Algorytmy i modele inspirowane biologicznie

3. Warianty i specjalizacje algorytmów ewolucyjnych

Maciej Komosiński

Fundusze Europejskie
Polska Cyfrowa

Rzeczpospolita Polska

Unia Europejska
Europejski Fundusz Rozwoju Regionalnego

Ang. *messy genetic algorithm*.

Podejście „nieporządne" (niechlujne) ma na celu polepszenie własności algorytmu genetycznego poprzez skuteczniejsze konstruowanie, wykorzystanie i przekształcanie schematów. Taki sam cel ma realizować operator inwersji, który omówimy w kolejnej prezentacji poświęconej mechanizmom inspirowanym naturą. Nieporządny algorytm genetyczny [Gol+93][Mic96, str. 122] wykorzystuje szczególną reprezentację osobników: genotypy są zmiennej długości, złożone z par (*etykieta*, *wartość*). Etykieta to opis znaczenia genu – podobnie jak w operatorze inwersji, etykieta może być pierwotnym numerem genu.

Dozwolone są genotypy niekompletne (niedospecyfikowane), tzn. nie określające wartości wszystkich genów. Genotyp może zawierać również geny nadmiarowe, a nawet sprzeczne.

Nieporządny algorytm genetyczny: działanie

Używa się trzech operatorów: cięcia, łączenia i mutacji. Operator cięcia rozcina z pewnym prawdopodobieństwem, w losowo wybranym miejscu, łańcuch bitów. Operator łączenia skleja z pewnym prawdopodobieństwem dwa genotypy. Mutacja jest identyczna z mutacją prostą.

Stosowana jest selekcja turniejowa. Proces ewolucji składa się z dwóch (potencjalnie wielokrotnie wykonywanych) faz: wyboru bloków budujących i stosowania operatorów. Liczebność populacji jest zmienna w trakcie działania algorytmu.

Nadmiarowość informacji w genotypie można prosto rozwiązać wybierając pierwszą napotkaną wartość danego genu w genotypie, ale istnieją też inne metody – np. uśrednianie wszystkich wartości genu albo stosowanie pewnego rodzaju głosowania, którą wybrać. Niedoprecyzowane genotypy, o ile nie są akceptowalne w danym problemie optymalizacji, rozwiązuje się uzupełniając brakujące geny najlepszymi znanymi wartościami danego genu z wcześniejszej fazy algorytmu.

Nieporządne algorytmy genetyczne w problemach zwodniczych działały kilkakrotnie lepiej od klasycznych algorytmów genetycznych z krzyżowaniem punktowym.

Wydajna optymalizacja – hipoteza cegiełek powraca

Wydajna optymalizacja – hipoteza cegiełek powraca

Epistaza: interakcja genów (zmiennych, cech) wpływa na jakość rozwiązania → łączymy (wiążemy) je w grupy (*linkage*).

Dyskusja: czy można w jakiś sposób „wykryć" epistazę elementów rozwiązania?

Wydajna optymalizacja – hipoteza cegiełek powraca

Epistaza: interakcja genów (zmiennych, cech) wpływa na jakość rozwiązania → łączymy (wiążemy) je w grupy (*linkage*).

Dyskusja: czy można w jakiś sposób „wykryć" epistazę elementów rozwiązania?

Ang. *linkage* w optymalizacji: nie należy mylić z biologicznym genetic linkage.

Wykrywanie zależności: techniki empiryczne i statystyczne

Wizja automatycznej dekompozycji problemu optymalizacji jest bardzo atrakcyjna – pozostaje kwestia metod i ich wydajności, dlatego to zagadnienie jest aktywnie badane.

Metody wykrywania zależności między genami dzieli się czasem na

- Empiryczne (np. DLED: Direct Linkage Empirical Discovery): próbkują i oceniają pełne otoczenie konkretnego osobnika, zatem uzyskują kompletną informację o (nie)zależności w jego lokalnym sąsiedztwie i dla jego konkretnego zestawu wartości genów.
- Statystyczne (np. H-GA albo rodzina metod GOMEA): statystycznie szacują niezależność genów bazując na osobnikach w populacji oraz ich ocenach.

Bazując na tej informacji dowiadujemy się, czy i jak można dokonać dekompozycji problemu – co może mieć miejsce już w trakcie działania algorytmu [PKF21, Rozdział 5]. Algorytm może dzięki temu odpowiednio zarządzać podpopulacjami optymalizującymi potencjalnie niezależne podproblemy, dostosowywać operator krzyżowania, mutacji, itp.

Szacowanie epistazy i dekompozycja – technika DLED

Załóżmy, że mamy osobnika, którego genotyp składa się z co najwyżej pięciu elementów. Istnienie każdego elementu reprezentujemy jednym bitem (np. cztery z pięciu elementów to np. osobnik 10111).

Współzależności między genami uwypuklają się w optimach lokalnych. Dlatego jeśli nam na tym zależy, można poddawać analizie osobniki będące optimami lokalnymi – można je np. wcześniej zoptymalizować metodą Greedy (z losową kolejnością sąsiadów–genów) zamieniając pojedyncze $1 \to 0$ i $0 \to 1$.

Szacowanie epistazy i dekompozycja – technika DLED

Załóżmy, że mamy osobnika, którego genotyp składa się z co najwyżej pięciu elementów. Istnienie każdego elementu reprezentujemy jednym bitem (np. cztery z pięciu elementów to np. osobnik 10111).

Współzależności między genami uwypuklają się w optimach lokalnych. Dlatego jeśli nam na tym zależy, można poddawać analizie osobniki będące optimami lokalnymi – można je np. wcześniej zoptymalizować metodą Greedy (z losową kolejnością sąsiadów–genów) zamieniając pojedyncze $1 \to 0$ i $0 \to 1$.

Na analizowanym genotypie osobnika wykonujemy dekompozycję typu DLED [PKF21]:

1. Dla każdego genu A wprowadzamy perturbację (zmieniamy wartość na odwrotną).
2. Sprawdzamy wszystkie pozostałe geny, jak dla genu A po perturbacji wartość innego genu B wpływa na fitness jeśli pozostaje niezmieniona oraz jeśli będzie zmieniona.
3. Decyzja o zależności jest binarna i wynika ze spełnienia warunku/warunków z kolejnego slajdu.

Technika DLED: warunki zależności genów

Warunki z artykułu [PTK23]:

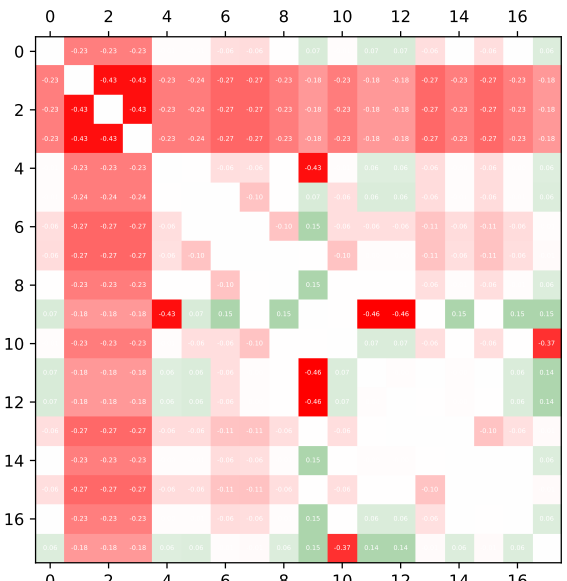
The situation changed when the Direct Empirical Linkage Discovery (DLED) was proposed [17]. To check the dependency between genes g, h, DLED requires computing $f(\boldsymbol{x})$, $f(\boldsymbol{x}, g)$, $f(\boldsymbol{x}, h)$, $f(\boldsymbol{x}, g, h)$ values, where $(\boldsymbol{x}, g)$ and $(\boldsymbol{x}, h)$ are the genotypes of individual $\boldsymbol{x}$ with genes g and h flipped, respectively. Finally, $f(\boldsymbol{x}, g, h)$ is the genotype of $\boldsymbol{x}$ with both genes flipped. In DLED, genes g and h are considered dependent if at least one of the conditions holds:

C1. $f(\mathbf{x}) < f(\mathbf{x}, g) \;\;\&\;\; f(\mathbf{x}, h) \geq f(\mathbf{x}, g, h)$
C2. $f(\mathbf{x}) = f(\mathbf{x}, g) \;\;\&\;\; f(\mathbf{x}, h) \neq f(\mathbf{x}, g, h)$
C3. $f(\mathbf{x}) > f(\mathbf{x}, g) \;\;\&\;\; f(\mathbf{x}, h) \leq f(\mathbf{x}, g, h)$
C4. $f(\mathbf{x}) < f(\mathbf{x}, h) \;\;\&\;\; f(\mathbf{x}, g) \geq f(\mathbf{x}, g, h)$
C5. $f(\mathbf{x}) = f(\mathbf{x}, h) \;\;\&\;\; f(\mathbf{x}, g) \neq f(\mathbf{x}, g, h)$
C6. $f(\mathbf{x}) > f(\mathbf{x}, h) \;\;\&\;\; f(\mathbf{x}, g) \leq f(\mathbf{x}, g, h)$

The above conditions can be interpreted as the following statement. If the modification of one gene changes the fitness relations for the values of the other gene, then these two genes are dependent. DLED is an ELL technique proven to report only the direct dependencies.

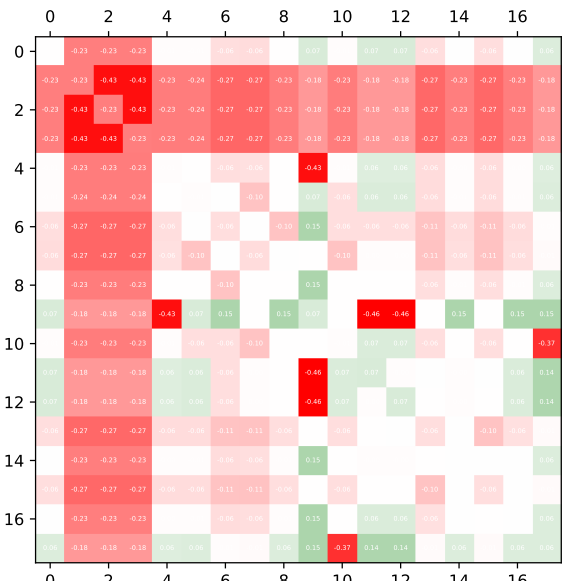
Szacowanie epistazy i dekompozycja – konkretny przykład

Zmiana w wartości fitness po wyłączeniu par genów w przykładowym osobniku



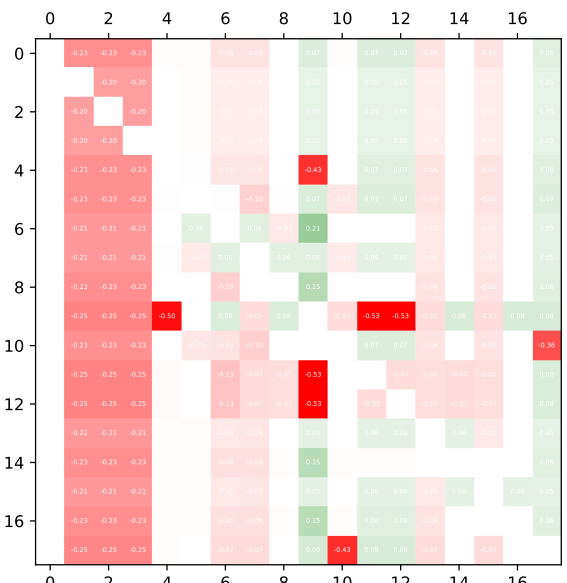
Szacowanie epistazy i dekompozycja – konkretny przykład

Zmiana w wartości fitness po wyłączeniu par genów; na przekątnej efekt wyłączenia jednego genu



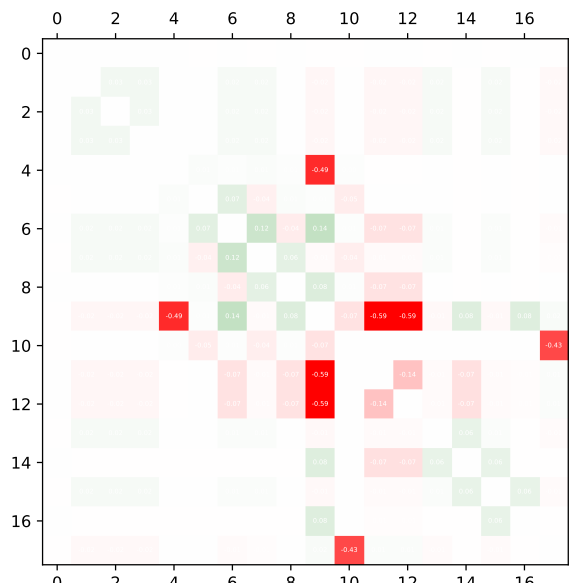
Szacowanie epistazy i dekompozycja – konkretny przykład

Odjęcie wartości na przekątnej od wierszy: jak interpretować efekt?



Szacowanie epistazy i dekompozycja – konkretny przykład

Addytywność wpływu lub jej brak

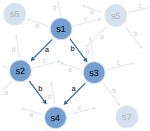


Szacowanie epistazy i dekompozycja

Co powinno być wierszami i kolumnami (w tym przykładzie to było „wyłączanie genów", ale czy o to zawsze chodzi?)

Szacowanie epistazy i dekompozycja

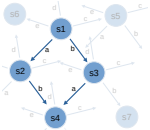
Co powinno być wierszami i kolumnami (w tym przykładzie to było „wyłączanie genów", ale czy o to zawsze chodzi?)



Jak wykorzystać te informacje w algorytmie podczas optymalizacji?

Szacowanie epistazy i dekompozycja

Co powinno być wierszami i kolumnami (w tym przykładzie to było „wyłączanie genów", ale czy o to zawsze chodzi?)

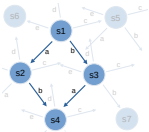


Jak wykorzystać te informacje w algorytmie podczas optymalizacji?

- optymalizować niezależne podzbiory genów osobno → zmniejszenie złożoności obliczeniowej
- zaprojektować krzyżowanie i mutację tak, aby zachowywały korzystne epistatyczne interakcje genów → traktować współpracujące epistatycznie grupy genów jako integralne zespoły → ochrona i skuteczna propagacja „cegiełek" budujących dobre rozwiązania [GT12; TB13]
- przykład: operator *Optimal Mixing* (OM) w algorytmie GOMEA. Wybiera z populacji rodzica i donora, a następnie tworzy potomka przenosząc z donora do rodzica allele współzależnych (współdziałających) genów. Akceptuje potomka tylko gdy jest on co najmniej tak dobry, jak rodzic – szczegóły dalej.

Szacowanie epistazy i dekompozycja

Co powinno być wierszami i kolumnami (w tym przykładzie to było „wyłączanie genów", ale czy o to zawsze chodzi?)



Jak wykorzystać te informacje w algorytmie podczas optymalizacji?

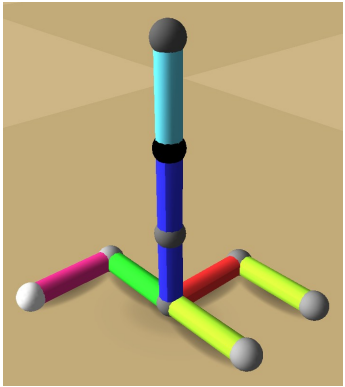
- optymalizować niezależne podzbiory genów osobno → zmniejszenie złożoności obliczeniowej
- zaprojektować krzyżowanie i mutację tak, aby zachowywały korzystne epistatyczne interakcje genów → traktować współpracujące epistatycznie grupy genów jako integralne zespoły → ochrona i skuteczna propagacja „cegiełek" budujących dobre rozwiązania [GT12; TB13]
- przykład: operator *Optimal Mixing* (OM) w algorytmie GOMEA. Wybiera z populacji rodzica i donora, a następnie tworzy potomka przenosząc z donora do rodzica allele współzależnych (współdziałających) genów. Akceptuje potomka tylko gdy jest on co najmniej tak dobry, jak rodzic – szczegóły dalej.

Jak zastosować analogiczne podejście do wykrycia współzależności trzeciego rzędu? (między trójkami genów?)

Szacowanie epistazy i dekompozycja – konkretny przykład

Bohater analizy: genotyp, fenotyp i fitness

Genotyp: /*9*/UDDDLFBFBRFBBFBFBR



Fitness: wysokość środka ciężkości konstrukcji
(0.47 dla oryginalnego genotypu – dobrego, ale nie lokalnie optymalnego).

Hierarchiczny algorytm genetyczny

Podobnie jak w przypadku nieporządnych algorytmów genetycznych, motywacją do rozwoju H-GA była chęć automatycznego odkrywania stopnia współzależności elementów rozwiązania w celu dekompozycji problemu. Poprzez próbkowanie specjalnie skonstruowanych rozwiązań można z pewnym prawdopodobieństwem określić zależność lub niezależność genów i grup genów, a następnie dla wykrytych niezależnych grup (modułów) prowadzić niezależną optymalizację [JTW04].

Aby zbadać w pełni niezależność dwóch genów od reszty rozwiązania, należałoby wygenerować zbiór rozwiązań, w których wszystkie możliwe pary wartości tych dwóch genów są otoczone wszystkimi możliwymi wartościami pozostałych genów (stanowiących „kontekst"). Następnie wszystkie te rozwiązania należałoby ocenić i wyznaczyć zależność między wartościami genów a wartością funkcji celu. Byłoby to bardzo kosztowne obliczeniowo, a przecież to byłby tylko test dla jednej pary genów! Dlatego też stosuje się próbkowanie, które umożliwia oszacowanie potencjalnej niezależności dla wszystkich podzbiorów genów w rozwiązaniu (por. GOMEA: Gene-pool Optimal Mixing EA [Thi18]).

Optymalne łączenie genów: GOMEA [Thi18; Dus+24]

Zmodyfikujmy AE tak, aby stał się mniej chaotyczny, bardziej systematyczny i celowy.

Optymalne łączenie genów: GOMEA [Thi18; Dus+24]

Zmodyfikujmy AE tak, aby stał się mniej chaotyczny, bardziej systematyczny i celowy.

(*) Dla uproszczenia (w przeciwieństwie do wcześniej omawianego problemu), rozważmy problem ze stałą liczbą genów, które mają ustalone role, np. $f(x_1, x_2, ..., x_n)$.

1. Wygeneruj losowo początkową populację rozwiązań.

Optymalne łączenie genów: GOMEA [Thi18; Dus+24]

Zmodyfikujmy AE tak, aby stał się mniej chaotyczny, bardziej systematyczny i celowy.

(*) Dla uproszczenia (w przeciwieństwie do wcześniej omawianego problemu), rozważmy problem ze stałą liczbą genów, które mają ustalone role, np. $f(x_1, x_2, ..., x_n)$.

1. Wygeneruj losowo początkową populację rozwiązań.
2. Wyucz się powiązań (*linkage*): skonstruuj model powiązań zawierający informacje o podzbiorach zmiennych, które mogą być epistatyczne (współzależne). Odkryj wzajemne zależności wyznaczając miary informacji wzajemnej.

Optymalne łączenie genów: GOMEA [Thi18; Dus+24]

Zmodyfikujmy AE tak, aby stał się mniej chaotyczny, bardziej systematyczny i celowy.

(*) Dla uproszczenia (w przeciwieństwie do wcześniej omawianego problemu), rozważmy problem ze stałą liczbą genów, które mają ustalone role, np. $f(x_1, x_2, ..., x_n)$.

1. Wygeneruj losowo początkową populację rozwiązań.
2. Wyucz się powiązań (*linkage*): skonstruuj model powiązań zawierający informacje o podzbiorach zmiennych, które mogą być epistatyczne (współzależne). Odkryj wzajemne zależności wyznaczając miary informacji wzajemnej.
3. Wprowadź zmiany za pomocą operatora optymalnego łączenia genów (*Gene-pool Optimal Mixing*, GOM) – dla każdego osobnika rodzicielskiego w populacji:

Optymalne łączenie genów: GOMEA [Thi18; Dus+24]

Zmodyfikujmy AE tak, aby stał się mniej chaotyczny, bardziej systematyczny i celowy.

(*) Dla uproszczenia (w przeciwieństwie do wcześniej omawianego problemu), rozważmy problem ze stałą liczbą genów, które mają ustalone role, np. $f(x_1, x_2, ..., x_n)$.

1. Wygeneruj losowo początkową populację rozwiązań.
2. Wyucz się powiązań (*linkage*): skonstruuj model powiązań zawierający informacje o podzbiorach zmiennych, które mogą być epistatyczne (współzależne). Odkryj wzajemne zależności wyznaczając miary informacji wzajemnej.
3. Wprowadź zmiany za pomocą operatora optymalnego łączenia genów (*Gene-pool Optimal Mixing*, GOM) – dla każdego osobnika rodzicielskiego w populacji:
   1. Utwórz jego kopię (potomka).

Optymalne łączenie genów: GOMEA [Thi18; Dus+24]

Zmodyfikujmy AE tak, aby stał się mniej chaotyczny, bardziej systematyczny i celowy.

(*) Dla uproszczenia (w przeciwieństwie do wcześniej omawianego problemu), rozważmy problem ze stałą liczbą genów, które mają ustalone role, np. $f(x_1, x_2, ..., x_n)$.

1. Wygeneruj losowo początkową populację rozwiązań.
2. Wyucz się powiązań (*linkage*): skonstruuj model powiązań zawierający informacje o podzbiorach zmiennych, które mogą być epistatyczne (współzależne). Odkryj wzajemne zależności wyznaczając miary informacji wzajemnej.
3. Wprowadź zmiany za pomocą operatora optymalnego łączenia genów (*Gene-pool Optimal Mixing*, GOM) – dla każdego osobnika rodzicielskiego w populacji:
   1. Utwórz jego kopię (potomka).
   2. Wybierz losowy podzbiór zmiennych zależnych z modelu powiązań (z 2).

Optymalne łączenie genów: GOMEA [Thi18; Dus+24]

Zmodyfikujmy AE tak, aby stał się mniej chaotyczny, bardziej systematyczny i celowy.

(*) Dla uproszczenia (w przeciwieństwie do wcześniej omawianego problemu), rozważmy problem ze stałą liczbą genów, które mają ustalone role, np. $f(x_1, x_2, ..., x_n)$.

1. Wygeneruj losowo początkową populację rozwiązań.
2. Wyucz się powiązań (*linkage*): skonstruuj model powiązań zawierający informacje o podzbiorach zmiennych, które mogą być epistatyczne (współzależne). Odkryj wzajemne zależności wyznaczając miary informacji wzajemnej.
3. Wprowadź zmiany za pomocą operatora optymalnego łączenia genów (*Gene-pool Optimal Mixing*, GOM) – dla każdego osobnika rodzicielskiego w populacji:
   1. Utwórz jego kopię (potomka).
   2. Wybierz losowy podzbiór zmiennych zależnych z modelu powiązań (z 2).
   3. Wybierz losowego osobnika „donora" z populacji.

Optymalne łączenie genów: GOMEA [Thi18; Dus+24]

Zmodyfikujmy AE tak, aby stał się mniej chaotyczny, bardziej systematyczny i celowy.

(*) Dla uproszczenia (w przeciwieństwie do wcześniej omawianego problemu), rozważmy problem ze stałą liczbą genów, które mają ustalone role, np. $f(x_1, x_2, ..., x_n)$.

1. Wygeneruj losowo początkową populację rozwiązań.
2. Wyucz się powiązań (*linkage*): skonstruuj model powiązań zawierający informacje o podzbiorach zmiennych, które mogą być epistatyczne (współzależne). Odkryj wzajemne zależności wyznaczając miary informacji wzajemnej.
3. Wprowadź zmiany za pomocą operatora optymalnego łączenia genów (*Gene-pool Optimal Mixing*, GOM) – dla każdego osobnika rodzicielskiego w populacji:
   1. Utwórz jego kopię (potomka).
   2. Wybierz losowy podzbiór zmiennych zależnych z modelu powiązań (z 2).
   3. Wybierz losowego osobnika „donora" z populacji.
   4. Skopiuj wartości zmiennych zależnych (z 3.2) donora do potomka.

Optymalne łączenie genów: GOMEA [Thi18; Dus+24]

Zmodyfikujmy AE tak, aby stał się mniej chaotyczny, bardziej systematyczny i celowy.

(*) Dla uproszczenia (w przeciwieństwie do wcześniej omawianego problemu), rozważmy problem ze stałą liczbą genów, które mają ustalone role, np. $f(x_1, x_2, ..., x_n)$.

1. Wygeneruj losowo początkową populację rozwiązań.
2. Wyucz się powiązań (*linkage*): skonstruuj model powiązań zawierający informacje o podzbiorach zmiennych, które mogą być epistatyczne (współzależne). Odkryj wzajemne zależności wyznaczając miary informacji wzajemnej.
3. Wprowadź zmiany za pomocą operatora optymalnego łączenia genów (*Gene-pool Optimal Mixing*, GOM) – dla każdego osobnika rodzicielskiego w populacji:
   1. Utwórz jego kopię (potomka).
   2. Wybierz losowy podzbiór zmiennych zależnych z modelu powiązań (z 2).
   3. Wybierz losowego osobnika „donora" z populacji.
   4. Skopiuj wartości zmiennych zależnych (z 3.2) donora do potomka.
   5. Oceń potomka po każdej jego zmianie.

Optymalne łączenie genów: GOMEA [Thi18; Dus+24]

Zmodyfikujmy AE tak, aby stał się mniej chaotyczny, bardziej systematyczny i celowy.

(*) Dla uproszczenia (w przeciwieństwie do wcześniej omawianego problemu), rozważmy problem ze stałą liczbą genów, które mają ustalone role, np. $f(x_1, x_2, ..., x_n)$.

1. Wygeneruj losowo początkową populację rozwiązań.
2. Wyucz się powiązań (*linkage*): skonstruuj model powiązań zawierający informacje o podzbiorach zmiennych, które mogą być epistatyczne (współzależne). Odkryj wzajemne zależności wyznaczając miary informacji wzajemnej.
3. Wprowadź zmiany za pomocą operatora optymalnego łączenia genów (*Gene-pool Optimal Mixing*, GOM) – dla każdego osobnika rodzicielskiego w populacji:
   1. Utwórz jego kopię (potomka).
   2. Wybierz losowy podzbiór zmiennych zależnych z modelu powiązań (z 2).
   3. Wybierz losowego osobnika „donora" z populacji.
   4. Skopiuj wartości zmiennych zależnych (z 3.2) donora do potomka.
   5. Oceń potomka po każdej jego zmianie.
   6. Gdy ocena potomka jest równa lub lepsza niż rodzica, zastąp rozwiązanie rodzica.

Optymalne łączenie genów: GOMEA [Thi18; Dus+24]

Zmodyfikujmy AE tak, aby stał się mniej chaotyczny, bardziej systematyczny i celowy.

(*) Dla uproszczenia (w przeciwieństwie do wcześniej omawianego problemu), rozważmy problem ze stałą liczbą genów, które mają ustalone role, np. $f(x_1, x_2, ..., x_n)$.

1. Wygeneruj losowo początkową populację rozwiązań.
2. Wyucz się powiązań (*linkage*): skonstruuj model powiązań zawierający informacje o podzbiorach zmiennych, które mogą być epistatyczne (współzależne). Odkryj wzajemne zależności wyznaczając miary informacji wzajemnej.
3. Wprowadź zmiany za pomocą operatora optymalnego łączenia genów (*Gene-pool Optimal Mixing*, GOM) – dla każdego osobnika rodzicielskiego w populacji:
   1. Utwórz jego kopię (potomka).
   2. Wybierz losowy podzbiór zmiennych zależnych z modelu powiązań (z 2).
   3. Wybierz losowego osobnika „donora" z populacji.
   4. Skopiuj wartości zmiennych zależnych (z 3.2) donora do potomka.
   5. Oceń potomka po każdej jego zmianie.
   6. Gdy ocena potomka jest równa lub lepsza niż rodzica, zastąp rozwiązanie rodzica.
4. Skocz do 2 (powtarzaj aż do uzyskania zbieżności).

Optymalne łączenie genów: GOMEA – uwagi

- Czego brakuje z tradycyjnego AE?

Optymalne łączenie genów: GOMEA – uwagi

- Czego brakuje z tradycyjnego AE?
- Analogie do optymalizacji lokalnej! (krok 3.6).

Optymalne łączenie genów: GOMEA – uwagi

- Czego brakuje z tradycyjnego AE?
- Analogie do optymalizacji lokalnej! (krok 3.6).
- Naturalny, wyraźny moment ostatecznego zakończenia algorytmu.

Optymalne łączenie genów: GOMEA – uwagi

- Czego brakuje z tradycyjnego AE?
- Analogie do optymalizacji lokalnej! (krok 3.6).
- Naturalny, wyraźny moment ostatecznego zakończenia algorytmu.
- Można iteracyjnie ulepszać jednego potomka, przechodząc przez różne podzbiory zmiennych zależnych i donorów (kroki 3.2–3.5, akceptując tylko donacje polepszające).

Optymalne łączenie genów: GOMEA – uwagi

- Czego brakuje z tradycyjnego AE?
- Analogie do optymalizacji lokalnej! (krok 3.6).
- Naturalny, wyraźny moment ostatecznego zakończenia algorytmu.
- Można iteracyjnie ulepszać jednego potomka, przechodząc przez różne podzbiory zmiennych zależnych i donorów (kroki 3.2–3.5, akceptując tylko donacje polepszające).
- Brak mutacji w podstawowym wariancie – polega na różnorodności w populacji początkowej.

Optymalne łączenie genów: GOMEA – uwagi

- Czego brakuje z tradycyjnego AE?
- Analogie do optymalizacji lokalnej! (krok 3.6).
- Naturalny, wyraźny moment ostatecznego zakończenia algorytmu.
- Można iteracyjnie ulepszać jednego potomka, przechodząc przez różne podzbiory zmiennych zależnych i donorów (kroki 3.2–3.5, akceptując tylko donacje polepszające).
- Brak mutacji w podstawowym wariancie – polega na różnorodności w populacji początkowej.
- Operator GOM przypomina (zastępuje) krzyżowanie.

Optymalne łączenie genów: GOMEA – uwagi

- Czego brakuje z tradycyjnego AE?
- Analogie do optymalizacji lokalnej! (krok 3.6).
- Naturalny, wyraźny moment ostatecznego zakończenia algorytmu.
- Można iteracyjnie ulepszać jednego potomka, przechodząc przez różne podzbiory zmiennych zależnych i donorów (kroki 3.2–3.5, akceptując tylko donacje polepszające).
- Brak mutacji w podstawowym wariancie – polega na różnorodności w populacji początkowej.
- Operator GOM przypomina (zastępuje) krzyżowanie.
- Jeśli liczba genów jest zmienna lub nie mają one ustalonych ról (nasze założenie (*): jak wykonalibyśmy krok 2 bez niego?), wtedy można użyć słabszego uogólnienia → Compatible Substitutions Optimization [KM25].

Optymalne łączenie genów: GOMEA – uwagi

- Czego brakuje z tradycyjnego AE?
- Analogie do optymalizacji lokalnej! (krok 3.6).
- Naturalny, wyraźny moment ostatecznego zakończenia algorytmu.
- Można iteracyjnie ulepszać jednego potomka, przechodząc przez różne podzbiory zmiennych zależnych i donorów (kroki 3.2–3.5, akceptując tylko donacje polepszające).
- Brak mutacji w podstawowym wariancie – polega na różnorodności w populacji początkowej.
- Operator GOM przypomina (zastępuje) krzyżowanie.
- Jeśli liczba genów jest zmienna lub nie mają one ustalonych ról (nasze założenie (*): jak wykonalibyśmy krok 2 bez niego?), wtedy można użyć słabszego uogólnienia → Compatible Substitutions Optimization [KM25].
- Lepsze wyniki niż tradycyjny AE (przypomnienie: wykres zbieżności w czasie oraz co oznacza “lepsze” w optymalizacji).

Strategie ewolucyjne (ang. *Evolutionary Strategies*, ES) rozwijały się przez pewien czas niezależnie od AG, jako metody służące do optymalizacji numerycznej. Wiele aspektów różni je od AG; wspólne jest wykorzystanie mechanizmów ewolucji podczas optymalizacji.

Strategie ewolucyjne (ang. *Evolutionary Strategies*, ES) rozwijały się przez pewien czas niezależnie od AG, jako metody służące do optymalizacji numerycznej. Wiele aspektów różni je od AG; wspólne jest wykorzystanie mechanizmów ewolucji podczas optymalizacji.

ES – naturalne pochodzenie: jedno z pierwszych zastosowań (1964) to *evolutionary design*, czyli projektownie konstrukcji (powiemy o tym problemie później i poeksperymentujemy na laboratorium). W celu minimalizacji oporów przepływu wody i optymalizacji kształtów rur, aby ocenić konstrukcję lub rurociąg, nie symulowano jej, tylko budowano [Rec84, zob. rys. na str. 123]; zmiany konstrukcji odpowiadały „mutacjom". Był to zatem sposób postępowania w świecie rzeczywistym realizujący algorytm optymalizacji.

Wczesne strategie ewolucyjne używały jedynie operatora mutacji, który modyfikował jedynego przetwarzanego osobnika. Inaczej niż w algorytmach genetycznych, osobnik był parą składającą się z wektora wartości zmiennych i wektora odchyleń standardowych (stałego podczas całego procesu ewolucji). Mutacja polegała na zmianie każdej zmiennej wektora wartości o losowy czynnik wygenerowany zgodnie z rozkładem normalnym o odpowiednim odchyleniu standardowym (określonym w wektorze odchyleń standardowych). Osobnik po mutacji zastępował swojego przodka jedynie wówczas, gdy był od niego lepszy i dopuszczalny.

Taka strategia została nazwana dwuelementową (ponieważ w pewnym momencie istnieje jeden potomek i jeden przodek) i jest oznaczana (1+1)-ES, a jej działanie przypomina *Local Search*.

Ulepszeniem strategii dwuelementowej jest strategia wieloelementowa, w której, podobnie jak w AG, istnieje populacja osobników. Wprowadza się dodatkowo operację krzyżowania jednorodnego, jednak nie stosuje się jej do wszystkich osobników, tylko do dwóch z nich – tak, że powstaje jeden potomek, który zastępuje osobnika najsłabszego (jeden nowy osobnik – a więc analogicznie, jak w algorytmach ewolucyjnych typu *steady-state*).

Kolejnym udoskonaleniem było stosowanie krzyżowania wiele razy w jednym kroku (powstawało wielu potomków), a następnie wybór z przodków i potomków *POPSIZE* osobników (tzw. selekcja typu „plus"). W innym podejściu wybiera się osobników do następnego pokolenia tylko z grupy potomków (tzw. selekcja typu „przecinek", ang. *comma selection*), co jest korzystne w zadaniach ze zmieniającym się optimum.

Ogólny i zwięzły zapis architektury ES ma postać $(\mu/\rho, \lambda)$-ES albo $(\mu/\rho + \lambda)$-ES, gdzie μ to liczba rodziców, $\rho \leq \mu$ to liczba rodziców z których wywodzą się potomkowie, λ to liczba potomków, przecinek oznacza selekcję tylko ze zbioru potomków, a plus – z obu zbiorów: rodziców i potomków.

Strategie ewolucyjne: specjalna notacja

Ogólny i zwięzły zapis architektury ES ma postać $(\mu/\rho, \lambda)$-ES albo $(\mu/\rho + \lambda)$-ES, gdzie μ to liczba rodziców, $\rho \leq \mu$ to liczba rodziców z których wywodzą się potomkowie, λ to liczba potomków, przecinek oznacza selekcję tylko ze zbioru potomków, a plus – z obu zbiorów: rodziców i potomków.

than the $(1, 5)$-ES with
pplicability of the CMA

of a $(1, 10)$-ES vs. that of a $(1 + 10)$-ES

(3) The selection operator
$(1+\mu) - ES$ is chosen in

ited operating zones. The
on the classical $(\mu+\lambda)$-ES
search for new solutions are

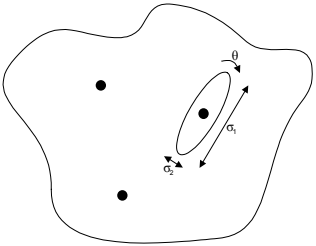
from genetic repair are significant, and the performance of the $(\mu/\mu, \lambda)$-ES is much more robust in the presence of noise than that of the $(1+1)$-ES.

(μ_I, λ)-CMA-ES,
from λ and μ as
tational effort, th

- selection procedure type $(\mu + \lambda)$;
- recombination operator takes the average

Stosuje się ulepszony operator mutacji, który zmienia nie tylko wartość zmiennej, ale także odchylenie standardowe zmian, które podlega również ewolucji. Do reprezentacji osobnika, oprócz wartości zmiennych i odchyleń standardowych, można wprowadzić dodatkowo informację o preferowanym kącie odchylenia podczas procesu przeszukiwania i w ten sposób poprawić szybkość zbieżności strategii ewolucyjnych. Zmienna jest wówczas reprezentowana przez jej wartość, odchylenie standardowe i kąt odchylenia, i wszystkie te wielkości podlegają ewolucji pozwalając na samoadaptację i umożliwiając dokładne dostrojenie lokalne.

Używa się również krzyżowania arytmetycznego (średnia ważona rodziców).



Rysunek: Parametry mutacji w strategiach ewolucyjnych.

Strategie ewolucyjne: macierz kowariancji mutacji

Znana z wydajności jest strategia ewolucyjna dostosowująca macierz kowariancji mutacji, CMA-ES,* której implementacja jest dostępna np. w bibliotece DEAP.** Ta strategia jest też adekwatna dla problemów źle uwarunkowanych (ang. ill-conditioned).

Metoda CMA-ES ma wiele parametrów, istnieje też wiele alternatywnych mechanizmów dla każdego kroku tej metody. Można stosować wartości domyślne oraz polityki wielu uruchomień uwalniające użytkownika od konieczności podawania wartości jakichkolwiek parametrów.

*https://en.wikipedia.org/wiki/CMA-ES
**https://deap.readthedocs.io/en/master/examples/cmaes.html

Strategie ewolucyjne: główna idea CMA-ES

- ustalamy środek populacji,

Strategie ewolucyjne: główna idea CMA-ES

- ustalamy środek populacji,
- próbkujemy rozwiązania z wielowymiarowego (n) rozkładu normalnego (dany jednym parametrem – izometryczny, albo n parametrami – skalowanie równoległe do osi, albo $\binom{n}{2}$ parametrami czyli macierzą kowariancji – umożliwia obracanie),

Strategie ewolucyjne: główna idea CMA-ES

- ustalamy środek populacji,
- próbkujemy rozwiązania z wielowymiarowego (n) rozkładu normalnego (dany jednym parametrem – izometryczny, albo n parametrami – skalowanie równoległe do osi, albo $\binom{n}{2}$ parametrami czyli macierzą kowariancji – umożliwia obracanie),
- oceniamy wszystkie rozwiązania,

Strategie ewolucyjne: główna idea CMA-ES

- ustalamy środek populacji,
- próbkujemy rozwiązania z wielowymiarowego (n) rozkładu normalnego (dany jednym parametrem – izometryczny, albo n parametrami – skalowanie równoległe do osi, albo $\binom{n}{2}$ parametrami czyli macierzą kowariancji – umożliwia obracanie),
- oceniamy wszystkie rozwiązania,
- przesuwamy środek populacji: ustawiamy go w miejscu średniej ważonej jakością (rankingową) najlepszych osobników. Rankingowość powoduje nieczułość na niewielkie zaburzenia oceny („chropowatość" krajobrazu) oraz jego wyginanie – stopień wklęsłości,

Strategie ewolucyjne: główna idea CMA-ES

- ustalamy środek populacji,
- próbkujemy rozwiązania z wielowymiarowego (n) rozkładu normalnego (dany jednym parametrem – izometryczny, albo n parametrami – skalowanie równoległe do osi, albo $\binom{n}{2}$ parametrami czyli macierzą kowariancji – umożliwia obracanie),
- oceniamy wszystkie rozwiązania,
- przesuwamy środek populacji: ustawiamy go w miejscu średniej ważonej jakością (rankingową) najlepszych osobników. Rankingowość powoduje nieczułość na niewielkie zaburzenia oceny („chropowatość" krajobrazu) oraz jego wyginanie – stopień wklęsłości,
- rozproszenie nowych (próbkowanych) osobników jest proporcjonalne do prędkości, z jaką przemieszcza się środek populacji: wolniejsze przemieszczanie $\rightarrow$ mniejsze rozproszenie,

Strategie ewolucyjne: główna idea CMA-ES

- ustalamy środek populacji,
- próbkujemy rozwiązania z wielowymiarowego (n) rozkładu normalnego (dany jednym parametrem – izometryczny, albo n parametrami – skalowanie równoległe do osi, albo $\binom{n}{2}$ parametrami czyli macierzą kowariancji – umożliwia obracanie),
- oceniamy wszystkie rozwiązania,
- przesuwamy środek populacji: ustawiamy go w miejscu średniej ważonej jakością (rankingową) najlepszych osobników. Rankingowość powoduje nieczułość na niewielkie zaburzenia oceny („chropowatość" krajobrazu) oraz jego wyginanie – stopień wklęsłości,
- rozproszenie nowych (próbkowanych) osobników jest proporcjonalne do prędkości, z jaką przemieszcza się środek populacji: wolniejsze przemieszczanie $\rightarrow$ mniejsze rozproszenie,
- aktualizujemy macierz kowariancji tak, żeby rozciągnąć nieco wielowymiarowy rozkład normalny w kierunku przemieszczenia środka populacji. W ten sposób będziemy dalej podążali zgodnie z aproksymowanym gradientem oczekiwanej jakości rozwiązań.

Ewolucja różnicowa (*differential evolution*)

Specyficzną cechą ewolucja różnicowej jest mutacja różnicowa [SP97]. W każdej iteracji ewolucji, dla każdego osobnika o z populacji N osobników powtarzamy:

- losujemy n unikatowych z N osobników, wybieramy z nich osobnika bazowego β i osobnika różnicy δ (dla $n = 3$, β może być wybrany z nich losowo, a δ może być różnicą dwóch pozostałych osobników),
- tworzymy osobnika tymczasowego („donora") $\omega = \beta + F\delta$ (F – stała),
- krzyżujemy ω z o,
- decydujemy czy efekt krzyżowania ma zastąpić oryginalnego o, czy nie (selekcja).

DE jest znana ze swojej prostoty, małej liczby parametrów (przykładowa implementacja) i szybkiej zbieżności. Nie wymaga określenia osobnego, niezależnego rozkładu prawdopodobieństwa dla mutacji – mutacja wynika ze stanu populacji. Warianty DE są konkurencyjne w stosunku do innych algorytmów w corocznych konkursach optymalizacyjnych.

Tworzenie osobnika tymczasowego ω: porównaj później omawiane krzyżowanie simpleksowe.

Programowanie ewolucyjne (*evolutionary programming*)

Trzy główne różnice pomiędzy EP a GA:

1. Reprezentacja rozwiązania nie musi być binarna – wynika naturalnie z problemu.
2. Mutacja zmienia fragmenty rozwiązania, przy czym małe zmiany są częstsze, a duże – rzadsze.
3. Krzyżowanie może nie występować.

Obecnie „evolutionary programming" jest nazwą rzadko używaną, zamiast niej mówi się o algorytmie ewolucyjnym – co oznacza ogólnie, że użyto algorytmu przystosowanego do danego problemu. Stopień jego przystosowania bywa różny; najczęściej obejmuje reprezentację i operatory.

Wykorzystuje się wiele reprezentacji osobników: zbiór, lista, permutacja*, drzewo, graf nieskierowany, graf skierowany, macierz, wyrażenia logiczne, reguły (jak w genetycznym uczeniu maszynowym, LCS/GBML), sieci neuronowe, automaty, wyrażenia opisane gramatyką (np. zapisane w ONP), wyrażenia o strukturze drzewa, programy (jak w omawianym dalej GP), ...

*Krzyżowanie dla permutacji: OX, PMX, ERO, inne: https://hrcak.srce.hr/file/163313

Reprezentacja liczb rzeczywistych – operatory

Bardzo często w AE (i w optymalizacji w ogóle!) stosuje się reprezentację wartości ciągłych. Geny kodują liczby rzeczywiste w takim formacie, jak jest to przyjęte na procesorach (zmienna precyzja w zależności od bezwzględnej wartości liczby).

Pytanie: jakie można zaproponować operatory krzyżowania i mutacji (oprócz typowych, takich jak wymiana genów albo wielopunktowe) dla wektora liczb? Proponując operatory pamiętaj o celu krzyżowania i mutacji.

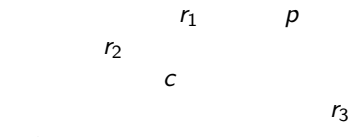
Krzyżowanie: np. średnia z rodziców, lub średnia ważona by uzyskać dwóch różnych potomków. Średnia ważona to krzyżowanie **arytmetyczne** – dzieci są liniową kombinacją rodziców: $d_1 = r_1 \cdot a + r_2 \cdot (1 - a)$, $d_2 = \ldots$

Wagę a można losować przy każdym wykonaniu.

Krzyżowanie simpleksowe

Wyznaczamy centroid c rodziców r.

- Wariant bez dostępu do jakości rozwiązań – SPX [TYH99]: losujemy potomka z (powiększonej) przestrzeni kombinacji liniowych rodziców (odsuniętych od c o ε – *the expanding rate*).
- Wariant z uwzględnieniem jakości: tworzymy potomka p jako przesunięcie od najgorszego osobnika/rodzica dalej poprzez punkt c.



Reprezentacja liczb rzeczywistych – mutacja

- *uniform random* (**Flat**) – ustal gen na wartość losową z dozwolonego przedziału
- *creep* – zmień gen o wartość losowaną z pewnego rozkładu (np. normalnego albo jednostajnego – np. $-3..+3$, itp.)

Aby uniezależnić się od „osi" (tzn. aby mutacja nie przebiegała tylko równolegle do osi, czyli nie dotyczyła pojedynczych parametrów, co byłoby niekorzystne gdyby funkcja celu była np. obróconą wersją funkcji zależnej wprost od parametrów), mutuje się naraz wszystkie geny (i wtedy stosujemy rozkład normalny losowanej zmiany a nie równomierny – zastanów się dlaczego).

Chcąc zapewnić, że taka mutacja naraz n elementów wektora przesunie bieżące rozwiązanie o taką samą odległość w n-wymiarowej przestrzeni, jak zrobiłaby to mutacja tylko jednego wymiaru, przez jaką wartość należy podzielić (znormalizować) każdą z n wylosowanych wartości zmiany w n-wymiarowym wektorze?

Co zrobić, jeśli po zmutowaniu wartość genu wychodzi poza dozwolony zakres? Jakie metody rozwiązania tego problemu można zaproponować? [Bul99; Bul01]

Reprezentacja liczb rzeczywistych – mutacja

Metody radzenia sobie z mutantami spoza dozwolonego przedziału

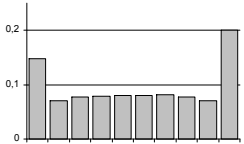
- **Pochłaniaj**: niepoprawna po mutacji wartość jest obcinana do najbliższej granicy przedziału – znany trick max(min()).
- **Powtarzaj**: zmutowana wartość jest generowana tak długo, aż uzyskamy poprawną wartość.
- **Zastępuj**: zmutowana wartość jest generowana tak długo, aż uzyskamy poprawną wartość, za każdym razem wybierając od nowa przodka.

- **Ignoruj**: niepoprawna po mutacji wartość jest zapominana; mutant uzyskuje oryginalną wartość swojego przodka.
- **Odbijaj**: niepoprawna po mutacji wartość leżąca d powyżej (lub poniżej) najbliższej jej granicy przedziału jest zastępowana przez wartość leżąca d poniżej (lub powyżej) tej granicy.
- **Zawijaj**: mutowana cecha jest traktowana tak, jakby była cykliczna. Przedział się „zawija", a zmutowana wartość jest wyznaczana modulo szerokość przedziału.

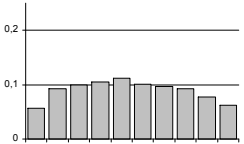
Czy ma znaczenie, którą z metod wybierzemy? Tak! [dyskusja pożądanych cech mutacji i wybór zwycięskiej metody].

Wyniki eksperymentalne: jak często zdarzały się po mutacji i „naprawie" różnymi metodami określone wartości genu? Przodek miał równe szanse na każdą wartość. Oś pozioma – przedział zmienności genu, oś pionowa – częstość występowania wartości potomka w podprzedziałach:

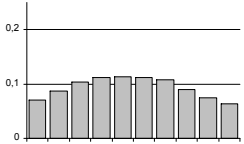
Pochłaniaj



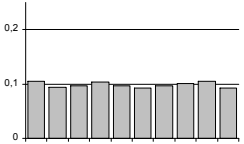
Powtarzaj



Zastępuj



Ignoruj, Odbijaj, Zawijaj, Flat – podobne do:



Reprezentacja liczb rzeczywistych – mutacja

Potrzeba dokładniejszej analizy zachowania mechanizmów naprawy

To, że w metodach **Ignoruj**, **Odbijaj**, **Zawijaj**, uzyskano taki sam rozkład jak we **Flat** nie oznacza, że metody te zachowują się tak samo. Faktycznie zasada ich działania jest zupełnie inna. Przykład: metoda **Ignoruj**. W okolicy granic przedziału więcej mutacji będzie nielegalnych i ignorowanych. Skoro ignorowanych, to rzadziej też będą trafiać się w ogóle wartości bliskie granicom. Kiedy już się trafią, rzadko doprowadzą do poprawnej mutacji...

Zatem to, że rozkład jest taki sam, nie gwarantuje jeszcze, że tak samo często zdarzają się efektywne (faktycznie zmieniające wartość genu) mutacje w poszczególnych podprzedziałach. Stąd oprócz rozkładu częstości wartości genu dla różnych metod należy też porównać inne parametry – np. ile liczb ubywa z określonego podprzedziału (przed mutacją) i przybywa (po mutacji), jaki jest między nimi związek, itp. W tym świetle metody **Ignoruj**, **Odbijaj**, **Zawijaj** różnią się między sobą, i żadna z nich nie zachowuje się tak jak **Flat**.

Rozpatrywaliśmy tu nieskomplikowany rodzaj mutacji i proste mechanizmy, a jednak te drobne elementy mają duży wpływ na proces ewolucji. Na przykład **Pochłaniaj** ukierunkowuje (*bias*) **cały czas** dryf genetyczny ku krańcowym wartościom dozwolonych przedziałów. Nie będąc tego świadomym można wyciągnąć wniosek, że są one optymalne i ewolucja je faworyzuje – tymczasem jest to ciągły wpływ mutacji.

Przypomnienie z poprzedniego semestru (i ew. uzupełnienie)

- Mając kilka pomysłów na operatory krzyżowania, skąd wiemy, który z nich będzie prawdopodobnie lepszy?

Przypomnienie z poprzedniego semestru (i ew. uzupełnienie)

- Mając kilka pomysłów na operatory krzyżowania, skąd wiemy, który z nich będzie prawdopodobnie lepszy?
- Jak świadomie tworzyć (projektować) skuteczne operatory krzyżowania?

Przypomnienie z poprzedniego semestru (i ew. uzupełnienie)

- Mając kilka pomysłów na operatory krzyżowania, skąd wiemy, który z nich będzie prawdopodobnie lepszy?
- Jak świadomie tworzyć (projektować) skuteczne operatory krzyżowania?
- Czym charakteryzuje się DPX – *distance-preserving crossover*?

Przypomnienie z poprzedniego semestru (i ew. uzupełnienie)

- Mając kilka pomysłów na operatory krzyżowania, skąd wiemy, który z nich będzie prawdopodobnie lepszy?
- Jak świadomie tworzyć (projektować) skuteczne operatory krzyżowania?
- Czym charakteryzuje się DPX – *distance-preserving crossover*?
- Jak wiedza z powyższych pytań przenosi się na operator mutacji?

Przypomnienie z poprzedniego semestru (i ew. uzupełnienie)

- Mając kilka pomysłów na operatory krzyżowania, skąd wiemy, który z nich będzie prawdopodobnie lepszy?
- Jak świadomie tworzyć (projektować) skuteczne operatory krzyżowania?
- Czym charakteryzuje się DPX – *distance-preserving crossover*?
- Jak wiedza z powyższych pytań przenosi się na operator mutacji?
- Czym jest *globalna wypukłość* i jak ją wykryć?

Przypomnienie z poprzedniego semestru (i ew. uzupełnienie)

- Mając kilka pomysłów na operatory krzyżowania, skąd wiemy, który z nich będzie prawdopodobnie lepszy?
- Jak świadomie tworzyć (projektować) skuteczne operatory krzyżowania?
- Czym charakteryzuje się DPX – *distance-preserving crossover*?
- Jak wiedza z powyższych pytań przenosi się na operator mutacji?
- Czym jest *globalna wypukłość* i jak ją wykryć?
- Co ocenia miara FDC?

Przypomnienie z poprzedniego semestru (i ew. uzupełnienie)

- Mając kilka pomysłów na operatory krzyżowania, skąd wiemy, który z nich będzie prawdopodobnie lepszy?
- Jak świadomie tworzyć (projektować) skuteczne operatory krzyżowania?
- Czym charakteryzuje się DPX – *distance-preserving crossover*?
- Jak wiedza z powyższych pytań przenosi się na operator mutacji?
- Czym jest *globalna wypukłość* i jak ją wykryć?
- Co ocenia miara FDC?

Przypomnienie z poprzedniego semestru (i ew. uzupełnienie)

- Mając kilka pomysłów na operatory krzyżowania, skąd wiemy, który z nich będzie prawdopodobnie lepszy?
- Jak świadomie tworzyć (projektować) skuteczne operatory krzyżowania?
- Czym charakteryzuje się DPX – *distance-preserving crossover*?
- Jak wiedza z powyższych pytań przenosi się na operator mutacji?
- Czym jest *globalna wypukłość* i jak ją wykryć?
- Co ocenia miara FDC?

- Zaproponuj różne miary odległości (niepodobieństwa) rozwiązań: D_1, D_2, D_3, ... opierając się na różnych cechach/właściwościach rozwiązań, które wpływają na ich jakość

- Mając kilka pomysłów na operatory krzyżowania, skąd wiemy, który z nich będzie prawdopodobnie lepszy?
- Jak świadomie tworzyć (projektować) skuteczne operatory krzyżowania?
- Czym charakteryzuje się DPX – *distance-preserving crossover*?
- Jak wiedza z powyższych pytań przenosi się na operator mutacji?
- Czym jest *globalna wypukłość* i jak ją wykryć?
- Co ocenia miara FDC?

- Zaproponuj różne miary odległości (niepodobieństwa) rozwiązań: D_1, D_2, D_3, ... opierając się na różnych cechach/właściwościach rozwiązań, które wpływają na ich jakość
- Wyznacz FD_1C, FD_2C, ... i odkryj miarę maksymalizującą FDC dla danego F

Przypomnienie z poprzedniego semestru (i ew. uzupełnienie)

- Mając kilka pomysłów na operatory krzyżowania, skąd wiemy, który z nich będzie prawdopodobnie lepszy?
- Jak świadomie tworzyć (projektować) skuteczne operatory krzyżowania?
- Czym charakteryzuje się DPX – *distance-preserving crossover*?
- Jak wiedza z powyższych pytań przenosi się na operator mutacji?
- Czym jest *globalna wypukłość* i jak ją wykryć?
- Co ocenia miara FDC?

- Zaproponuj różne miary odległości (niepodobieństwa) rozwiązań: D_1, D_2, D_3, ... opierając się na różnych cechach/właściwościach rozwiązań, które wpływają na ich jakość
- Wyznacz FD_1C, FD_2C, ... i odkryj miarę maksymalizującą FDC dla danego F
- Zbuduj operatory krzyżowania (DPX) i mutacji (sąsiedztwa) respektujące wybraną miarę D.

Ilościowe ocenianie podobieństwa/odległości rozwiązań

Miara podobieństwa rozwiązań ma liczne zastosowania – przydaje się m.in. do:

- testowania pomysłów na operator krzyżowania – różne cechy rozwiązań i wartości FDC,
- prowadzenia selekcji ze ściskiem – omówionej wcześniej,
- szacowania różnorodoności w populacji i oceny zbieżności,
- analizy struktury populacji; analizy skupień w zbiorze rozwiązań,
- utrzymywania „gatunków" podczas ewolucji – te metody będą jeszcze omawiane,

oraz wszędzie tam, gdzie występuje potrzeba wyznaczenia różnicy dwóch rozwiązań, np. w przedstawionych już: ewolucji różnicowej i krzyżowaniu simpleksowym.

Jeśli rozwiązania posiadają prostą reprezentację (rozważ kilka przykładów), wtedy pomysły na miary podobieństwa mogą nasuwać się same. Dla złożonych reprezentacji (rozważ kilka przykładów) przydatne mogą być pojęcia odległości edycyjnej* oraz odległości spychaczowej** (tłum. MK).

*https://en.wikipedia.org/wiki/Edit_distance
**https://en.wikipedia.org/wiki/Earth_mover%27s_distance

Embriogeneza

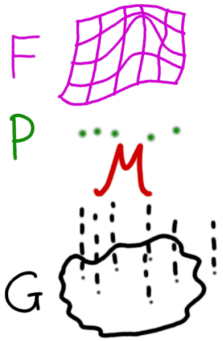
Embriogeneza: mapowanie genotyp $\rightarrow$ fenotyp. Dla prostych reprezentacji i równomiernych, jednorodnych przestrzeni takich jak kompletna przestrzeń bitów, liczb lub permutacji, pierwszym (domyślnym) pomysłem jest trywialne, bezpośrednie mapowanie 1:1.

Ale czy takie mapowanie jest najlepszym wyborem?

Embriogeneza

Embriogeneza: mapowanie genotyp → fenotyp. Dla prostych reprezentacji i równomiernych, jednorodnych przestrzeni takich jak kompletna przestrzeń bitów, liczb lub permutacji, pierwszym (domyślnym) pomysłem jest trywialne, bezpośrednie mapowanie 1:1.

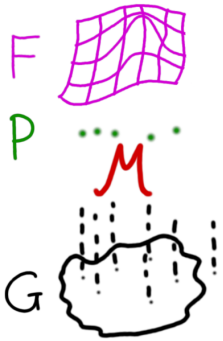
Ale czy takie mapowanie jest najlepszym wyborem?



Embriogeneza

Embriogeneza: mapowanie genotyp → fenotyp. Dla prostych reprezentacji i równomiernych, jednorodnych przestrzeni takich jak kompletna przestrzeń bitów, liczb lub permutacji, pierwszym (domyślnym) pomysłem jest trywialne, bezpośrednie mapowanie 1:1.

Ale czy takie mapowanie jest najlepszym wyborem?



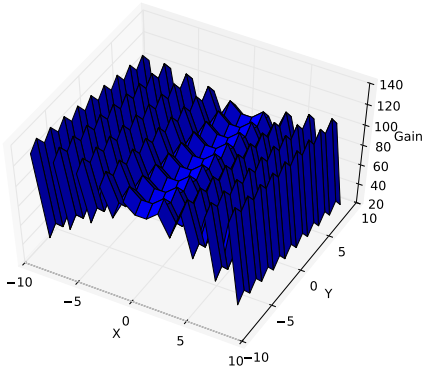
Przypomnij sobie RGB ↔ HSL, sygnał ↔ widmo, kran  ↔  , ...

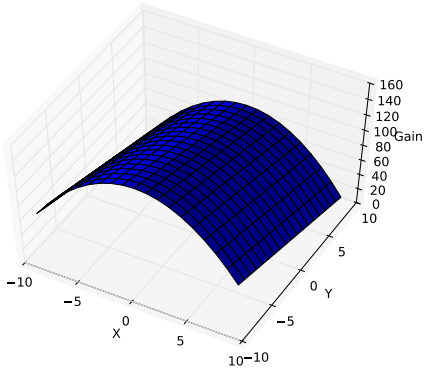
Embriogeneza – po co?

Zastanów się, w jakich sytuacjach mapowanie genotyp $\rightarrow$ fenotyp powinno (albo musi?) być bardziej skomplikowane. Jakie cechy mapowania powinna zapewnić procedura mapująca przestrzeń genotypów w przestrzeń fenotypów?

Embriogeneza – po co?

Zastanów się, w jakich sytuacjach mapowanie genotyp → fenotyp powinno (albo musi?) być bardziej skomplikowane. Jakie cechy mapowania powinna zapewnić procedura mapująca przestrzeń genotypów w przestrzeń fenotypów?





Embriogeneza – po co?

Zastanów się, w jakich sytuacjach mapowanie genotyp → fenotyp powinno (albo musi?) być bardziej skomplikowane. Jakie cechy mapowania powinna zapewnić procedura mapująca przestrzeń genotypów w przestrzeń fenotypów?

Pomyśl teraz o naturze, o organizmach żywych i o biologicznym mapowaniu genotyp → fenotyp. Jakie ono jest i czy jest korzystne? Czy dałoby się zrealizować to mapowanie lepiej?

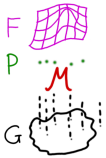
Embriogeneza – po co?

Zastanów się, w jakich sytuacjach mapowanie genotyp → fenotyp powinno (albo musi?) być bardziej skomplikowane. Jakie cechy mapowania powinna zapewnić procedura mapująca przestrzeń genotypów w przestrzeń fenotypów?

Pomyśl teraz o naturze, o organizmach żywych i o biologicznym mapowaniu genotyp → fenotyp. Jakie ono jest i czy jest korzystne? Czy dałoby się zrealizować to mapowanie lepiej?

Jeśli przestrzeń fenotypów jest inna niż genotypów (co ma miejsce np. wtedy, kiedy rozwiązania są bardzo skomplikowane – wyobraź sobie optymalizację mostu, samochodu, robota, ...), potrzebna jest procedura „mapowania" jednej przestrzeni w drugą. W biologii ten proces to embriogeneza (rozwój z genotypu do stadium zarodka – budowanie ciała). Ale nawet dla identycznych przestrzeni, niebezpośrednie mapowanie może przynieść korzyści.

Embriogeneza – decyzje i ich konsekwencje [Rot06]



- nadmiarowość: wiele genotypów → jeden fenotyp
  - sąsiednia (*synonymous*): genotypy tworzące ten sam fenotyp są sąsiadami
    - równoliczna każdy fenotyp powstaje z takiej samej liczby genotypów
    - różnoliczna: przeciwnie
  - odległa (*non-synonymous*): niekorzystne dla optymalizacji
- skalowanie alleli: na ile jednakowy jest wpływ alleli na fitness
- lokalność: podobieństwo (bliskość) genotypów skorelowane z podobieństwem odpowiadających im fenotypów
  - wysoka: dobrze! mapowanie nie zwiększa trudności problemu
  - niska: zwiększa trudność problemu

Powyższe własności można oszacować liczbowo.

Embriogeneza – właściwości i zalety

Możliwe powody skłaniające do stosowania nietrywialnego mapowania [Ben99]:

- ograniczenie (zmniejszenie) przestrzeni przeszukiwania (rekurencyjne, hierarchiczne, itd.),
- lepsze próbkowanie przestrzeni przeszukiwania (tworzące topologię zwiększającą FDC),
- bardziej złożone rozwiązania w przestrzeni fenotypów („procedura wzrostu" zapisana w genotypie),
- lepsza obsługa ograniczeń (mapowanie **każdego** genotypu na poprawny fenotyp),

oraz:

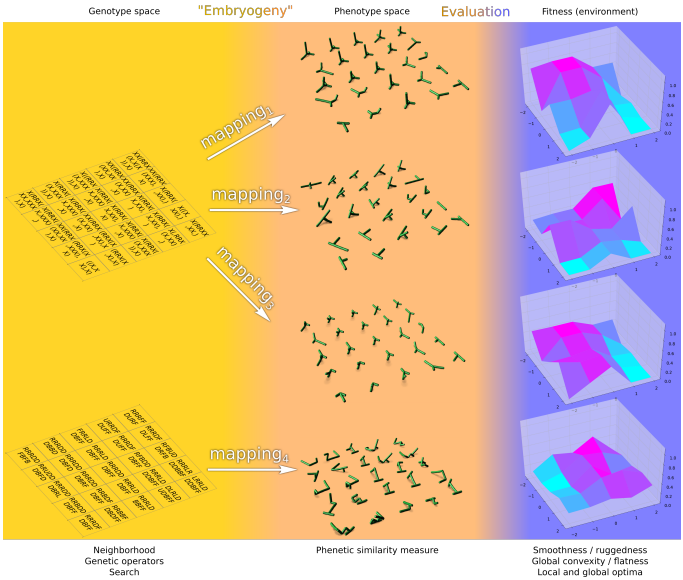
- kompresja: proste genotypy opisują skomplikowane fenotypy,
- powtarzanie: genotypy mogą opisywać symetrię, modularność, podprocedury, itd.,
- adaptacja: fenotypy mogą rozwijać się „adaptacyjnie" (żeby dopasować się do ograniczeń, albo żeby naprawiać błędy).

Embriogeneza – trudności

- potrzeba dużego doświadczenia, aby ręcznie zdefiniować embriogenezę zapewniającą wymienione na poprzednim slajdzie zalety,
- trudno automatycznie wyewoluować embriogenezę (potrzeba specjalizowanych operatorów zabezpieczających przed „puchnięciem" (ang. *bloat*) genotypów i fenotypów, epistazą i psuciem potomków przez operatory lub słabym dziedziczeniem informacji).

W większości zastosowań embriogeneza to niezmienne, zaprojektowane przez człowieka reguły odwzorowujące genotyp w jego znaczenie (*external embryogeny*). Więcej informacji – patrz późniejszy wykład o automatycznym (ewolucyjnym) projektowaniu.

Embriogeneza – ilustracja



Embriogeneza – opis ilustracji

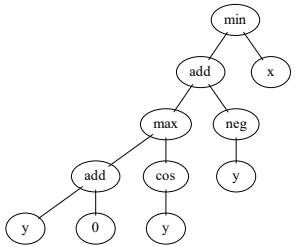
Schemat na poprzednim slajdzie pokazuje związki między przestrzenią genetyczną, fenetyczną, i krajobrazem przystosowania.

Zauważ, że różne embriogenezy (zatem też różne zbiory fenotypów, ich topologie i krajobrazy przystosowania) mogą być wynikiem:

1. różnych reprezentacji genetycznych i przeznaczonych dla nich operatorów (na schemacie są pokazane dwie),
2. różnych interpretacji (pokazano trzy) genów w ramach danej reprezentacji,
3. tej samej reprezentacji genetycznej i tej samej interpretacji genów, ale różnych operatorów mutacji/sąsiedztwa (schemat tej sytuacji nie pokazuje).

Programowanie genetyczne (*genetic programming*)

Programowanie genetyczne* służy do optymalizacji wyrażeń i programów. Cechą charakterystyczną jest drzewiasta reprezentacja rozwiązań pozwalająca na zakodowanie programów, choć istnieje też mniej popularny wariant z kodowaniem liniowym.**



Rysunek: Wyrażenie `min(add(max(add(y, 0), cos(y)), neg(y)), x))` czyli `min(max(y + 0, cos(y)) + (-y), x)` czyli `min(x, max(y, cos(y)) - y)`.

*Darmowa książka:
http://www0.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/poli08_fieldguide.pdf

**https://en.wikipedia.org/wiki/Linear_genetic_programming

Programowanie genetyczne – budowa drzewa

Wyrażenia przechowywane w populacji składają się z elementów należących do zbioru funkcji ***F*** (węzły drzewa) i zbioru terminali ***T*** (liście drzewa). Zbiory te można dobrać wedle potrzeb i dostosować do rozwiązywanego problemu. Przestrzeń rozwiązań stanowią wszystkie kombinacje wyrażeń złożonych z elementów obu zbiorów.

Zbiór funkcji	
Rodzaj	Przykłady
Arytmetyczne	$+$, *, /
Matematyczne	sin, cos, exp
Logiczne	AND, OR, NOT
Warunkowe	IF-THEN-ELSE
Pętle	FOR, REPEAT

Zbiór terminali	
Rodzaj	Przykłady
Zmienne	$\vec{x}$, y, x172
Stałe	3, 0.45, π
Procedury	rand, go_left, read_proximity

„Procedury" mogą byc funkcjami lub akcjami bezargumentowymi.

Programowanie genetyczne – przykład użycia biblioteki DEAP

```
from deap import gp
# https://deap.readthedocs.io/en/master/tutorials/advanced/gp.
    html
# https://deap.readthedocs.io/en/master/examples/gp_symbreg.html

pset = gp.PrimitiveSet("MAIN", 2) # two arguments (x and y)
pset.addPrimitive(operator.add, 2)
pset.addPrimitive(operator.sub, 2)
pset.addPrimitive(operator.mul, 2)
pset.addPrimitive(operator.neg, 1)
pset.addPrimitive(min, 2)
pset.addPrimitive(max, 2)
pset.addPrimitive(math.cos, 1)
pset.addPrimitive(math.sin, 1)
pset.addEphemeralConstant("rand101", lambda: random.randint
    (-1,1))
pset.renameArguments(ARG0='x')
pset.renameArguments(ARG1='y')
```

Pożądane są dwie cechy zbiorów $\boldsymbol{F}$ i $\boldsymbol{T}$:

1. *domknięcie* (ang. *closure*) – każda funkcja działa dla dowolnych wartości i typów argumentów zwracanych przez dowolną funkcję lub terminal,
2. *wystarczalność* (ang. *sufficiency*) – dostępne w obu zbiorach elementy pozwalają na zbudowanie rozwiązania stawianego problemu.

Zastanów się, jak można zapewnić właściwość *domknięcia*.

Zastanów się, jak można zapewnić właściwość *domknięcia*.

Właściwość *domknięcia* można uzyskać zabezpieczając odpowiednio funkcje (np. licząc zawsze wartość bezwzględną dla argumentu pierwiastka) albo karając niepoprawne wyrażenia (obniżając im wartość dopasowania). Albo ustawić flagi procesora/programu/systemu operacyjnego tak, żeby wszelkie operacje nie powodowały wyjątków... (historia jako przykład: długa symulacja numeryczna pod Linuxem i różnica tej samej symulacji pod Windows).

```
def protectedDiv(left, right):
    try:
        return left / right
    except ZeroDivisionError:
        return 1

pset.addPrimitive(protectedDiv, 2)
```

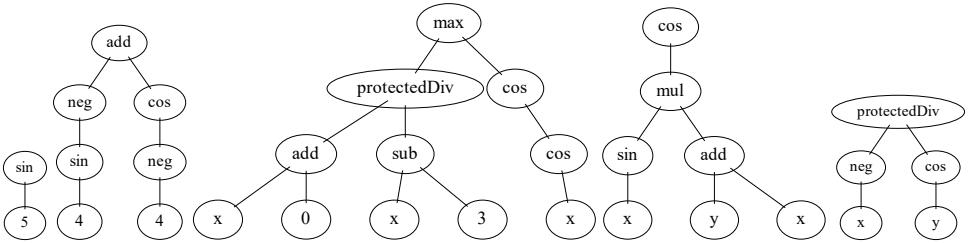
Jeśli nie zapewnimy *wystarczalności*, GP będzie starało się znaleźć (najlepsze) przybliżenie rozwiązania za pomocą dostępnych środków.

Podstawowe metody tworzenia populacji początkowej

Podstawowe metody tworzenia populacji początkowej

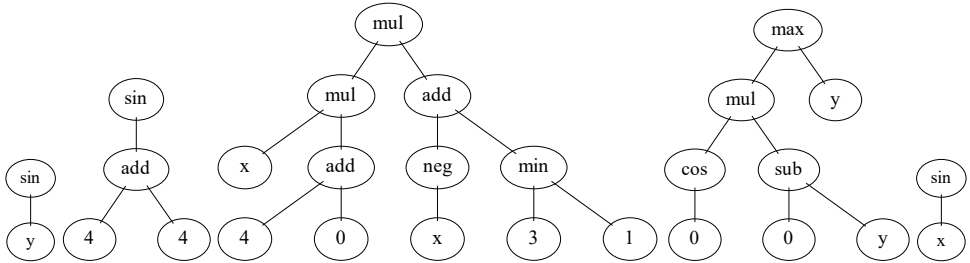
- *Full*: wybieraj węzły z $\boldsymbol{F}$ jeśli głębokość jest poniżej ustalonego progu, a jeśli nie, to z $\boldsymbol{T}$. Wszystkie drzewa będą miały taką samą głębokość – przykłady na kolejnym slajdzie.
- *Grow*: wybieraj węzły z $\boldsymbol{F} \cup \boldsymbol{T}$ jeśli głębokość jest poniżej ustalonego progu, a jeśli nie, to z $\boldsymbol{T}$. Drzewa będą miały różną głębokość i kształt – przykłady na slajdzie następnym.
- *Ramped half-and-half*: połowa populacji metodą *full*, połowa metodą *grow* – zapewnia zróżnicowanie populacji początkowej.

Tworzenie osobników metodą *Full*



Rysunek: Pięć osobników wygenerowanych metodą *Full*, `gp.genFull(pset,1,3)` (DEAP wymaga dwóch parametrów, nie jednego) dla $\boldsymbol{T}=\{x, y, 0, 1, 2, 3, 4, 5\}$.

Tworzenie osobników metodą *Grow*



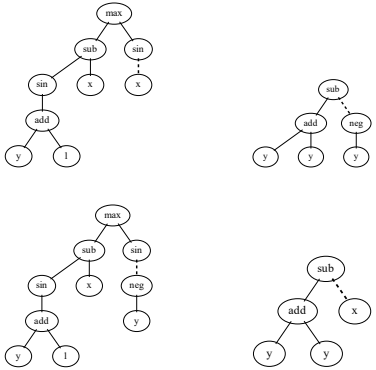
Rysunek: Pięć osobników wygenerowanych metodą *Grow*, `gp.genGrow(pset,1,3)`, dla $\boldsymbol{T}$={x, y, 0, 1, 2, 3, 4, 5}. W metodzie `genGrow()` DEAP'a nie ma sensu ustawienie argumentów min_depth i max_depth na taką samą wartość, bo generowane drzewa będą miały wtedy wszystkie liście na tej samej głębokości – tak samo, jakby drzewa generowała metoda `genFull()`.

Krzyżowanie

Krzyżowanie

Krzyżowanie w GP to najczęściej wymiana losowo wybranych poddrzew u obu rodziców.

```
toolbox.register("mate", gp.cxOnePoint)
```



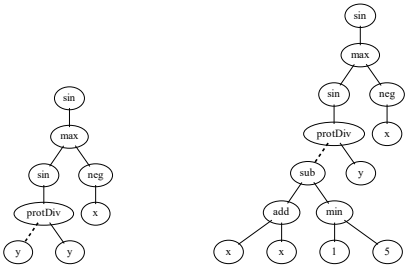
Rysunek: Krzyżowanie w GP. U góry rodzice wygenerowani metodą `gp.genGrow(pset,2,4)`. Na dole dzieci utworzone metodą `gp.cxOnePoint(parent1,parent2)`.

Mutacja

Mutacja

Typowa mutacja to wybranie losowego miejsca w oryginalnym drzewie i zastąpienie poddrzewa nowym, wygenerowanym jedną z powyżej opisanych metod.

```
toolbox.register("expr_mut", gp.genFull, min_=0, max_=2)
toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr_mut,
    pset=pset)
```



Rysunek: Po lewej przodek wygenerowany metodą `gp.genGrow(pset,2,5)`. Po prawej mutant utworzony metodą `gp.mutUniform(parent, toolbox.expr_mut, pset=pset)`, przy wcześniejszym `toolbox.register("expr_mut", gp.genFull, min_=0, max_=2)`.

„Puchnięcie” rozwiązań

„Puchnięcie" rozwiązań

Żeby zabezpieczyć się przed niekontrolowanym „puchnięciem" wyrażeń (ang. *bloat*), można włączyć do oceny rozwiązań kary za rozmiar wyrażenia lub zastosować ograniczenia na głębokość drzewa.

```
toolbox.decorate("mate", gp.staticLimit(key=operator.attrgetter(
    "height"), max_value=13))
toolbox.decorate("mutate", gp.staticLimit(key=operator.
    attrgetter("height"), max_value=11))
```

Ponieważ wyrażenia lub programy generowane przez GP mają przypadkowy charakter, trudno byłoby je uruchamiać bezpośrednio w systemie operacyjnym – bezpieczniej jest je interpretować lub oceniać w wirtualnym środowisku (np. w maszynie wirtualnej albo „sandboxie"). Ocena jakości rozwiązania wymaga najczęściej jego wyliczenia lub użycia w wielu sytuacjach (różne wartości argumentów, różne lokalizacje robota, itp.).

```
# Exception: MemoryError - Error in tree evaluation: Python
    cannot evaluate a tree higher than 90.
```

Można stosować omówione wcześniej, standardowe metody selekcji, ale często lepszych wyników dostarcza selekcja Lexicase. W tej metodzie nie agreguje się błędów każdego rozwiązania na wszystkich testach do jednej liczby. Zamiast tego, aby wybrać jednego osobnika z populacji, najpierw losujemy kolejność testów, a potem wybieramy te osobniki, które na pierwszym teście (z tej losowej kolejności) uzyskały najlepszy wynik w populacji. Jeśli takich osobników jest więcej niż jeden, rozważamy wśród nich tak samo drugi test, potem ewentualnie trzeci, itd.

Dyskusja: jak takie podejście różni się od metod selekcji z ewolucyjnej optymalizacji wielokryterialnej takich jak NSGA czy SPEA?

Dyskusja: krajobraz przystosowania, globalna wypukłość i skuteczność optymalizacji w GP.

Regresja symboliczna to typowe zastosowanie GP, w którym szukamy funkcji opisującej możliwie dokładnie zadane punkty. O ile w tradycyjnych metodach regresji postać szukanej funkcji jest ustalona (poszukujemy tylko współczynników), o tyle w GP można łatwo manipulować postacią funkcji, a nawet szukać pewnych klas funkcji lub dowolnych funkcji (stąd ta metoda regresji nazywana jest *symboliczną*).

Sterowanie postacią szukanego wyrażenia odbywa się za pomocą odpowiedniego doboru elementów zbioru funkcji $\boldsymbol{F}$ i terminali $\boldsymbol{T}$, oraz narzucaniem ewentualnych ograniczeń na głębokość drzewa, liczbę wystąpień funkcji ze zbioru $\boldsymbol{F}$, itp.

Regresja symboliczna – szukamy $f(x) = x^2 - x$

Przykładowy eksperyment #1: Znajdź wyrażenie najlepiej opisujące zbiór punktów należących do funkcji $f(x) = x^2 - x$. Pamiętajmy, że w praktyce funkcja ta jest nieznana i chcemy ją odkryć! Do dyspozycji GP dajemy to co w przykładowych źródłach powyżej, czyli oprócz x operatory $\text{neg}, +, -, *, /, \max, \min, \sin, \cos$, i dodatkowo też stałe $-1, 0, 1$.

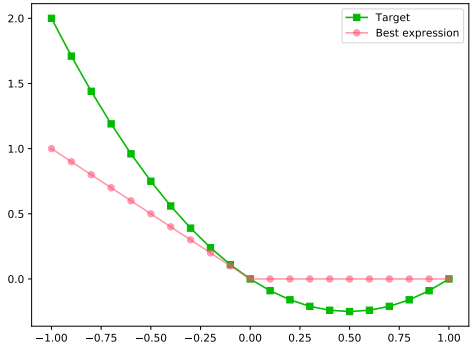
Regresja symboliczna – przykład użycia biblioteki DEAP


```
def target_function(x):
    return x**2 - x  # in a real application, this is what we
     look for!

def eval_expr(individual, points):
    # transform the tree expression into a callable function
    func = toolbox.compile(expr=individual)
    # evaluate the mean squared error between the expression and
     the target function
    sqerrors = ((func(x) - target_function(x))**2 for x in points
    )
    return math.fsum(sqerrors) / len(points),

toolbox.register("evaluate", eval_expr, points=[x/10. for x in
    range(-10,11)])
```

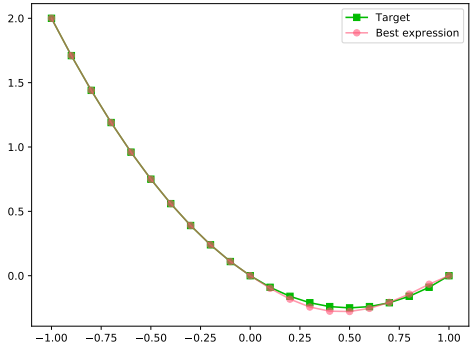
Szukamy $f(x) = x^2 - x$. Pierwsze pokolenie



Rysunek: Rozwiązanie najlepsze w pierwszym pokoleniu (czyli w losowo wygenerowanej populacji).

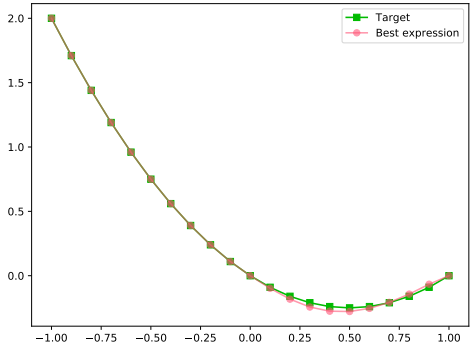
```
mul(min(0, x), neg(1))
```

Szukamy $f(x) = x^2 - x$. Ostatnie pokolenie



Rysunek: Rozwiązanie najlepsze po zakończeniu ewolucji.

Szukamy $f(x) = x^2 - x$. Ostatnie pokolenie



Rysunek: Rozwiązanie najlepsze po zakończeniu ewolucji.

```
sub(x, add(min(min(min(0, x), mul(0, add(0, max(1, 0)))),
add(x, max(x, mul(add(0, x), neg(x))))), max(add(min(min(x, 0),
add(min(sin(x), x), max(sin(x), add(add(0, 0), sin(sin(sin(x)))))))),
max(sin(add(min(sin(x), sin(sin(sin(sin(x))))), max(sin(sin(x)), -1))),
x)), x)))
```

Szukamy $f(x) = x^2 - x$. Inne parametry...

Po zwiększeniu rozmiaru populacji i liczby pokoleń: `mul(add(-1, x), min(x, x))`. Podobnie, po ograniczeniu złożoności wyrażeń (intensyfikacja przeszukiwania prostych wyrażeń): `mul(add(-1, x), protectedDiv(x, 1))`.

Przykładowy eksperyment #2: Znajdź układ logiczny realizujący funkcję XOR, czyli $\{x_1, x_2, y\} = \{(0,0,0); (0,1,1); (1,0,1); (1,1,0)\}$.

W tym eksperymencie GENERATIONS=100 oraz POPSIZE=150, a w przypadku niepowodzenia – kolejna próba z POPSIZE=1500.

Regresja symboliczna – szukamy XOR: funkcje i terminale


```
def nand(input1, input2):
    return not(input1 and input2)

def if_then_else(input, output1, output2):
    return output1 if input else output2

pset = gp.PrimitiveSetTyped("main", [bool, bool], bool) # let's
    use strongly-typed GP as an example
pset.addPrimitive(operator.xor, [bool, bool], bool)
pset.addPrimitive(operator.or_, [bool, bool], bool)
pset.addPrimitive(operator.and_, [bool, bool], bool)
pset.addPrimitive(operator.not_, [bool], bool)
pset.addPrimitive(nand, [bool, bool], bool) # custom
pset.addPrimitive(if_then_else, [bool, bool, bool], bool) #
    custom
pset.addTerminal(True, bool)

pset.renameArguments(ARG0="x1")
pset.renameArguments(ARG1="x2")
```

Regresja symboliczna – szukamy XOR: ocenianie


```
def eval_expr(individual):
    # transform the tree expression into a callable function
    func = toolbox.compile(expr=individual)
    # evaluate the error between the expression and the target
    function
    err = 0
    for x1 in (False,True):
        for x2 in (False,True):
            target = x1^x2
            actual = func(x1,x2)
            if target != actual:
               err += 1
    return err,
```

Regresja symboliczna – szukamy XOR: wyniki

- Wszystkie operatory oraz stała `True` jak w kodzie powyżej: `xor(if_then_else(x2, True, x2), x1)`

Regresja symboliczna – szukamy XOR: wyniki

- Wszystkie operatory oraz stała `True` jak w kodzie powyżej:
  `xor(if_then_else(x2, True, x2), x1)`

- Samo `if-then-else`: brak idealnego rozwiązania (najmniejszy błąd $= 1$)

Regresja symboliczna – szukamy XOR: wyniki

- Wszystkie operatory oraz stała `True` jak w kodzie powyżej:
  `xor(if_then_else(x2, True, x2), x1)`

- Samo `if-then-else`: brak idealnego rozwiązania (najmniejszy błąd $= 1$)
- Tylko `if-then-else` oraz `not`:
  `if_then_else(x1, not_(x2), x2)`

Regresja symboliczna – szukamy XOR: wyniki

- Wszystkie operatory oraz stała `True` jak w kodzie powyżej:
  `xor(if_then_else(x2, True, x2), x1)`

- Samo `if-then-else`: brak idealnego rozwiązania (najmniejszy błąd $= 1$)
- Tylko `if-then-else` oraz `not`:
  `if_then_else(x1, not_(x2), x2)`

- Tylko `not` oraz `and`: brak idealnego rozwiązania (najmniejszy błąd $= 1$)

Regresja symboliczna – szukamy XOR: wyniki

- Wszystkie operatory oraz stała `True` jak w kodzie powyżej:
  `xor(if_then_else(x2, True, x2), x1)`

- Samo `if-then-else`: brak idealnego rozwiązania (najmniejszy błąd $= 1$)
- Tylko `if-then-else` oraz `not`:
  `if_then_else(x1, not_(x2), x2)`

- Tylko `not` oraz `and`: brak idealnego rozwiązania (najmniejszy błąd $= 1$)
- Samo `nand`:
  `nand(nand(nand(x2, x1), x2), nand(x1, nand(x1, x2)))`

Regresja symboliczna – szukamy XOR: wyniki

- Wszystkie operatory oraz stała `True` jak w kodzie powyżej:
  `xor(if_then_else(x2, True, x2), x1)`

- Samo `if-then-else`: brak idealnego rozwiązania (najmniejszy błąd $= 1$)
- Tylko `if-then-else` oraz `not`:
  `if_then_else(x1, not_(x2), x2)`

- Tylko `not` oraz `and`: brak idealnego rozwiązania (najmniejszy błąd $= 1$)
- Samo `nand`:
  `nand(nand(nand(x2, x1), x2), nand(x1, nand(x1, x2)))`

- Trójka `and`, `or`, `not`:
  `and_(not_(and_(x2, x1)), or_(x1, x2))`

Dyskusja: czy warto stosować upraszczanie wyrażeń podczas ewolucji?

Dyskusja: czy warto stosować upraszczanie wyrażeń podczas ewolucji?

Dyskusja: w jakich dziedzinach GP ma szansę konkurować z człowiekiem, w jakich go przewyższyć, a w jakich nie ma szans? Dlaczego?

Semantyczne GP (semantyka = zbiór efektów działania osobnika na zbiorze testów) oraz geometryczne semantyczne GP (operatory genetyczne uwzględniają topologię przestrzeni semantyk) [Bak+19].

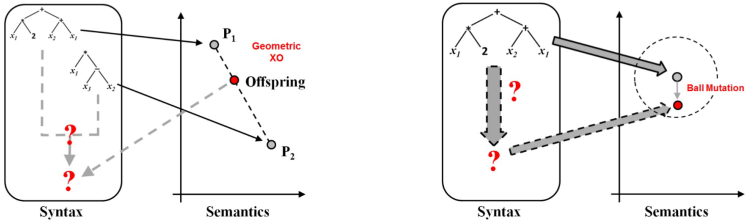


Figure 1: Geometric semantic crossover (plot (a)) (respectively geometric semantic mutation (plot (b))) performs a transformation on the syntax of the individual that corresponds to geometric crossover (respectively geometric mutation) on the semantic space. In this figure, the unrealistic case of a bidimensional semantic space is considered, for simplicity.

Por. wcześniejsze przypomnienie FDC, DPX, "Jak świadomie tworzyć (projektować) skuteczne operatory krzyżowania?" oraz embriogenezy/mapowania.

Szukamy algorytmu uczenia sieci neuronowej

Przykładowy eksperyment #3: Znajdź algorytm uczenia sieci neuronowej...

Architektura ewolucyjna: „ewolucja regularyzowana” (Rys. 2) [Rea+20].
Odkrycia ewolucji – Rys. 6:

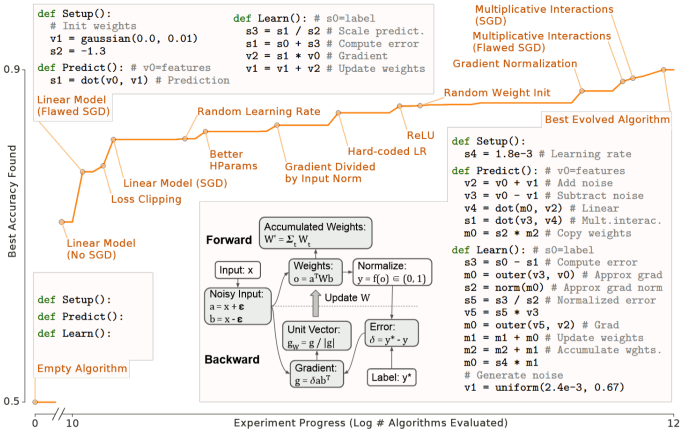


Figure 6: Progress of one evolution experiment on projected binary CIFAR-10. Callouts indicate some beneficial discoveries. We also print the code for the initial, an intermediate, and the final algorithm. The last is explained in the flow diagram. It outperforms a simple

Hiperheurystyki i samo-programujące się algorytmy

*http://en.wikipedia.org/wiki/Hyper-heuristic

Struktura algorytmu ewolucyjnego (rodzaj selekcji, krzyżowania, mutacji, ...) może podlegać kontroli GP (czyli ewolucyjnemu ulepszaniu) [Wąs97; BT96; OG03; Olt05]. GP pozwala na „zbudowanie" algorytmu optymalizacji z modułów, przy czym możliwe są nietypowe architektury: wiele rodzajów mutacji, nietypowe operatory działające na części populacji, wielokrotna selekcja w jednym kroku itp. – zależnie od stopni swobody GP.

Wyniki – lepsze od tradycyjnego, ustalonego AE, ale kosztem...

Porównaj: twierdzenie *No Free Lunch* oraz hiperheurystyki* przeszukujące przestrzeń heurystyk i ich kombinacji [Ros05; ÖBK08; Bur+10].

*http://en.wikipedia.org/wiki/Hyper-heuristic

A gdyby tak...

Jeśli masz czas i lubisz SF, przeczytaj
https://www.teamten.com/lawrence/writings/coding-machines/.

Bibliografia I

[Bak+19] Illya Bakurov i in. "A regression-like classification system for geometric semantic genetic programming". W: *Proceedings of the 11th International Joint Conference on Computational Intelligence (IJCCI)*. T. 1. 2019, s. 40–48. URL: https://run.unl.pt/bitstream/10362/87064/1/Regression_like_Classification_System_Geometric_Semantic_Genetic.pdf.

[Ben99] Peter Bentley. *Evolutionary design by computers*. Morgan Kaufmann, 1999.

[BT96] Andreas Bölte i Ulrich Wilhelm Thonemann. "Optimizing simulated annealing schedules with genetic programming". W: *European Journal of Operational Research* 92.2 (1996), s. 402–416. ISSN: 0377-2217. DOI: 10.1016/0377-2217(94)00350-5.

[Bul01] Seth Bullock. "Smooth operator? Understanding and visualising mutation bias". W: *European Conference on Artificial Life*. Springer. 2001, s. 602–612. DOI: 10.1007/3-540-44811-X_68.

[Bul99] Seth Bullock. "Are artificial mutation biases unnatural?" W: *European Conference on Artificial Life*. Springer. 1999, s. 64–73. DOI: 10.1007/3-540-48304-7_11. URL: https://eprints.soton.ac.uk/261452/1/10.1.1.40.2753.pdf.

[Bur+10] E. K. Burker i in. "A classification of hyper-heuristic approaches". W: *Handbook of Metaheuristics* (2010), s. 449–468.

[Dus+24] Arkadiy Dushatskiy i in. "Parameterless gene-pool optimal mixing evolutionary algorithms". W: *Evolutionary computation* 32.4 (2024), s. 371–397. URL: https://arxiv.org/pdf/2109.05259.

[Gol+93] David E. Goldberg i in. *Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms*. Sprawozdanie techniczne 93004. 1993, s. 1–16. URL: http://repository.ias.ac.in/81677/1/110-a.pdf.

Bibliografia II

[GT12] Brian W. Goldman i Daniel R. Tauritz. "Linkage tree genetic algorithms: variants and analysis". W: *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. 2012, s. 625–632. URL: http://www.cmap.polytechnique.fr/~nikolaus.hansen/proceedings/2012/GECCO/proceedings/p625.pdf.

[JTW04] E. D. de Jong, D. Thierens i R. A. Watson. "Hierarchical genetic algorithms". W: *Lecture notes in computer science* (2004), s. 232–241. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=20bde9fe796a5e15c9007c7dc770b35aa731751d.

[KM25] Maciej Komosinski i Konrad Miazga. "GOM-Based Compatible Substitutions Optimization for Variable-Length Representation Gray-Box Problems". W: *Genetic and Evolutionary Computation Conference (GECCO '25 Companion)*. Malaga, Spain: ACM, 2025, s. 571–574. DOI: 10.1145/3712255.3726717. URL: http://www.framsticks.com/files/common/GOM-BasedCompatibleSubstitutionsOptimization.pdf.

[Mic96] Z. Michalewicz. *Algorytmy genetyczne + struktury danych = programy ewolucyjne*. Wydawnictwa Naukowo-Techniczne, 1996.

[ÖBK08] E. Özcan, B. Bilgin i E. E. Korkmaz. "A comprehensive analysis of hyper-heuristics". W: *Intelligent Data Analysis* 12.1 (2008), s. 3–23.

[OG03] Mihai Oltean i Crina Groşan. "Evolving evolutionary algorithms using multi expression programming". W: *European Conference on Artificial Life*. Springer. 2003, s. 651–658. URL: https://www.researchgate.net/profile/Mihai_Oltean2/publication/226167912_Evolving_Evolutionary_Algorithms_Using_Multi_Expression_Programming/links/55dac32308aed6a199aaf916.pdf.

[Olt05] Mihai Oltean. "Evolving evolutionary algorithms using linear genetic programming". W: *Evolutionary Computation* 13.3 (2005), s. 387–410. URL: https://mihaioltean.github.io/oltean_mit_draft_2005.pdf.

Bibliografia III

[PKF21] Michal W. Przewozniczek, Marcin M. Komarnicki i Bartosz Frej. "Direct linkage discovery with empirical linkage learning". W: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2021, s. 609–617. URL: https://www.cs.put.poznan.pl/mkomosinski/lectures/optimization/extras/DLED-epistasis-GECCO2021.pdf.

[PTK23] Michal W. Przewozniczek, Renato Tinós i Marcin M. Komarnicki. "First Improvement Hill Climber with Linkage Learning – on Introducing Dark Gray-Box Optimization into Statistical Linkage Learning Genetic Algorithms". W: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '23*. ACM, 2023, s. 946–954. DOI: 10.1145/3583131.3590495.

[Rea+20] Esteban Real i in. "AutoML-Zero: Evolving machine learning algorithms from scratch". W: *International Conference on Machine Learning*. PMLR. 2020, s. 8007–8019. URL: https://proceedings.mlr.press/v119/real20a/real20a.pdf.

[Rec84] Ingo Rechenberg. "The Evolution Strategy. A Mathematical Model of Darwinian Evolution". W: *Synergetics – From Microscopic to Macroscopic Order*. Oprac. Eckart Frehland. Berlin, Heidelberg: Springer Berlin Heidelberg, 1984, s. 122–132. DOI: 10.1007/978-3-642-69540-7_13.

[Ros05] P. Ross. "Hyper-heuristics". W: *Search Methodologies* (2005), s. 529–556.

[Rot06] Franz Rothlauf. *Representations for genetic and evolutionary algorithms*. Springer, 2006. DOI: 10.1007/3-540-32444-5.

[SP97] Rainer Storn i Kenneth Price. "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces". W: *Journal of Global Optimization* 11.4 (1997), s. 341–359. ISSN: 1573-2916. DOI: 10.1023/A:1008202821328.

Bibliografia IV

[TB13] Dirk Thierens i Peter A. N. Bosman. "Hierarchical problem solving with the linkage tree genetic algorithm". W: *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. 2013, s. 877–884. URL: https://homepages.cwi.nl/~bosman/publications/2013_hierarchicalproblemsolving.pdf.

[Thi18] Dirk Thierens. *Model-Based Evolutionary Algorithms, Part 2: Linkage Tree Genetic Algorithm*. 2018. URL: https://ics-websites.science.uu.nl/docs/vakken/ea/slides/LTGA_GOMEA.pdf.

[TYH99] Shigeyoshi Tsutsui, Masayuki Yamamura i Takahide Higuchi. "Multi-parent recombination with simplex crossover in real coded genetic algorithms". W: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation – Volume 1*. 1999, s. 657–664. URL: https://www.researchgate.net/profile/Shigeyoshi-Tsutsui/publication/243776468_Multi-parent_recombination_with_simplex_crossover_in_real-coded_genetic_algorithms/links/00463531fa92bd4738000000/Multi-parent-recombination-with-simplex-crossover-in-real-coded-genetic-algorithms.pdf.

[Wąs97] Piotr Wąsiewicz. "Self-programming of Algorithms". W: *Proceedings of the 2nd National Conference on Evolutionary Computation and Global Optimization*. Rytro: Oficyna Wydawnicza Politechniki Warszawskiej, wrzesień 1997, s. 293–297.

Fundusze Europejskie
Polska Cyfrowa

Rzeczpospolita Polska

Unia Europejska
Europejski Fundusz Rozwoju Regionalnego