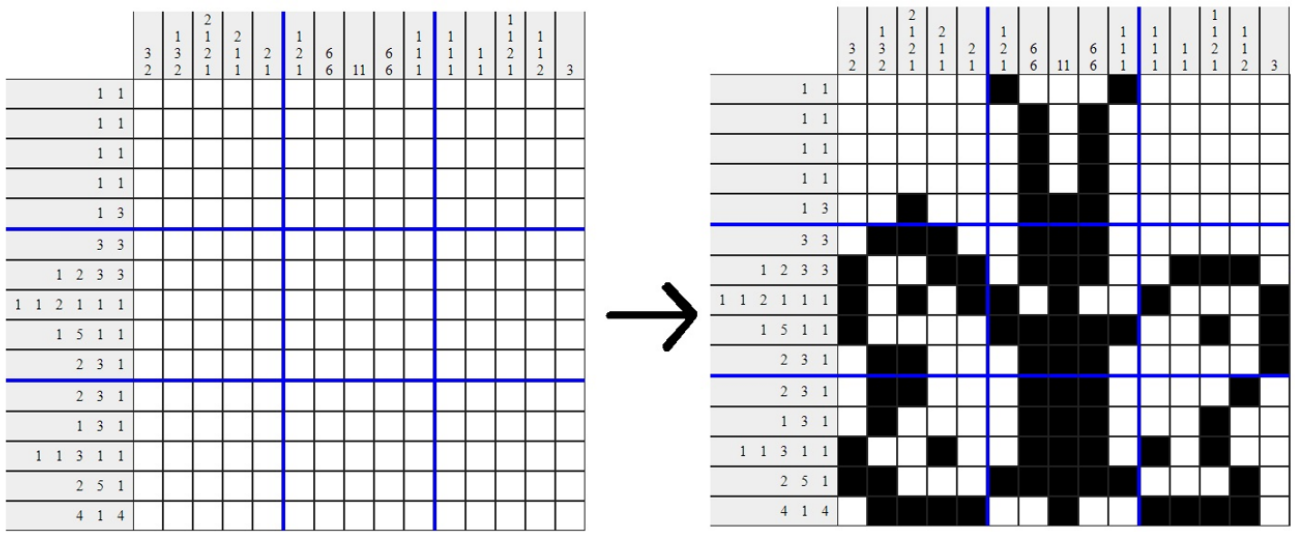
Nonogram

Zamaluj pola na obrazku na czarno, tak aby reguły na brzega obrazka dobrze opisywały liczbę czarnych pól w danym rzędzie i kolumnie. Przykładowo, jeśli przy wierszu stoi "2 3 1", to znaczy, że w danym wierszu jest ciąg dwóch zamalowanych kratek, przerwa (przynajmniej jedna biała kratka), trzy zamalowane kratki, przerwa, jedna zamalowana kratka. Umiejscowienie zamalowanych ciągów nie jest wskazane.

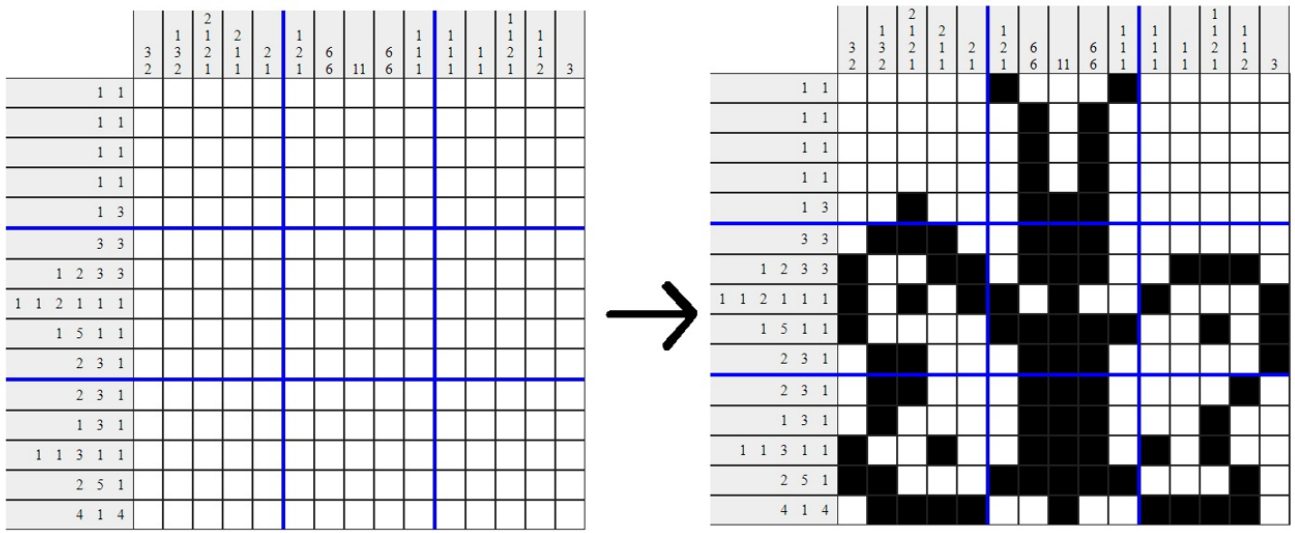


Nonogram

Zamaluj pola na obrazku na czarno, tak aby reguły na brzega obrazka dobrze opisywały liczbę czarnych pól w danym rzędzie i kolumnie. Przykładowo, jeśli przy wierszu stoi "2 3 1", to znaczy, że w danym wierszu jest ciąg dwóch zamalowanych kratek, przerwa (przynajmniej jedna biała kratka), trzy zamalowane kratki, przerwa, jedna zamalowana kratka. Umiejscowienie zamalowanych ciągów nie jest wskazane.

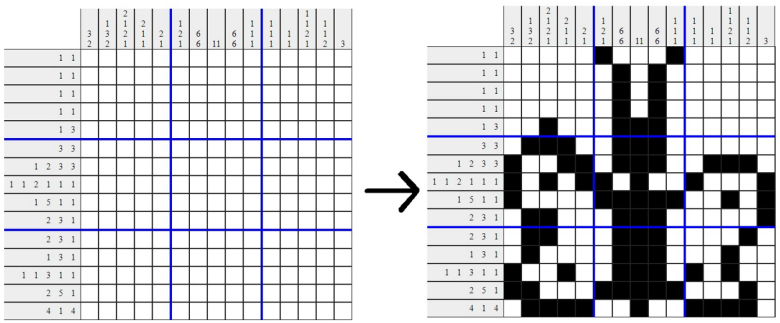


Nonogram



Jak zakodować rozwiązanie jako chromosom? Jaką ma długość?
Jak stworzyć sensowną funkcję fitness?

Nonogram

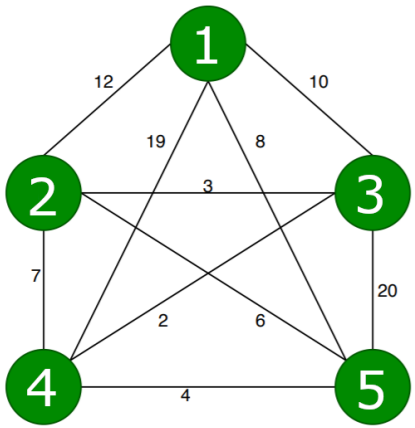


Pomysły:

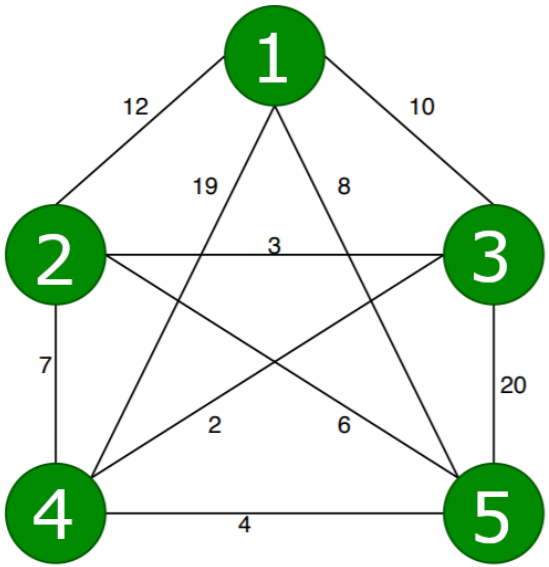
- Chromosom to cały obszar łamigłówki (255 genów). Geny: 0 = pole niezamalowane, 1 = pole zamalowane.
- Chromosom to współrzędne 46 bloków z wierszy. Geny: liczby od 1 do 15 wskazujące na umiejscowienie danego bloku w wierszu.
- Fitness = jak bardzo reguły są spełnione. Do zastanowienia...

Problem komiwojażera

Problem komiwojażera (ang. Travelling Salesman Problem, TSP) to problem związany z grafami nieskierowanym (istnieje też wersja dla skierowanych), w których wierzchołki reprezentują miasta, a krawędzie z wagami reprezentują ulice z ich długościami. W problemie komiwojażera chcemy podać jak najkrótszą ścieżkę, która przejdzie przez wszystkie wierzchołki jeden raz, a następnie wróci do wierzchołka początkowego (najkrótszy cykl Hamiltona).

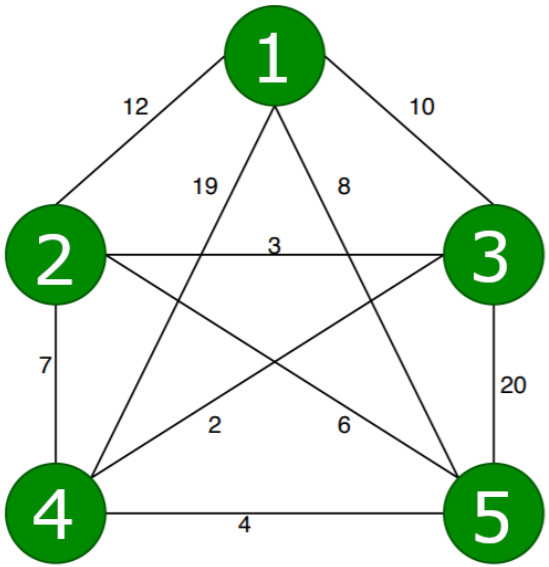


Problem komiwojażera



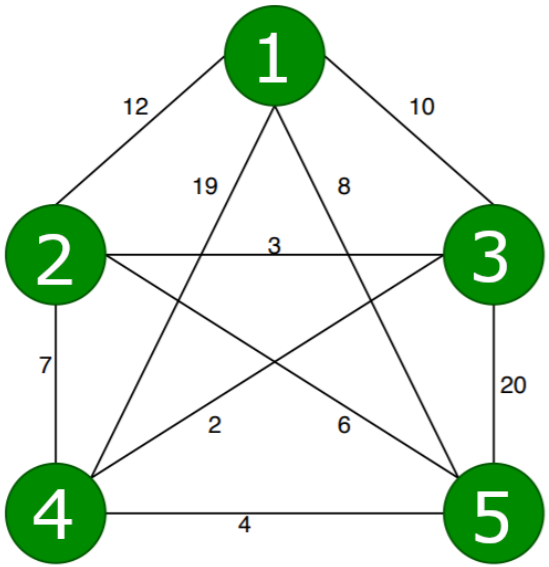
- Jak kodować chromosomy?

Problem komiwojażera



- Jak kodować chromosomy?
- Wypisanie numerów wierzchołków w kolejności odwiedzania, np. $x = [1, 2, 4, 5, 3]$

Problem komiwojażera



- Jak kodować chromosomy?
- Wypisanie numerów wierzchołków w kolejności odwiedzania, np. $x = [1, 2, 4, 5, 3]$
- Fitness = długość trasy. Przykład $fitness(x) = 12 + 7 + 4 + 20 + 10 = 53$.

Problem komiwojażera - krzyżowanie

- Pojawia się problem: jak krzyżować takie chromosomy?
- Weźmy $x = [1, 2, 4, 5, 3]$ i $y = [1, 3, 2, 5, 4]$
- Rozetnijmy je w proporcjach 40%/60% i skrzyżujmy.
- Otrzymamy potomków: $a = [1, 2, 2, 5, 4]$, $b = [1, 3, 4, 5, 3]$
- Wierzchołki powtarzają się w chromosomach - to nielegalne!

Problem komiwojażera - krzyżowanie

- Na szczęście istnieje *krzyżowanie z częściowym mapowaniem* (ang. partially mapped crossover, PMX), które takich felernych potomków naprawia.
- Przykład. Rodzice chcą wymienić się 3, 4 i 5 genem.

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

- Następuje krzyżowanie. Powstają proto-dzieci:

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

- Jak widać dzieci mają nielegalnie powtarzające się geny.

Problem komiwojażera - krzyżowanie

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

- Do naprawy potomków tworzymy mapowanie na podstawie wymienianych genów. Mapowanie pozwala zamieniać następujące elementy $2 \leftrightarrow 5$, $3 \leftrightarrow 6$ i $6 \leftrightarrow 4$.
- Korzystając z tego mapowania, usuwamy powtarzające się geny poza obszarem wymiany. Zaznaczmy je.

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

Problem komiwojażera - krzyżowanie

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

Mapowanie: $2 \leftrightarrow 5$, $3 \leftrightarrow 6$ i $6 \leftrightarrow 4$

- W pierwszym dziecku zamieniamy 2 na 5. Następnie zamieniamy 3 na 6 (to nadal nie działa), więc ponownie 6 na 4.
- W drugim dziecku zamieniamy 4 na 6 (nadal nie działa), więc 6 na 3. Nastepnie 5 na 2.
- Otrzymujemy:

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

- Otrzymaliśmy potomków o legalnej strukturze.

Problem komiwojażera - mutacja

- Mutacja zamieniające losowe geny na inne też nie jest tutaj dobra.
  Przed mutacją: $x = [1, 2, 3, 4, 5, 6]$.
  Po mutacji: $x = [1, 6, 3, 4, 5, 6]$.
- Możemy zastosować inne operatory mutacji np. zamianę genów (swap).
  Przed mutacją: $x = [1, 2, 3, 4, 5, 6]$.
  Po mutacji: $x = [1, 6, 3, 4, 5, 2]$.
- Odwrócenie kolejności fragmentu chrosomomu, np. genów, 2,3,4 (inversion).
  Przed mutacją: $x = [1, 2, 3, 4, 5, 6]$.
  Po mutacji: $x = [1, 4, 3, 2, 5, 6]$.
- Przeniesienie części genów np. 2,3,4 w inne miejsce (displacement).
  Przed mutacją: $x = [1, 2, 3, 4, 5, 6]$.
  Po mutacji: $x = [1, 5, 2, 3, 4, 6]$.

Problem komiwojażera + pyGAD

- W paczce pygad da się zrealizować rozwiązywanie TSP algorytmem genetycznym.
- Można definiować własne krzyżowanie i mutacje: `https://pygad.readthedocs.io/en/latest/README_pygad_ReadTheDocs.html#user-defined-crossover-mutation-and-parent-selection-operat`
- Krzyżowanie PMX nie jest dostępne w pygad (należy napisać samodzielnie), ale mutacje typu swap czy inversion są dostępne.
- Zamiast pisać PMX można w pygad zablokować zduplikowane geny (allow duplicate genes = False). Funkcja ta jednak nie jest mądra, zgodnie z dokumentacją działa randomowo: *Solves the duplicates in a solution by randomly selecting new values for the duplicating genes*.

Część 2

Część 2: Optymalizacja algorytmu genetycznego

Optymalizacja algorytmu genetycznego

- Algorytm genetyczny szuka *globalnego* maksimum funkcji fitness, ale może utknąć w *lokalnym* maksimum. (maksimum).
- Powodem tego może być nieoptymalnie skonfigurowany algorytm genetyczny. Sprawdź:
  - Czy kodowanie chromosomu jest dobre. Może istnieją inne podejścia?
  - Czy funkcja fitness działa poprawnie? Może jej ocenianie nie prowadzi ewolucji w dobry sposób ku najlepszemu rozwiązaniu?
  - Czy populacja jest odpowiednio duża. Dla trudniejszych problemów bierze się często kilkaset, a nawet powyżej tysiąca osobników.
  - Czy selekcja, krzyżowanie, mutacja są dobrze ustawione. Może warto je pozmieniać?

Optymalizacja algorytmu genetycznego

- Bywa jednak, że mimo wielu prób i optymalizacji algorytm genetyczny nadal nie znajduje optymalnego rozwiązania.
- Algorytm może utknąć w maksimum lokalnym (ślepy zaułek) i może nie chcieć z niego wyjść.
- Warto wówczas zadbać o większą różnorodność genetyczną chromosomów. Chcemy, zeby algorytm genetyczny szukał rozwiązań w innych obszarach.
- Być może warto zrezygnować z przenoszenia rodziców do pokolenia dzieci, ale mimo wszystko zapamiętywać najlepsze rozwiązanie (na boku). Innymi słowy: usuwamy elityzm.
- Inny sposób: wprowadzić karę za niszę (niche penalty) tzn jeśli kilka chromosmów jest do siebie bardzo podobnych, to dostają karę do fitness.
- Jeszcze innym sposobem na wymianę "zasiedziałych" chromosomów jest wymiana cześci populacji na losowe chromsomomy (random immigrants).

Optymalizacja algorytmu genetycznego

Dość skuteczną strategią jest zastosowanie *modelu wyspowego* (ang. island model genetic algorithms), czyli algorytmu genetycznego z wieloma populacjami. Można sobie wyobrazić kilka wysp, z których każda ma swoją populację. Na każdej działa algorytm genetyczny szukając najlepszego rozwiązania (tutaj szczególnie przydatne jest zrównoleglanie obliczeń, 1 wyspa $=$ 1 wątek). Przez większość czasu wyspy nie mają ze sobą komunikacji, ale co parę pokoleń parę procent osobników może przepłynąć z wyspy na wyspę, co nazywamy migracją. Taki model, z uwagi na większe rozproszenie, ma mniejszą szansę na wpadnięcie w lokalne maksima.

Optymalizacja algorytmu genetycznego

Model wyspowy algorytmu genetycznego wymaga eksperymentów z konfiguracją. Nalezy zwrócić uwagę na poniższe aspekty:

- Jak często wystepuje migracja? Może 5 pokoleń?
- Ile osóbników migruje? Czy kilka procent to dobry wybór?
- Jakie osobniki migrują? Czy najsilniejsze, czy losowe?
- Na jakie wyspy migrują osobniki? Wybieramy topologię połączeń.