

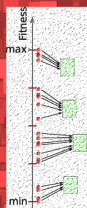
Algorytmy ewolucyjne

[cytowanie tego skryptu]

Maciej Komosiński

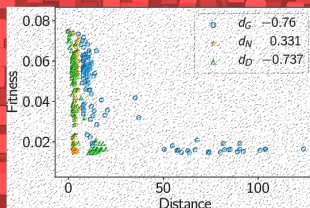
2025

CC BY-NC 4.0

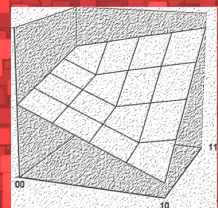
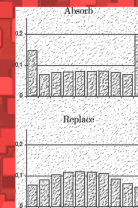
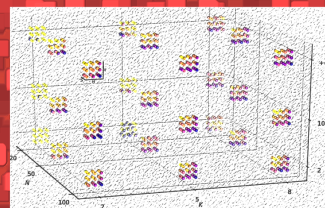


$$p_i = \frac{f_i}{\sum_{j=1}^n f_j} = \frac{f_i}{n\mu}$$

```
*** 0 *** 0 ***
*** 0 *** 1 ***
*** 1 *** 0 ***
*** 1 *** 1 ***
```



$$P = \begin{bmatrix} I_a & 0 \\ R & Q \end{bmatrix}$$



Materiały pomocne w przypomnieniu tematyki ewolucyjnej poruszanej na wykładach.

<http://www.cs.put.poznan.pl/mkomosinski/site/>

Spis treści

1	Optymalizacja	3
1.1	Przeszukiwanie wykorzystujące sąsiedztwo pojedynczego rozwiązania	3
1.2	Dobór wartości parametrów; zastosowanie interakcyjne i wsadowe	4
2	Algorytmy ewolucyjne	6
2.1	Podział	6
2.2	Algorytmy genetyczne (genetic algorithms)	7
2.2.1	Budowa algorytmu i parametry	8
2.2.2	Selekcja	8
2.2.3	Skalowanie	14
2.2.4	Krzyżowanie	18
2.2.5	Mutacja	19
2.2.6	Podsumowanie parametryzacji AG	21
2.2.7	Twierdzenie o schematach	23
2.2.8	Problemy zawodne (trudne dla AG) i twierdzenie „No Free Lunch”	23
2.2.9	Nieporządkny algorytm genetyczny	28
2.2.10	Hierarchiczny algorytm genetyczny	29
2.2.11	Empiryczna i teoretyczna ocena AG	32
2.2.12	Neutralność	32
2.2.13	Dryf genetyczny	33
2.2.14	Przykład analizy teoretycznej: brak mutacji	34
2.3	Strategie ewolucyjne (<i>evolutionary strategies</i>)	36
2.4	Ewolucja różnicowa (<i>differential evolution</i>)	38
2.5	Programowanie ewolucyjne (<i>evolutionary programming</i>)	39
2.5.1	Reprezentacja liczb rzeczywistych	40
2.5.2	Krzyżowanie i mutacja a globalna wypukłość	43
2.5.3	Embriogeneza	44
2.6	Programowanie genetyczne (<i>genetic programming</i>)	48

2.6.1	Regresja symboliczna	54
2.6.2	Hiperheurystyki i samo-programujące się algorytmy	60
2.7	Mechanizmy inspirowane naturą	61
2.7.1	Reprezentacja	61
2.7.2	Operatory	62
2.7.3	Funkcja celu i logika algorytmu	63
2.8	Systemy klasyfikatorowe (CFS/LCS/GBML)	66
2.8.1	Input and output interfaces	70
2.8.2	Main cycle	72
2.8.3	<i>Learning Classifier Systems</i> (LCS)	73
2.8.4	Good and bad classifiers	74
2.8.5	The need for competition	74
2.8.6	Quality of classifiers	74
2.8.7	Adaptation by credit assignment	75
2.8.8	The Bucket Brigade algorithm	75
2.8.9	Adaptation by rule discovery	77
2.8.10	Podsumowanie	79
2.9	Inne techniki w AE	79
2.9.1	Wiele kryteriów	79
2.9.2	Wzbogacanie wiedzę	79
2.9.3	Sterowanie różnorodnością i techniki jakość–różnorodność	80
2.9.4	Obsługa ograniczeń	81
2.10	Równoległe algorytmy ewolucyjne	81
2.11	Jak zrobić skuteczny AE? Na co zwrócić uwagę?	83
2.12	Architektury koewolucyjne	85
2.12.1	Kooperatywne	85
2.12.2	Konkurencyjne	85

Rozdział 1

Optymalizacja

1.1 Przeszukiwanie wykorzystujące sąsiedztwo pojedynczego rozwiązania

Zanim przejdziemy do omówienia inspirowanych biologicznie algorytmów optymalizacji (w tym algorytmów ewolucyjnych), zapoznamy się z działaniem podstawowych, prostszych algorytmów [\[url\]](#).

- Podstawowe zagadnienia dotyczące optymalizacji:
 - `OptWprowadzenie.pdf` (ostatnie slajdy) <https://youtu.be/uk-aWMH0Rvs>
 - `MetaheurystykiPodsumowanie.pdf` (porównywanie algorytmów)
- Idea algorytmów lokalnej optymalizacji: `LS.pdf` <https://youtu.be/oe94AHDQap0>
- Algorytm symulowanego wyżarzania (1983): `SA.pdf` <https://youtu.be/gX-X85dCib0>
- Algorytm przeszukiwania tabu (1986): `TS.pdf` <https://youtu.be/HsGJrSBFQNs>

Nieźle wprowadzenie do optymalizacji globalnej oraz opis algorytmów omawianych na tym przedmiocie zawiera książka dostępna online [\[Wei09\]](#).

Metody optymalizacji globalnej można podzielić pod względem własności determinizmu [\[ENS99\]](#):

- deterministyczne

- siatki (*grid methods, covering methods*): w przestrzeni rozwiązań tworzy się siatkę (równomierną lub nie – nierównomierna sprzyja lepszej eksploatacji, bo gęstość siatki zależy od jakości rozwiązań), którą się przeszukuje
- uogólnionego spadku (*generalized descent*)
 - * trajektorii cząstki (*trajectory methods*): cząstka (rozwiązanie) porusza się w polu energetycznym wynikającym z optymalizowanej funkcji; strategia takiego poruszania się, choć może być chaotyczna¹, jest deterministyczna
 - * kary (*penalty methods*): wielokrotnie uruchamia się metodę optymalizacji lokalnej zmieniając funkcję celu o człon kary, co zmienia krajobraz optymalizowanej funkcji i ma zapobiec zbieganiu do tego samego optimum lokalnego
- deterministyczna wersja TS (*tabu search*)
- niedeterministyczne
 - poszukiwań losowych: Monte Carlo (MC) które w czystej postaci jest algorytmem losowym; CRS (*Controlled Random Search*) – połączenie MC i metody sympleksu nieliniowego; SA (*simulated annealing*)
 - z wykorzystaniem stochastycznego modelu funkcji celu²
 - populacyjne, np. algorytmy ewolucyjne

...ale podczas omawiania rozdziału 2.2.2 przeprowadzimy dyskusję o tym, co to znaczy, że algorytm ma niedeterministyczne elementy, co ta własność wnosi istotnego i kiedy jest niezbędna.

1.2 Dobór wartości parametrów; zastosowanie interakcyjne i wsadowe

Podobnie jak w wielu algorytmach sztucznej inteligencji, optymalne wartości parametrów zależą od charakteru rozwiązywanego zadania, którego jednak a priori (i zwykle również a posteriori) nie znamy. Dobór zależy również od zastosowania – interakcyjnego (*on-line*) lub wsadowego (*off-line*). Przy podejściu wsadowym zainteresowani jesteśmy najlepszym znalezionym podczas pracy algorytmu rozwiązaniem. Przy podejściu interakcyjnym („bieżącym”) interesuje nas, by dany algorytm optymalizacji dawał jak najlepsze wyniki przez

¹Chaos nie oznacza losowości, a wysoką wrażliwość na warunki początkowe.

²https://en.wikipedia.org/wiki/Stochastic_optimization, https://en.wikipedia.org/wiki/Stochastic_programming

cały czas. Dla oceny działania w trybach *on-line* i *off-line* De Jong zaproponował specjalne wskaźniki [Gol03, str. 125]; wymyśl dwa najprostsze.

Rozdział 2

Algorytmy ewolucyjne

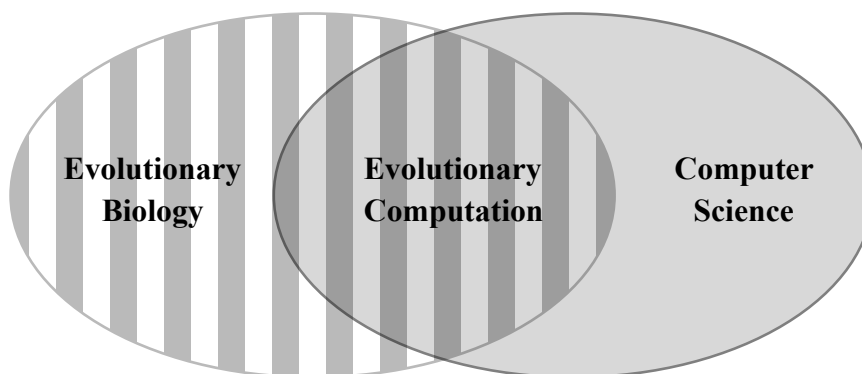
Materiały wideo do tego rozdziału:

<https://www.youtube.com/playlist?list=PLl33p3GENNQHztKdRM1LVGd85VTDb02Ub>

2.1 Podział

Evolutionary Computation (EC) / Algorithms (EA)

EA is based upon biological observations that date back to Charles Darwin's discoveries in the 19th century: the means of natural selection and the survival of the fittest, and theories of evolution.



Rysunek 2.1: Obliczenia ewolucyjne jako część informatyki i biologii.

- Genetic Algorithms (GA)
- Evolution Strategies (ES)

- Evolutionary Programming (EP)
- Genetic Programming (GP)
- Classifier Systems (CFS), Genetic-Based Machine Learning (GBML)
- Various coevolutionary architectures
- ...

GA: created by John Holland (1973, 1975), made famous by David Goldberg (1989)

EP: created by Lawrence Fogel (1963), developed by his son, David Fogel (1992)

ES: created by Ingo Rechenberg (1973), promoted by Thomas Bäck (1996)

GP: developed by John Koza (1992)

Zastosowania AE:

- optymalizacja funkcji
- badania operacyjne – szeregowanie, optymalizacja, ...
- wielokryterialne wspomaganie decyzji
- przetwarzanie obrazów, rozpoznawanie wzorców
- algorytmy adaptacyjne w grach
- sterowanie; robotyka; projektowanie ewolucyjne/ewolucja konstrukcji
- biologia – symulacje (gatunków, populacji, ...
- nauki społeczne – symulacje grup
- sztuczne życie
- ...

2.2 Algorytmy genetyczne (genetic algorithms)

Wszystkie genotypy są wektorami binarnymi o takiej samej, stałej długości. Jeśli problem optymalizacji wymaga innej reprezentacji (np. permutacji), wtedy rozwiązania potrzebują kodowania, dekodowania i ew. naprawy. Operatory genetyczne są nieświadome oryginalnej reprezentacji rozwiązań, więc mogą być takie same dla każdego problemu optymalizacji. Porównajmy to do natury... bardzo różne gatunki, ten sam podstawowy kod.

2.2.1 Budowa algorytmu i parametry

Główna pętla:

```
t := 0
inicjalizuj P(t)
ocień P(t)
dopóki (nie warunek-zakończenia)
{
    t := t + 1
    wybierz P(t) z P(t - 1)
    zmień P(t)
    oceń P(t)
}
```

Parametry:

- wielkość populacji *POPSIZE*
- prawdopodobieństwo krzyżowania *PXOVER*
- prawdopodobieństwo mutacji *PMUT*
- wybór kryterium stopu
- wybór mechanizmu selekcji (pozytywnej i ew. negatywnej)
- wybór wartości parametrów mechanizmu selekcji

Proste demo: <http://www.alife.pl/opt/p>

Sposób tworzenia kolejnej puli genów w AG: „*steady state*” (algorytm stanu ustalonego/inkrementacyjny) albo „*generational replacement*” (pokoleniowy). W „*steady state GA*” nie wszystkie osobniki ulegają modyfikacji – część trafia do kolejnego pokolenia nie zmieniona. W szczególnym przypadku zmieniamy tylko jednego osobnika i algorytm działa płynnie, bez wyraźnie zaznaczonych pokoleń, i szybciej korzysta z nowopowstałych pomysłów (możliwa jest szybsza zbieżność).

2.2.2 Selekcja

Rola selekcji.

- Do czego potrzebna jest selekcja w algorytmie ewolucyjnym?
- Co działałoby się, gdyby selekcja była czysto losowa?
- Co działałoby się, gdyby selekcja była deterministyczna i dawała równą szansę każdemu osobnikowi?
- Czy można wyrazić siłę presji selekcyjnej liczbowo?

Dyskusja o rodzajach niedeterminizmu i roli niedeterminizmu w algorytmach i w informatyce.

Rozważ cztery generatory sekwencji losowych: oparty o kolejne liczby i *modulo*, pseudolosowy [kiepski](#), pseudolosowy [dobry](#), i prawdziwie losowy.

```

int random1(int n)
    static int c=-1
    c++
    c=c%n
    return c

int random2(int n)
    static int c=0
    c=(c*...+...)%n
    return c

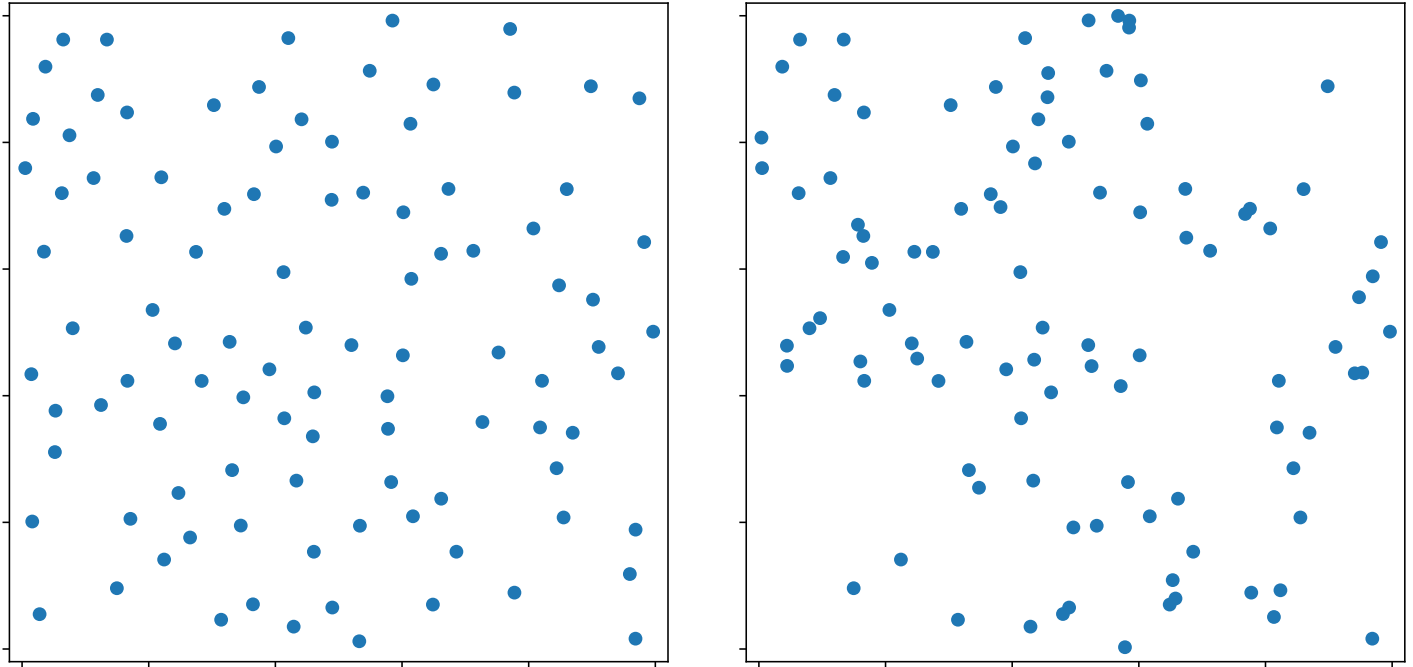
int random3(int n)
    //very
    //complicated
    //logic
    return ...

int random4(int n)
    return 🌞%n

```

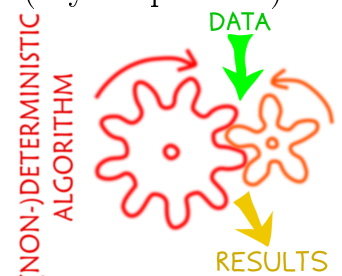
- przykład suplementacji (np. 4 substancje co 2 dni każda, nie znamy interakcji)
- przykład dawania prezentów
- przykład *dithering*'u sygnału ([dźwięk](#), [obraz](#), ...) i [zaokrąglania](#)
- i wreszcie przykład algorytmu...

Rola niedeterminizmu w algorytmice – „losowość”, dyskusja.



Rola niedeterminizmu w algorytmice – wnioski.

- wymyślasz jakiś algorytm, np. optymalizacyjny (albo inny). Co skłoni Cię do użycia w nim funkcji `random()`?
 - przykłady: *Greedy* i *Steepest* z wieloma sąsiadami o takiej samej jakości, indukcja drzew decyzyjnych z wieloma atrybutami o równej entropii, ...
 - spróbuj kompletne „udeterministycznić” SA. Czy będą jakieś negatywne konsekwencje?
- w jakich sytuacjach „nieskrępowana”, pełna losowość jest korzystna?
- dlaczego [przejmujemy się](#) ...10807777796980348..., ale nie ...108073517296980348...? Pierwsza sekwencja jest wprost z RNG, druga jest „poprawiona” (czy na pewno?)
- kiedy prawdziwa losowość jest preferowana względem dobrej pseudolosowości?
- i wreszcie: czy powinien dostać to finansowanie i dlaczego??



Od etapu reprodukcji oczekujemy powielenia osobników dobrych. Im większa będzie dominacja rozwiązań lepszych nad gorszymi (większy napór selekcyjny), tym mniejsza będzie różnorodność otrzymanej populacji. Te dwa aspekty selekcji (preferencja lepszych nad gorszymi oraz utrzymanie różnorodności) są ze sobą w pewnym stopniu sprzeczne, choć oba są też w pewnym stopniu pożądane.

Silę presji selekcyjnej można wyrazić liczbowo, np. dzieląc prawdopodobieństwo wyboru najlepszego osobnika w populacji przez prawdopodobieństwo wyboru osobnika przeciętnego (mającego przystosowanie równe medianie przystosowań w populacji).

Sterowanie naporem selekcyjnym odbywa się np. za pomocą skalowania ocen osobników. Zbyt silny napór doprowadzi do przedwczesnej zbieżności (do optimum lokalnego), ponieważ osobniki najlepsze w danym momencie uzyskają przewagę liczebną i zdominują pozostałe rozwiązania. Z kolei niski napór selekcyjny zapewni dużą różnorodność osobników w populacjach, co może spowodować nieefektywność całej ewolucji i upodobnienie jej do przeszukiwania losowego.

Oznaczmy przez f_i ocenę osobnika i -tego ($i = 1..POPSIZE$), a przez e_i – liczbę jego oczekiwanych kopii w nowej (kolejnej) populacji, $e_i = POPSIZE \cdot f_i / \sum f_j$.

Najbardziej popularne techniki selekcji [Gol03] to:

- Wybór losowy proporcjonalnie do jakości z powtórzeniami (tj. ze zwracaniem), nazywany potocznie zasadą ruletki: osobnikom przydzielane są pola na tarczy ruletki, których wielkość jest proporcjonalna do ich ocen f_i . Następnie kręci się tarczą „ruletki” $POPSIZE$ razy wybierając do nowej populacji wylosowanego osobnika. Tą samą zasadę realizuje metoda *stochastic universal sampling*¹, która zapewnia lepsze własności wyboru losowego.
- Wybór losowy według reszt bez powtórzeń: każdy osobnik otrzymuje tyle kopii w nowej populacji, ile wynosi część całkowita jego e_i . Pozostałe wolne miejsca zapełnia się podejmując losową decyzję, dla każdego osobnika z prawdopodobieństwem będącym częścią ułamkową jego e_i , czy ma trafić do nowej populacji. Przykład: 4 osobniki, $f=[1,3,5,6]$.
- Wybór według turniejów losowych: wybiera się losowo k osobników, a następnie zwy-

¹https://en.wikipedia.org/wiki/Stochastic_universal_sampling

ciężcę (osobnika o najwyższej ocenie) umieszcza się w nowej populacji i proces jest powtarzany aż do wypełnienia wszystkich miejsc w nowej populacji. Bardziej staranny wariant tej techniki dba o to, żeby każdy osobnik wziął udział w tej samej liczbie turniejów.

Inne techniki selekcji:

- Wybór deterministyczny według reszt: każdy osobnik otrzymuje tyle kopii w nowej populacji, ile wynosi część całkowita jego e_i , a pozostałe wolne miejsca w populacji zapełnia się w kolejności malejących części ułamkowych e_i osobników.
- Wybór losowy według reszt z powtórzeniami: każdy osobnik otrzymuje tyle kopii w nowej populacji, ile wynosi część całkowita jego spodziewanej liczby kopii (e_i). Pozostałe miejsca zapełnia się według zasady ruletki, wykalibrowanej według części ułamkowych e_i .
- Wybór porządkowy: osobnikom nadaje się rangi odpowiadające ich pozycjom w uszeregowaniu od najlepszego do najgorszego. Wybór następuje na podstawie funkcji prawdopodobieństwa określonej nie na ocenach osobników, a na ich pozycjach w rankingu. Stosuje się różne funkcje prawdopodobieństwa – linowe i nieliniowe, przy czym parametry tych funkcji pozwalają na sterowanie naporem selekcyjnym.

Zadanie: podziel wymienione 6 technik na dwie kategorie – zależnie od sposobu, w jaki wykorzystują wartości funkcji przystosowania.

Dodatkowe własności selekcji:

- Model elitarny (elitarystyczny): spełnia oczekiwanie, że proces selekcji nie powinien prowadzić do utraty najlepszego znalezionej dotąd osobnika. Jeśli taki osobnik nie trafia w naturalny (wynikający z zastosowanej metody selekcji) sposób do kolejnej populacji, włącza się go do niej i w ten sposób informacja o najlepszym dotychczasowym rozwiązaniu zostaje zawsze zachowana.
- Model ze współczynnikiem zatłoczenia (ze ściskiem): podobnie jak w naturze, gdzie gatunki wypełniające niszę ekologiczną muszą walczyć o ograniczone zasoby – w modelu ze ściskiem (ang. *crowding model*) nowe osobniki zastępują osobniki stare (z poprzedniej populacji) z uwzględnieniem ich podobieństwa, tzn. nowe osobniki zajmują miejsce najbardziej podobnych do nich starych osobników. Współczynnik ścisku (parametr) wpływa na sposób zastępowania osobników [DJ75; Mah92].

Meta-schematy selekcji:

W poniższych metodach selekcji, części populacji (podpopulacje) mogą być niezależnie przetwarzane – metody te mogą zatem pełnić również rolę schematu rozpraszania i zrównoleglania ewolucji.

- Model wyspowy: dzieli całą populację na podpopulacje, w których działa wybrany schemat selekcji (np. turniejowa, ruletkowa czy inny). Ewolucja odbywa się niezależnie na każdej wyspie, z okresową migracją części genotypów między wyspami. Ten model zwiększa zdolność eksploracji.
- Selekcja konwekcyjna: inaczej niż w tradycyjnym modelu wyspowym, podział na podpopulacje następuje wedle podobieństwa wartości funkcji celu rozwiązań. Selekcja konwekcyjna poprawia zdolność eksploracji AE dzięki odpowiedniemu zbalansowaniu presji selekcyjnej [KU17; KM18]. Animacje sposobu działania [tutaj](#). Dyskusja: jak zadziała ten meta-schemat w wariantach *EqualNumber* i *EqualWidth* [KM18, Rys. 3], kiedy w podpopulacjach selekcja będzie losowa (np. rozmiar turnieju = 1), w porównaniu do modelu wyspowego i do standardowego, jednopopulacyjnego AE z losową selekcją?

Wymienione techniki selekcji mają swoje wady i zalety; w szczególności pierwsza z nich – metoda ruletki – cechuje się wysokim rozrzutem losowym, co powoduje duże różnice pomiędzy faktycznie uzyskiwanymi a oczekiwanymi liczbami osobników. Stąd powstało wiele technik (jak np. wybór losowy według reszt bez powtórzeń) pozbawionych tej wady. Wybór metody ma duży wpływ na zachowanie algorytmu, w szczególności na zdolność przekraczania siodeł² podczas optymalizacji [CG97].

Niekiedy (zależnie od przyjętej architektury AG) oprócz zastosowania selekcji pozytywnej konieczne jest wykorzystanie również selekcji negatywnej. Jej rolą jest zrobienie miejsca w populacji na nowe genotypy: selekcja negatywna decyduje, które genotypy usunąć z populacji. Można stosować podobne mechanizmy, jak przy selekcji pozytywnej; dwie przykładowe, naiwne metody to usuwanie najgorszego i usuwanie losowego.

Przykładowe pytania

- Wymień i omów poznane metody selekcji.
- Jakie wady i jakie zalety mają poszczególne metody?

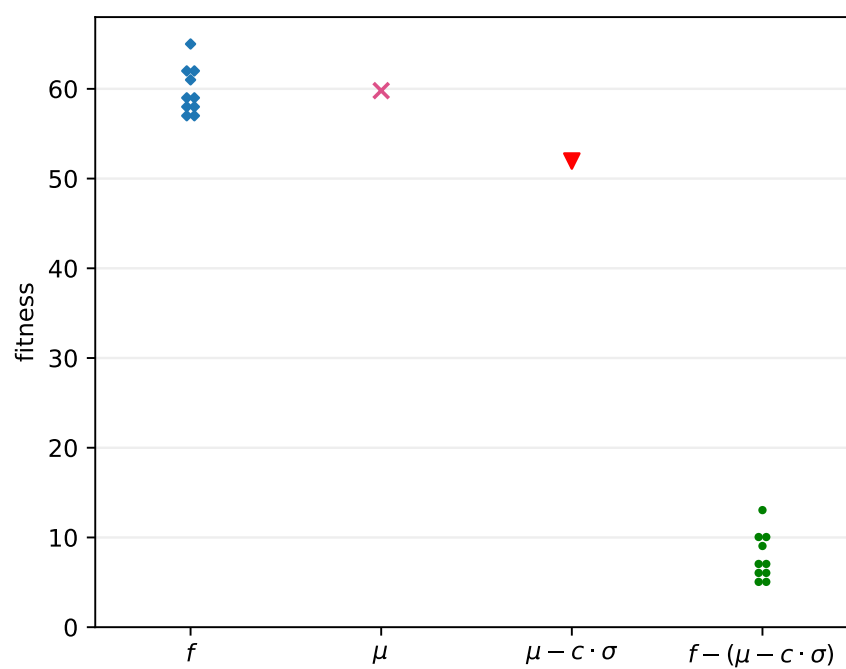
²https://pl.wikipedia.org/wiki/Punkt_siode%C5%82owy

2.2.3 Skalowanie

Dyskusja: skąd wynika potrzeba skalowania? Analiza zachowania selekcji ruletkowej na początku ewolucji oraz w późniejszych etapach.

- Skalowanie liniowe: $f' = af + b$. Współczynniki a i b dobiera się w ten sposób, by przystosowanie f' osobnika najlepszego było zadaną wielokrotnością (na przykład $2\times$) dopasowania osobnika „średniego”. Po skalowaniu mogą pojawić się ujemne wartości przystosowania – można wówczas wyzerować je lub dokonać innego przekształcenia liniowego.
- Skalowanie potęgowe: $f' = f^k$. Współczynnik k zależy od konkretnego rozwiązywanego zadania, stąd metoda ta nie jest specjalnie przydatna.
- Skalowanie σ -obcinające (obcinanie na poziomie zależnym od odchylenia standardowego). Wartości przystosowania zależą nie tylko od wartości pierwotnych ocen osobników, ale także od rozkładu ocen w populacji. Wyznacza się średnie dopasowanie populacji μ oraz odchylenie standardowe ocen w populacji σ , a przystosowanie (w przypadku maksymalizacji) $f' = f - (\mu - c \cdot \sigma)$. Ujemne wartości f' zastępuje się zerem. Współczynnik c określa napór selekcyjny: im większe c , tym mniejszy napór.

Np. 10 osobników o ocenach 57, 57, 58, 58, 59, 59, 61, 62, 62, 65. Średnia μ wynosi 59.8, a odchylenie standardowe $\sigma = 2.6$.



Rysunek 2.2: Wykres skalowania (z odcięciem na poziomie zależnym od odchylenia standardowego) dla współczynnika $c = 3$.

Zadanie obliczeniowe (i interpretacyjne): przed skalowaniem, prawdopodobieństwo wybrania osobnika i -tego p_i to iloraz wartości dopasowania osobnika (f_i) i sumy dopasowań wszystkich osobników ($j = 1..n$).

$$p_i = \frac{f_i}{\sum_{j=1}^n f_j} = \frac{f_i}{n\mu}$$

Ile wyniesie p'_i , jeśli $f'_i = f_i - (\mu - c\sigma)$?

Wyraż p'_i jako funkcję p_i i innych możliwie łatwo interpretowalnych wyrażeń.

Odpowiedź:

$$p'_i = \frac{f_i - \mu}{nc\sigma} + \frac{1}{n} = \frac{\mu}{c\sigma} \left(p_i - \frac{1}{n} \right) + \frac{1}{n}$$

Zinterpretuj powyższy wzór i odnieś do Rys. 2.2. Czym jest $\frac{1}{n}$? Rozważ sytuację, gdzie μ jest małe a σ duże (początek ewolucji) oraz odwrotną, gdy ewolucja jest już zaawansowana.

Przypomnienie: jaki brzmiał jeden z pomysłów pozwalający reprezentować „presję selekcyjną” jako pojedynczą wartość?

Mechanizm skalowania pomaga utrzymać stały napór selekcyjny podczas całej ewolucji, niezależnie od własności optymalizowanej funkcji. Kontrolowanie naporu selekcyjnego jest bardzo ważne – bez tego niemożliwa jest efektywna optymalizacja.

Stosowanie skalowania ma sens, jeśli użyta metoda selekcji wykorzystuje ilorazowość skali wartości przystosowania (a np. selekcja turniejowa i rankingowa nie wykorzystują).

Inna (lepsz niż σ -obcinające?) funkcja skalowania: prawdopodobieństwo wybrania osobnika,

$$p'_i = \frac{1}{1 + \exp\left(\frac{f_i - M}{\sigma}\right)}$$

f_i – wartość funkcji celu osobnika,

M – **mediana** funkcji celu wszystkich osobników w populacji,

σ – odchylenie standardowe funkcji celu w populacji.

Dla maksymalizacji należy odwrócić znak argumentu $\exp()$.

Zalety:

- Minimalny wpływ krańcowo dobrze/źle przystosowanych osobników (w ciągu całego procesu ewolucji), więc znika problem przedwczesnej zbieżności. Dlaczego minimalny? Bo użyto mediany, a nie średniej. Krańcowy osobnik wpłynie tylko na umiarkowane zwiększenie σ .

- Skuteczne rozdzielenie osobników przeciętnie przystosowanych na dwie grupy (powyżej i poniżej mediany).
- Na początku ewolucji zwykle występuje duże σ , czyli małe zróżnicowanie p'_i osobników. Pod koniec ewolucji – zbieżność więc małe σ , zatem osobniki lepsze dużo silniej promowane.

Aby zobaczyć przykładowy rozkład wartości funkcji oceny w populacji podczas ewolucji, zobacz [KU17, górny wykres na rys. 9]. Jak wyjaśnisz utrzymujące się przez długi czas poziome linie (grupy osobników o podobnych, szczególnych wartościach funkcji celu)?

2.2.4 Krzyżowanie

Dyskusja: czy krzyżowanie jest niezbędne w algorytmie ewolucyjnym?

Metody klasyczne:

- Krzyżowanie proste – jednopunktowe.

a b c d e f g h	a b c d e f G H
A B C D E F G H	A B C D E F g h

- Uogólniony, wielopunktowy operator krzyżowania (parametrem jest liczba punktów cięcia):
 - Dla parzystej liczby cięć łańcuchy bitów traktuje się jako zamknięte pierścienie.
 - Dla nieparzystej liczby (jak np. 1 w krzyżowaniu prostym) uwzględnia się początek lub koniec łańcucha bitów jako jeden z krańców zamienianego odcinka.

Doświadczenia dowiodły, że zwiększanie liczby punktów cięcia dla pewnych reprezentacji rozwiązań pogarsza osiągi algorytmu genetycznego. Przyczyną jest większy wpływ niszczący takiego krzyżowania: im więcej punktów cięcia, tym intensywniej rozrywane są wartościowe schematy (zob. twierdzenie o schematach, 2.2.7).

- Krzyżowanie odcinkowe, w którym ustalony współczynnik zmiany odcinka s określa prawdopodobieństwo wystąpienia punktu cięcia w danym punkcie genotypu. Dla genotypu o długości l oczekujemy $l \cdot s$ punktów cięcia, choć liczba ta waha się ze względu na czynnik losowy (prawdopodobieństwo s).
- Krzyżowanie jednorodne. Dla każdego bitu powstającego potomka pierwszego decyduje się (z prawdopodobieństwem p), od którego z dwóch rodziców otrzyma on dany bit. Drugi potomek otrzymuje bit od pozostałego rodzica. Przy $p = \frac{1}{2}$ potomkowie

mają równe szanse na odziedziczenie poszczególnych bitów od jednego lub drugiego rodzica. Gdy p maleje, potomkowie upodabniają się do rodziców. Krzyżowanie jednorodne jest jeszcze bardziej szczegółowe od wielopunktowego – wymianie podlegają bity, nie odcinki. Dla zadań, w których wzajemna lokalizacja bitów nie gra roli, taki rodzaj krzyżowania może być korzystny.

Metody bardziej zaawansowane:

- *Tasowanie*: dodatkowy mechanizm [ECS89] stosowany przy niektórych rodzajach krzyżowania. Polega on na losowej zamianie miejsc bitów w łańcuchach rodziców (u obu tak samo), dokonaniu krzyżowania i przywróceniu pierwotnego porządku bitów u potomków (zastanów się, po co stosuje się taką operację i dla jakich operatorów krzyżowania ma sens?)
- Krzyżowanie adaptacyjne: w genotypie oprócz bitów rozwiązania mogą być przechowywane informacje o miejscach, w których nastąpiło krzyżowanie. Rozwiązania słabe zanikają (wraz z informacją o miejscach krzyżowania), podczas gdy rozwiązania dobre (i informacja o korzystnych punktach krzyżowania) utrzymują się i ulepszają. Tak więc miejsca cięcia podlegają ewolucji wraz z populacją rozwiązań. Rozszerzeniem tego pomysłu jest [rejestrwanie dopasowania](#) (oceny) różnych operatorów genetycznych i wybór operatora z uwzględnieniem jego dopasowania.
- Krzyżowanie z wieloma przodkami: rekombinacji podlegają geny losowane z puli genów rodziców uzyskanych w wyniku selekcji, włączając w to przypadek tzw. „orgii”, w których rozwiązanie-potomek może mieć więcej niż dwóch rodziców (zastanów się, jaki wpływ ma liczba rodziców?)

Możliwe jest również wykonywanie operacji krzyżowania i mutacji jednocześnie – utworzenie operatora rekombinacji wykonującego te dwie operacje w jednym przebiegu.

Podczas ewolucji chcemy uniknąć nadmiernego podobieństwa osobników (i utrzymać różnorodność populacji), a zarazem wymusić odpowiedni napór selekcyjny. Ponieważ operacja krzyżowania działa w kierunku ujednolicania populacji, umiejętne jej stosowanie (w obecności mutacji) może służyć utrzymaniu odpowiedniej różnorodności i właściwego naporu selekcyjnego. Miarą różnorodności osobników może być ich entropia; korzystne jest wówczas uzależnienie od niej prawdopodobieństwa krzyżowania i/lub mutacji.

2.2.5 Mutacja

Dyskusja: czy mutacja jest niezbędna w algorytmie ewolucyjnym?

Mutacja zapobiega przedwczesnej zbieżności oraz utracie fragmentów rozwiązania z powodu

zbytniego ujednolicenia populacji. Mutacja prosta polega na zamianie bitu na przeciwny z pewnym prawdopodobieństwem. W początkowej fazie ewolucji korzystne jest zwykle wysokie prawdopodobieństwo mutacji, które zapobiega zbieżności w kierunku lokalnego optimum; pod koniec oczekuje się dokładniejszego dostrojenia do najlepszego rozwiązania. Stąd wniosek, iż prawdopodobieństwo mutacji powinno spadać wraz z upływem czasu – np. malejąca funkcja numeru pokolenia (co przypomina nieco efekt „temperatury” w SA). Innym rozwiązaniem jest uzależnienie tempa mutacji od różnorodności populacji (podobnie, jak przy krzyżowaniu). Wówczas zbytne ujednolicenie populacji będzie sygnałem do zwiększenia prawdopodobieństwa mutacji.

Z reguły korzystne jest to, że po mutacji osobnik jest podobny w pewnym stopniu do rodzica, oraz że każda mutacja powoduje zmiany zbliżonej wielkości. Czy taka sytuacja ma miejsce, kiedy zastosujemy standardowe kodowanie binarne liczb? Nie. Mutacje będą się zdarzać raz na bitach mniej, raz bardziej znaczących, powodując czasem niewielkie zmiany, czasem ogromne.

Jeśli zastosujemy [kod Gray’a](#), w którym sąsiednie liczby różnią się tylko jednym bitem (odległość Hamminga), to zawsze sąsiednie liczby są osiągalne przez pojedynczą mutację. Ale czy to coś pomoże?

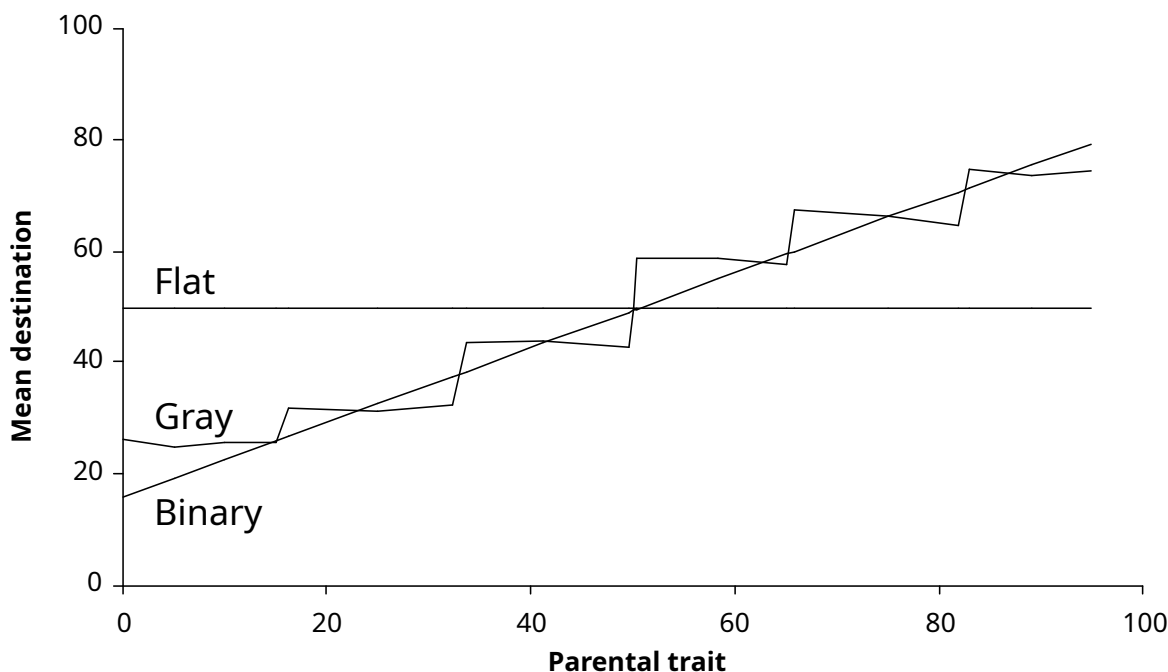
Przedstawienie graficzne: bity = osie X, Y, Z.

	bit 0	bit 1	bit 2
0	0	0	0
1	1	0	0
2	1	1	0
3	0	1	0
4	0	1	1
5	1	1	1
6	1	0	1
7	0	0	1

	bit 0	bit 1	bit 2
0	0	0	0
1	1	0	0
2	1	1	0
3	0	1	0
4	0	1	1
5 (*)	0	0	1
6	1	0	1
7 (*)	1	1	1

cykl Hamiltona 3D (graniczne różnią się też o 1)

ścieżka Hamiltona 3D



Rysunek 2.3: Silne nieciągłości operatora mutacji mają stały wpływ (*bias*) na ewolucję [Bul99, str. 69]. Flat: mutacja polega na wylosowaniu wartości genu z dozwolonego przedziału. Kod Gray’a nie pozwala na uniknięcie wad związanych z mutowaniem zakodowanych wartości liczbowych.

Uwaga: przy kodowaniu Gray’a każda mutacja zmienia wartość genu, ale także jego parzystość. Jeśli *wartość genu* decyduje wraz z jego **parzystością** o jakiejś właściwości (np. *nasilenie lewo/prawo-ręczności*), to z każdą mutacją będzie się zmieniał i *stopień* „ręczności”, i jej **kierunek** (lewo/prawo).

Dyskusja: jakie kodowanie spełniłoby oczekiwanie, że każda mutacja zmienia zakodowaną wartość o ± 1 ?

2.2.6 Podsumowanie parametryzacji AG

Wprowadzanie różnych udoskonaleń i rozszerzeń AG (przykład: adaptacyjne prawdopodobieństwa mutacji) prowadzi niekiedy do powstania kolejnych parametrów. Z drugiej strony część takich modyfikacji pozwala na automatyczne dostrojenie wartości parametrów. Nowe parametry są często łatwiej interpretowalne i/lub ich wpływ na działanie algorytmu i uzyskiwane wyniki stają się bardziej przewidywalny.

Dyskusja: wyobraź sobie, że masz jakiś konkretny problem optymalizacji. Jakie wartości parametrów ustalisz i jakie mechanizmy wybierzesz? Jakimi kryteriami kierujesz się wybierając akurat te, a nie inne wartości?

Wielkość populacji ma wpływ przede wszystkim na bezwładność algorytmu. Przy większych populacjach algorytm reaguje wolniej, dlatego przy zastosowaniach *on-line* zaleca się względnie małe populacje. Duże populacje, szczególnie na początku ewolucji, potrzebują więcej czasu by dotrzeć do rozwiązań dobrych. Z kolei dla zastosowań *off-line* duże populacje są korzystne, ponieważ dysponują większą ilością informacji i pozwalają na dokładniejsze przeszukiwanie przestrzeni rozwiązań. Duża liczba osobników zmniejsza zarazem ryzyko utknięcia w optimum lokalnym. Trzeba jednak tak dobrać wielkość populacji, żeby zdążyła ona zbiec do regionu dobrych rozwiązań w dostępnym czasie.

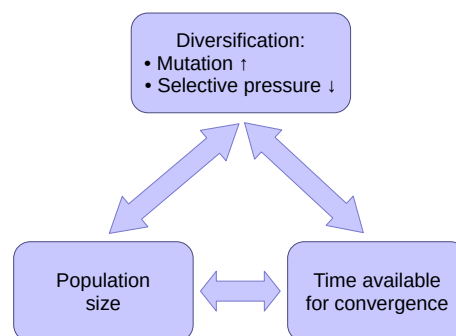
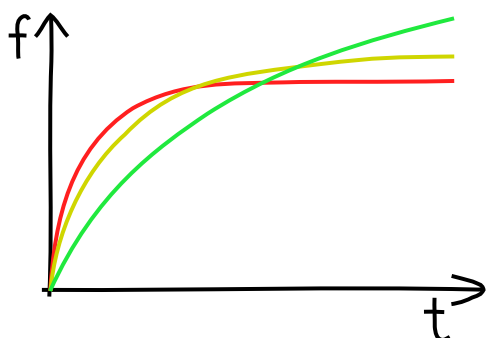
Prawdopodobieństwo krzyżowania wpływa na stopień wymiany informacji w populacji, a także na jej ujednolicanie. Typowa wartość zalecana w prostym AG, by uzyskać kompromis pomiędzy efektywnością *off-line* i *on-line*, to 60%. Przy stosowaniu metod selekcji o niskim rozrzucie losowym (tzn. bardziej deterministycznych, niż np. reguła ruletki) zaleca się wyższe prawdopodobieństwo krzyżowania – nawet 100% [Gol03, str. 130]. Te zalecenia trzeba jednak traktować z dużym dystansem – trudno podać uniwersalną, dobrą wartość nie znając znaczenia rozwiązania w danym problemie optymalizacji, sposobu kodowania binarnego i charakterystyki (skuteczności) operatora krzyżowania.

Prawdopodobieństwo mutacji odpowiada za liczbę losowych zmian bitów osobników. Zaleca się, by było ono odwrotnością liczby zmiennych decyzyjnych (jeśli zmienne decyzyjne są binarne, to mutacji ilu genów spodziewamy się w każdym pokoleniu?). Czasem spotyka się prawdopodobieństwo mutacji będące odwrotnością liczby osobników w populacji (mutacji ilu genów spodziewamy się w każdym pokoleniu?)

Mechanizm selekcji o właściwościach korzystniejszych od mechanizmu ruletki to opisany wcześniej wybór losowy według reszt, bez powtórzeń lub z powtórzeniami, albo metody turniejowe. Polecany mechanizm skalowania to odcięcie na poziomie zależnym od odchylenia standardowego, ewentualnie połączone ze skalowaniem liniowym.

Dobrym kryterium stopu przy zastosowaniach *off-line* jest liczba pokoleń bez poprawy (choć bez wprowadzenia progu minimalnej zmiany algorytm może działać zbyt długo z powodu ciągłych, nieznaczająco małych polepszeń). Można też monitorować średnie podobieństwo w populacji i kończyć optymalizację, gdy zaniknie różnorodność.

Interakcje parametrów:



2.2.7 Twierdzenie o schematach

Twierdzenie o schematach³ (ang. *schema theorem*) opisuje zasadę działania algorytmów ewolucyjnych. Dla AG, „schemat”⁴ to ciąg symboli 0, 1 oraz * (gwiazdka zastępuje 0 lub 1). Dla specyficznego AE, jakim jest AG z mutacją prostą i krzyżowaniem prostym, określamy *rzęd schematu* jako liczbę ustalonych pozycji w schemacie, oraz *rozpiętość schematu* jako odległość między jego skrajnymi ustalonymi pozycjami.

Przystosowaniem schematu jest średnia z przystosowania wszystkich pasujących do niego genotypów. Dość proste obliczenia prowadzą do wniosku–twierdzenia, że *liczba genotypów pasujących do schematów posiadających wyższe od średniego przystosowanie, niski rząd i małą rozpiętość, będzie rosła wykładniczo w kolejnych pokoleniach*.

Hipoteza cegiełek mówi z kolei, że za efektywnym działaniem AE stoi zdolność do korzystnego łączenia krótkich ciągów genów (efekt synergii), zobacz też Epistaza (str. 25), Tasowanie (str. 19), Inwersja (str. 62), Nieporządną AG (rozdział 2.2.9).

2.2.8 Problemy zawodne (trudne dla AG) i twierdzenie „No Free Lunch”

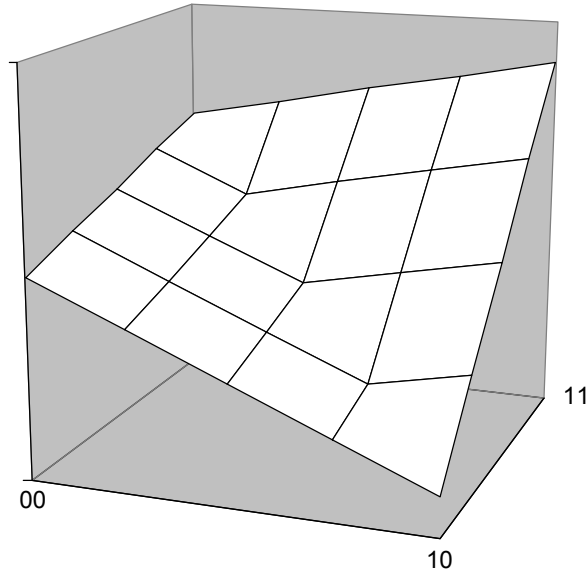
NFL: dla wszystkich problemów optymalizacji jakie mogą teoretycznie zaistnieć, skuteczność wszystkich algorytmów jest średnio taka sama (analogiczne twierdzenia obowiązują w uczeniu maszynowym, kompresji danych, itd.). Rozważ konsekwencje tego twierdzenia i porównaj: hiperheurystyki z rozdziału 2.6.2.

Aby skonstruować taki problem, który sprawi trudność algorytmowi genetycznemu, postarajmy się zanegować hipotezę cegiełek. Chcemy, by dobre „cegiełki” po połączeniu tworzyły niekorzystną strukturę. Najprostszy taki przypadek można uzyskać dla genotypów o długości 2 (jest to problem zwodniczy rzędu drugiego).

Założmy, że istnieją cztery schematy:

³https://en.wikipedia.org/wiki/Holland%27s_schema_theorem

⁴[https://en.wikipedia.org/wiki/Schema_\(genetic_algorithms\)](https://en.wikipedia.org/wiki/Schema_(genetic_algorithms))



Rysunek 2.4: Przykładowy problem zwodniczy rzędu drugiego, typu I.

```

*** 0 *** 0 ***
*** 0 *** 1 ***
*** 1 *** 0 ***
*** 1 *** 1 ***

```

Gwiazdki odpowiadają dowolnej liczbie symboli nieustalonych (ale pozycje ustalone są we wszystkich schematach na tych samych miejscach). Oznaczmy średnie oceny schematów przez f_{00} , f_{01} , f_{10} , f_{11} i schemat z dwoma jedynekami jest globalnym optimum (f_{11}). Aby problem był zwodniczy, chcemy, by schematy z jednym ustalonym zerem były lepsze od odpowiednich schematów z jedyneką, to znaczy, by spełniona była przynajmniej jedna z nierówności:

$$f_{0*} > f_{1*} \quad f_{0*} = (f_{00} + f_{01})/2, \text{ itd.}$$

$$f_{*0} > f_{*1}$$

Taki problem nazywany jest minimalnym problemem zwodniczym (ang. *minimal deceptive problem*, *MDP*), ponieważ nie istnieje problem zwodniczy rzędu pierwszego. Są dwa typy problemów zwodniczych rzędu drugiego (por. Rys. 2.4):

Typ I: $f_{01} > f_{00}$

Typ II: $f_{00} \geq f_{01}$

W *MDP* funkcja przystosowania *nie może* być przedstawiona jako liniowa kombinacja poszczególnych alleli, czyli w postaci $f(x_1, x_2) = b + \sum_{i=1}^2 a_i x_i$.

To, że problem jest zwodniczy nie oznacza jeszcze, iż algorytm genetyczny nie znajdzie optimum. Oznacza to jednak, że funkcja przystosowania (przedstawiona na osi pionowej) nie może być wyrażona jako liniowa kombinacja poszczególnych bitów, zatem występuje zjawisko epistazy – nieliniowości. Zwykle taki problem nie okazuje się jednak AG-trudny, czyli algorytm genetyczny może znaleźć rozwiązanie optymalne. Zachowanie algorytmu zależy jednak od wielu czynników, takich jak na przykład początkowa obecność schematów w populacji osobników. W szczególności, jeśli wszystkie cztery schematy występują w populacji początkowej, problem zwodniczy rzędu 2 typu I nie jest AG-trudny. Dla zadań typu II zachowanie algorytmu zależy od proporcji występowania schematów w populacji: jeśli schemat 00 będzie występował w przewadze, algorytm może być zbieżny do rozwiązania nieoptymalnego (choć taka sytuacja występuje rzadko). Bardziej szczegółowa analiza problemów zwodniczych znajduje się w [Gol03, str. 63–70, 375–382].

Powyższe rozważania dotyczyły klasycznego algorytmu genetycznego, z krzyżowaniem prostym i mutacją prostą. Przedstawiony problem zwodniczy jest reprezentantem zadań, które mogą sprawiać trudność algorytmom genetycznym z powodu występowania izolowanych optimów. Oczywiście takie problemy stanowią również trudność dla innych algorytmów optymalizacji.

Zaproponowano wiele udoskonaleń algorytmów genetycznych, które mają zapobiegać opisanej sytuacji. Można jej uniknąć stosując specyficzne kodowanie (potrzebna jest wtedy wiedza o optymalizowanej funkcji), można zastosować operator inwersji (str. 62), można również stosować nieporządne (rozdział 2.2.9) lub hierarchiczne (rozdział 2.2.10) algorytmy genetyczne.

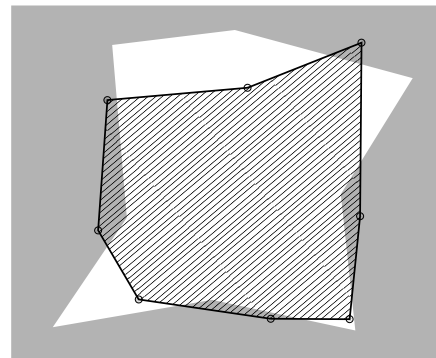
Epistaza (ang. *epistasis*): cecha reprezentacji (i ew. operatorów dla niej) – stopień zależności pomiędzy różnymi genami w chromosomie [Dav90]. Jeśli dana reprezentacja posiada wysoką epistazę, fenotypowy efekt pewnych genów zależy od alleli (wartości) innych genów (poligeniczność).

Zerowa epistaza: każdy gen niezależnie wpływa na wartość funkcji celu (i wtedy nie warto stosować AE). Dla skuteczności AE **tym lepiej**, im mniejsza epistaza. Dla niektórych definicji funkcji celu, projektując reprezentację i operatory może opłacać się zaakceptować niewielki wzrost epistazy, jeśli zyskamy korzystniejszy związek między topologią przestrzeni przeszukiwania, a krajobrazem przystosowania (por. Rys. 2.8).

Oszacuj mniej więcej epistazę dla następujących funkcji celu $f(x_1, x_2)$, gdzie x to liczby – wartości genów: $x_1 + x_2$, $x_1 - x_2$, $x_1 \cdot x_2$, $x_1 + x_2 \cdot x_2$, $x_1 + x_1 \cdot x_2$, $\frac{x_1}{x_2}$, $x_2 + \frac{x_1}{x_2}$.

Przykład.

Optymalizujemy kształt figury 2D (albo przekrój obiektu 3D); geny opisują rozmieszczenie wierzchołków w przestrzeni (współrzędne x_i, y_i).



Porównaj epistazę dla następujących scenariuszy:

1. Same geny x_i, y_i oraz operator mutacji przesuwający losowy wierzchołek.
2. Wprowadzamy dwa dodatkowe geny: obrotu i skali. Mutacja tych pojedynczych genów zmienia orientację (obrót) i rozmiar całej figury.
3. Zamiast tych dwóch dodatkowych genów wprowadzamy specjalne operatory mutacji-obrotu i mutacji-skalowania.

W scenariuszu (1) zmiana orientacji i rozmiaru figury jest możliwa jedynie przez bardzo wiele niezależnych mutacji współrzędnych wierzchołków. Tymczasem w pewnych zadaniach (np. w optymalizacji online, gdy docelowy kształt otworu się zmienia), możliwość szybkiego obrotu i skalowania może przyspieszyć zbieżność i polepszyć jakość uzyskiwanych rozwiązań. W scenariuszu (2) rozważ wpływ konkretnej implementacji (matematyczne działania przekształcające genotyp $\rightarrow$ fenotyp) – chodzi o efekty złożenia mutacji (1) i (2).

Epistazę, jako stopień interakcji pomiędzy genami, można zamodelować tak [RW95; NK00]:

$$\begin{aligned} f(s) = & \text{stała} \\ & + \sum_{i=0}^{l-1} \text{efekt } s_i \\ & + \sum_{i=0}^{l-2} \sum_{j=i+1}^{l-1} \text{interakcja między } s_i \text{ i } s_j \\ & + \dots \\ & + \text{interakcja między } s_0, s_1, \dots, s_{l-1} \\ & + \text{losowy błąd.} \end{aligned}$$

s_i – gen i -ty,

l – liczba genów (numerowane od zera).

Interakcje mogą być dodatnie i ujemne: jeśli np. interakcja pomiędzy s_i i s_j ma taki sam znak, co s_i i s_j , to taka interakcja wzmacnia działanie tych dwóch genów i może być pożądana.

—

Problemy optymalizacyjne sprawiają trudność algorytmom ewolucyjnym (i innym algorytmom optymalizacji) z trzech powodów:

- *isolation* (igła w stogu siana)
- *deception* (nieskuteczność współpracy schematów/cegiełek budujących)
- *multimodality* (liczne optima lokalne)

Dwie ostatnie cechy nie są ani konieczne, ani wystarczające, by problem był GA-trudny.

Istnieją różne miary **przewidywania trudności** problemu dla algorytmu ewolucyjnego; znane są w szczególności proste miary *zmienności epistazy* oraz *korelacji odległości od optimum z wartością dopasowania* (por. rozdział 2.5.2). Praca [NK00] zawiera ich definicje, charakterystyki oraz dyskusję skuteczności.

Aby oszacować trudność problemu próbkuje się przestrzeń rozwiązań, a potem oblicza się funkcję oceny dla wybranych rozwiązań. Następnie wylicza się różne wskaźniki, biorące pod

uwagę odległość pomiędzy rozwiązaniami (według przyjętej reprezentacji genetycznej) i między ich ocenami. Obecnie miary te nie są jednak zadowalające: da się skonstruować takie problemy, dla których owe wskaźniki zawodzą (problem okazuje się dla algorytmu optymalizacji łatwy, jednak według miar jest trudny, lub na odwrót). Bardziej wiarygodne są miary biorące pod uwagę zachowanie AE podczas rozwiązywania danego problemu (tzw. miary *on-line*), jednak ich podstawową wadą jest złożoność – czas obliczeń potrzebny do oceny problemu.

2.2.9 Nieporządnym algorytm genetyczny

Ang. *messy genetic algorithm*.

Podejście „nieporządne” (niechlujne) ma na celu, podobnie jak opisany na str. 62 operator inwersji, polepszenie własności algorytmu genetycznego poprzez skuteczniejsze konstruowanie, wykorzystanie i przekształcanie schematów. Nieporządnym 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 operacji 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.

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.

2.2.10 Hierarchiczny algorytm genetyczny

Dyskusja: czy można w jakiś sposób „wykryć” epistazę?

Podobnie jak w przypadku nieporządných algorytmów genetycznych, motywacją do rozwoju H-GA była chęć automatycznego odkrywania stopnia współzależności (ang. *linkage*, nie mylić z *genetic linkage*) 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.

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

Wykrywanie zależności: techniki statystyczne i empiryczne

Metody wykrywania zależności między genami dzieli się czasem na statystyczne (np. H-GA albo rodzina metod GOMEA) i empiryczne (np. DLED). Te pierwsze bazują na osobnikach w populacji oraz ich ocenach i na tej podstawie statystycznie szacują niezależność genów. Te drugie próbują i oceniają pełne otoczenie konkretnego osobnika, zatem w jego lokalnym sąsiedztwie i dla jego konkretnego zestawu wartości genów uzyskują kompletną informację o (nie)zależności.

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

Założmy, 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 $1 \rightarrow 0$ i $0 \rightarrow 1$.

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 niezmienniona oraz jeśli będzie zmieniona.
3. Decyzja o zależności jest binarna i wynika ze spełnienia warunku/warunków poniżej.

Warunki z artykułu [PTK23]:

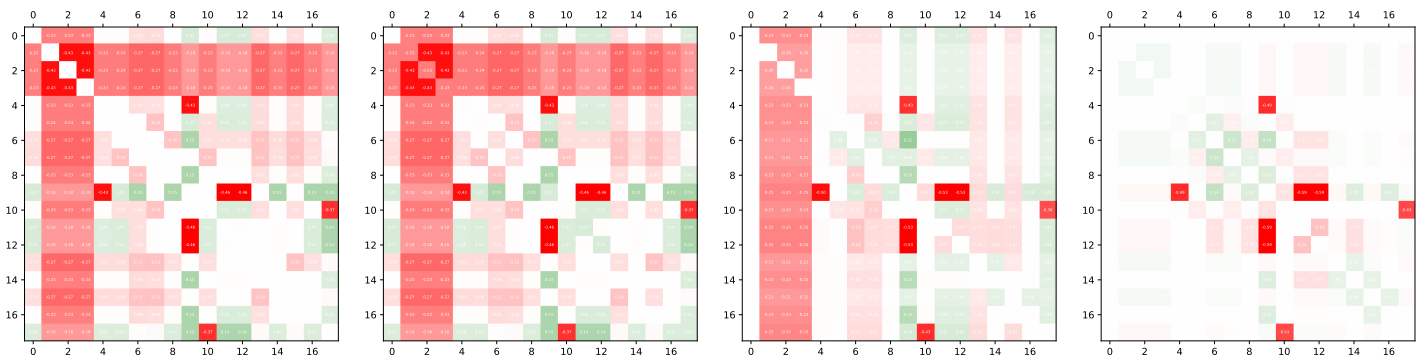
The situation changed when the Direct Empirical Linkage Discovery (DLED) was proposed [17]. To check the dependency between genes g , h , DLED requires computing $f(\mathbf{x})$, $f(\mathbf{x}, g)$, $f(\mathbf{x}, h)$, $f(\mathbf{x}, g, h)$ values, where $(\mathbf{x}, g)$ and $(\mathbf{x}, h)$ are the genotypes of individual $\mathbf{x}$ with genes g and h flipped, respectively. Finally, $f(\mathbf{x}, g, h)$ is the genotype of $\mathbf{x}$ 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.

Epistaza – przykład ciągły

- 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 $\rightarrow$ zmniejszenie złożoności obliczeniowej
 - zaprojektować krzyżowanie i mutację tak, aby zachowywały korzystne epistatyczne interakcje genów $\rightarrow$ traktować współpracujące epistatycznie grupy genów



Rysunek 2.5: (1) Zmiana w wartości fitness po wyłączeniu par genów w przykładowym osobniku. (2) Zmiana w wartości fitness po wyłączeniu par genów; na przekątnej efekt wyłączenia jednego genu. (3) Odjęcie wartości na przekątnej od wierszy: jak interpretować efekt? (4) Addytywność wpływu lub jej brak.

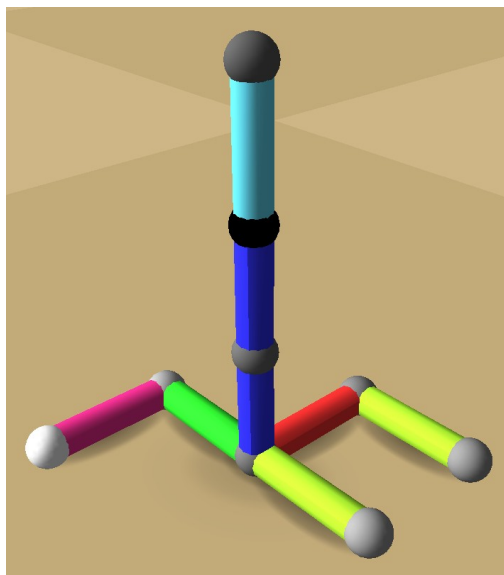
jako integralne zespoły → ochrona i skuteczna propagacja „cegiełek” budujących dobre rozwiązania [GT12; TB13]

- 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 [Thi18].

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

Bohater powyższej analizy: genotyp, fenotyp i fitness.

Genotyp: */*9*/UDDDLFBFBRFBBFBFB*



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

2.2.11 Empiryczna i teoretyczna ocena AG

Badanie efektywności i zachowania AE można prowadzić teoretycznie albo eksperymentalnie (empirycznie). Analiza teoretyczna daje ugruntowaną, pewną wiedzę, ale często da się tak badać tylko bardzo proste modele (tj. z wieloma założeniami). Analiza empiryczna prowadzi do mniej pewnej wiedzy, trudniej o uogólnienia, ale zawsze da się taką analizę przeprowadzić. Oprócz testowania skuteczności algorytmu na docelowym problemie optymalizacji, często używa się znanych problemów testowych⁵ albo sparametryzowanych modeli problemów pozwalających na sterowanie nasileniem epistazy, takich jak model NK⁶ oraz jego warianty z neutralnością: NKP (probabilistyczny) i NKQ (z kwantyzacją).

2.2.12 Neutralność

Konsekwencje neutralności w krajobrazach przystosowania [Gea+02]:

- potocznie/tradycyjnie uważa się, że krajobraz przystosowania „składa się z pagórków”, a mutacje powodują zmianę przystosowania
- prowadzi to do koncepcji „optimów lokalnych”, w których rozwiązania lub ich populacje mogą utknać

⁵https://en.wikipedia.org/wiki/Test_functions_for_optimization

⁶https://en.wikipedia.org/wiki/NK_model

- badania molekularne nad krajobrazami różnych sposobów zwijania RNA sugerują, że duża część mutacji na poziomie molekularnym jest selekcyjnie neutralna
- neutralność jest również obecna w wielu rzeczywistych problemach optymalizacyjnych (popularne „płaskowyzę” lub sąsiedzi tej samej jakości)
- zatem: wiele genotypów $[\rightarrow \text{jeden fenotyp}] \rightarrow \text{jedna wartość funkcji celu}$
- neutralność: jedna z przyczyn równowag przestankowych
- jeśli neutralność występuje często w krajobrazie, ryzyko uwięzienia populacji AE w lokalnych optimach jest niskie
- w związku z tym rośnie rola charakterystyk operatorów rekonfiguracji (mutacji/sąsiedztwa, krzyżowania) i rola dryfu genetycznego
- kluczowa rola ostrej i nieostrej nierówności w implementacjach przeszukiwania lokalnego, o której mówiliśmy wcześniej (ostra nierówność $\rightarrow$ wszystkie neutralne ruchy są potencjalnymi końcami)

2.2.13 Dryf genetyczny

Główne siły prowadzące ewolucję:

- co kieruje ewolucją? co wpływa na trajektorię populacji?
- krajobraz przystosowania kontra operatory rekonfiguracyjne
- co się stanie, gdy wszystkie lub większość mutacji będzie niekorzystna, a dawne osobniki nie będą zachowywane?
- „bias”/„neutralność” krajobrazu przystosowania vs. bias/neutralność operatorów rekonfiguracyjnych
- operatory rekonfiguracyjne: zmiany w częstości alleli, ale zwykle chcemy uniknąć „biasu”!
- widzisz populację osób o niebieskich, zielonych i brązowych oczach; po ilu pokoleniach wszyscy mają zielone oczy. Dlaczego?

Dryf genetyczny:

- wyłączmy presję selekcyjną, mutację i krzyżowanie
- [przykład kamyków w słoiku](#)

- dyskretny charakter populacji (składających się z odrębnych osobników), więc idealnie równy rozkład alleli nie jest możliwy (np. 1/64 wśród 5000 osobników)
- **wąskie gardła** (populacja tymczasowo kurczy się do bardzo małych rozmiarów)
 - efekt założyciela
- wpływ rozkładu cech i wielkości populacji
 - prawdopodobieństwo, że dana cecha ostatecznie zdominuje całą populację, to jej częstość w populacji
 - oczekiwana liczba pokoleń do wystąpienia całkowitej dominacji jest **proporcjonalna do wielkości populacji**
- wpływ losowości (nieograniczonej) vs. „sprawiedliwość” – przypomnienie: nasza wcześniejsza szczegółowa dyskusja

-
- jeśli nie znasz tych zjawisk, ich konsekwencje mogą być sprzeczne z intuicją, błędnie interpretowane i mylnie przypisywane innym mechanizmom!
 - interakcja krajobrazu przystosowania (selekcja), operatorów rekonfiguracji i dryfu genetycznego.

2.2.14 Przykład analizy teoretycznej: brak mutacji

Oto przykład prostej analizy teoretycznej algorytmu genetycznego zero-jedynkowego, z selekcją ruletkową i krzyżowaniem, bez mutacji. Rozważamy populację m osobników o długości k . Ile jest różnych stanów populacji? Ile jest stanów, które są atraktorami?

Użyjemy łańcuchów Markowa; niech π – stan, P – macierz prawdopodobieństw przejścia⁷. Jest 2^{mk} różnych stanów. Rozkład prawdopodobieństwa stanów, w których możemy wyładować po n krokach⁸ od stanu π to π -ty wiersz macierzy P^n – zapiszmy go jako πP^n . Ponieważ nie ma mutacji, pewne stany są absorbujące (nie ma z nich wyjścia) – wszystkie osobniki są równe; takich stanów jest $a = 2^k$. Zatem macierz P możemy wyrazić jako⁹

$$P = \begin{bmatrix} I_a & 0 \\ R & Q \end{bmatrix}$$

⁷https://en.wikipedia.org/wiki/Stochastic_matrix

⁸<https://www.youtube.com/watch?v=nnssRe5DewE>

⁹https://en.wikipedia.org/wiki/Absorbing_Markov_chain

gdzie I_a to macierz $a \times a$ wypełniona zerami oprócz jedynek po przekątnej, R to podmacierz $t \times a$ opisująca przejścia do stanu absorbującego, Q to podmacierz $t \times t$ opisująca przejścia do stanów nie-absorbujących; $t = 2^{mk} - a$. Dla n kroków będziemy mieli

$$P^n = \begin{bmatrix} I_a & 0 \\ N_n R & Q^n \end{bmatrix}$$

gdzie $N_n = I_t + Q + Q^2 + Q^3 + \dots + Q^{n-1}$, I_t to macierz $t \times t$ wypełniona zerami oprócz jedynek po przekątnej. Granicznie

$$\lim_{n \rightarrow \infty} P^n = \begin{bmatrix} I_a & 0 \\ (I_t - Q)^{-1} R & 0 \end{bmatrix}.$$

A zatem, co można łatwo zobaczyć mnożąc przykładowe macierze w numpy czy nawet w arkuszu kalkulacyjnym, nasz algorytm zaczynając ze stanu nie-absorbującego (prawdopodobieństwa $(I_t - Q)^{-1} R$) wyląduje na pewno w jakimś stanie absorbującym i w nim pozostanie (I_a). Obliczmy prawdopodobieństwo trafienia do stanu absorbującego [Fog00, str. 105]. Niech $\Gamma = \{0, 1\}$. Po n iteracjach, nasz algorytm znajdzie się w stanie γ , $\gamma \in (\Gamma^k)^m$:

$$\Pr(\gamma \in A) = \sum_{i=1}^a (\pi^* P^n)_i = \sum_{i=1}^a \left(\pi^* \begin{bmatrix} I_a \\ N_n R \end{bmatrix} \right)_i$$

gdzie $(\cdot)_i$ oznacza i -ty element poziomego wektora, A to zbiór wszystkich stanów absorbujących, a π^* to wektor poziomy opisujący prawdopodobieństwa rozpoczęcia algorytmu w każdym stanie populacji. Graniczne prawdopodobieństwo absorpcji

$$\lim_{n \rightarrow \infty} \sum_{i=1}^a \left(\pi^* \begin{bmatrix} I_a \\ N_n R \end{bmatrix} \right)_i = \sum_{i=1}^a \left(\pi^* \begin{bmatrix} I_a \\ (I - Q)^{-1} R \end{bmatrix} \right)_i = 1.$$

Zadanie dla chętnych. Stosujemy pokoleniowy algorytm ewolucyjny, populacja n osobników, g z n osobników jest dobrych, $n - g$ jest złych. Wykorzystujemy selekcję turniejową (wielkość turnieju k) ze zwracaniem, w której jeśli osobnik dobry spotka się z złym, wygrywa dobry.

Założ, że krzyżowanie i mutacja nie zmieniają charakterystyki osobnika (dobry zostaje dobrym, zły pozostaje złym).

1. Zakładając, że w populacji jest połowa osobników dobrych ($g = \frac{n}{2}$), jak zmieni się ta proporcja w kolejnym pokoleniu (tzn. po jednej selekcji)?
2. Zakładając, że wszystkie osobniki podlegają mutacji i mutacja nigdy nie polepsza złego osobnika, ale pogarsza osobnika dobrego z prawdopodobieństwem m , jakie musi być co najmniej g , żeby liczba dobrych w populacji nie spadała?

2.3 Strategie ewolucyjne (*evolutionary strategies*)

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* (patrz rozdział ??). 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 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 odchyłeń 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 odchyłeń 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*.

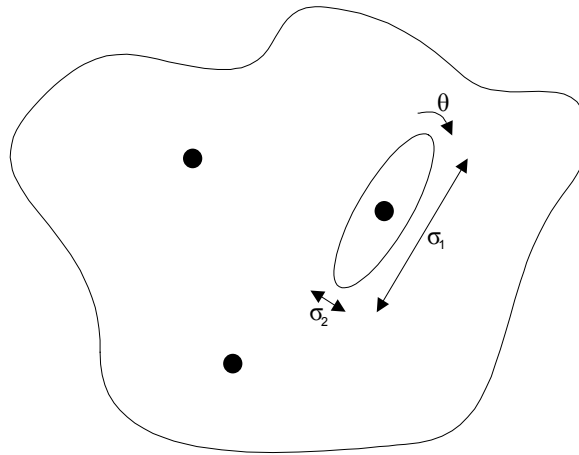
Ulepszeniem strategii dwuelementowej jest strategia wieloelementowa, w której, podobnie jak w AG, istnieje populacja osobników. Wprowadza się dodatkowo operację krzyżowania jednorodnego, jednak nie stosuje się jej do wszystkich osobników, tylko do dwóch z nich – tak, że powstaje jeden potomek, który 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 zwężły 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.

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 odchyłeń 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 2.6: Parametry mutacji w strategiach ewolucyjnych.

W szczególności znana z wydajności jest strategia ewolucyjna dostosowująca macierz kowariancji mutacji, CMA-ES,¹⁰ której implementacja jest dostępna np. w bibliotece DEAP.¹¹ 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.

Główna idea CMA-ES:

¹⁰<https://en.wikipedia.org/wiki/CMA-ES>

¹¹<https://deap.readthedocs.io/en/master/examples/cmaes.html>

- 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.

2.4 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 krzyżowanie simpleksowe z rozdziału 2.5.1.

2.5 Programowanie ewolucyjne (*evolutionary programming*)

Zbigniew Michalewicz wprowadził na takie podejście nazwę „program ewolucyjny” [Mic96].

EP, originally conceived¹² by Lawrence J. Fogel¹³ in 1960, is a stochastic OPTIMIZATION strategy similar to GAs, but instead places emphasis on the behavioral linkage between PARENTs and their OFFSPRING, rather than seeking to emulate specific GENETIC OPERATORS as observed in nature.

Three ways in which EP differs from GAs:

1. There is no constraint on the representation. The typical GA approach involves encoding the problem solutions as a string of representative tokens, the GENOME. In EP, the representation follows from the problem. Example: a neural network can be represented in the same manner as it is implemented, because the mutation operation does not demand a linear encoding. (In this case, for a fixed topology, real-valued weights could be coded directly as their real values and mutation operates by perturbing a weight vector with a zero mean multivariate Gaussian perturbation. For variable topologies, the architecture is also perturbed).
2. The mutation operation simply changes aspects of the solution according to a statistical distribution: minor variations in the behavior of the offspring are highly probable, substantial variations are unlikely. The severity of mutations is often reduced in time.
3. EP typically does not use any CROSSOVER as a GENETIC OPERATOR.

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, permutacja¹⁴, drzewo, graf nieskierowany, graf skierowany, macierz, wyrażenia logiczne, reguły (jak w GBML, rozdział 2.8), sieci neuronowe, automaty, wyrażenia opisane gramatyką (np. zapisane w ONP), wyrażenia o strukturze drzewa, programy (jak w GP, rozdział 2.6), ...

Przykładem takiej specyficznej reprezentacji jest ta stosowana w „gładkim algorytmie ewo-

¹²<https://www.kanadas.com/whats-ep.html>, email from https://en.wikipedia.org/wiki/David_B._Fogel

¹³https://en.wikipedia.org/wiki/Lawrence_J._Fogel

¹⁴Operatory krzyżowania dla permutacji: OX, PMX, ERO, inne: <https://hrcak.srce.hr/file/163313>

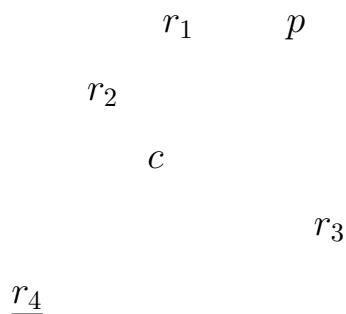
lucyjnym” [Gut94; Gut97]. Osobnik modeluje tu krzywe ciągłe i gładkie. Zastosowania: wygładzanie krzywych/serii danych o nieznanym modelu analitycznym, usuwanie zbędnego tła w sygnale fizycznym itp. Genotyp opisuje (dyskretną) formę krzywej za pomocą skończonej liczby par (x_i, y_i) , $i = 1..n$, n – liczba przedziałów. x_i można traktować jako i , choć przedziały mogą być również niejednakowe. Stosowane są również specyficzne operatory genetyczne, w szczególności gładkie krzyżowanie i mutacja.

2.5.1 Reprezentacja liczb rzeczywistych

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: $d_1 = r_1 \cdot a + r_2 \cdot (1 - a)$, $d_2 = \dots$. Wagę a można losować przy każdym wykonaniu.

Inny operator krzyżowania – 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 .



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.)

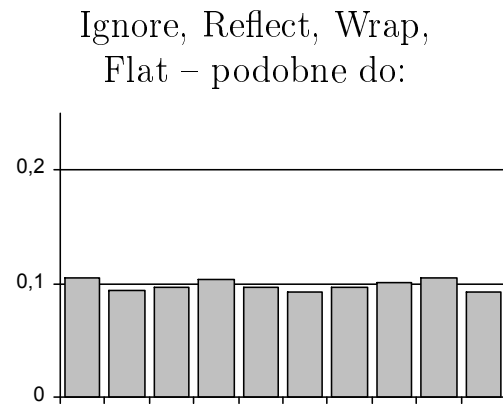
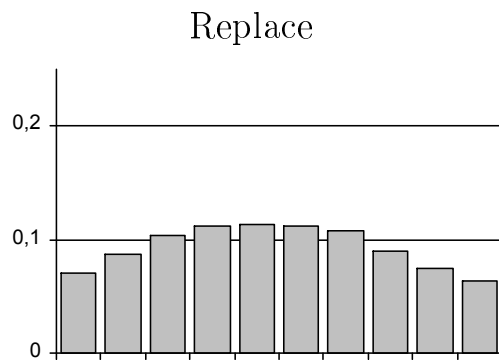
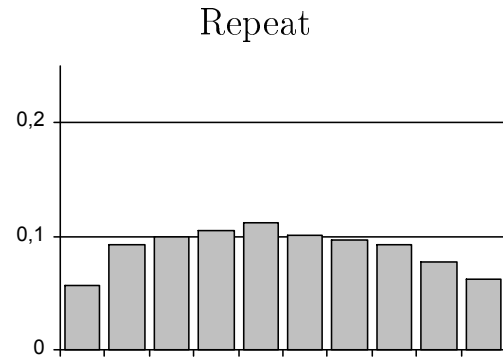
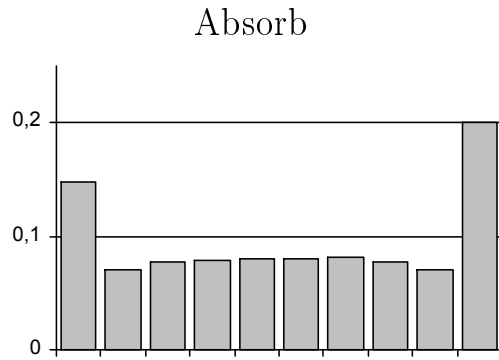
Aby uniezależnić się od „osi” (tzn. aby mutacja nie przebiegała tylko równolegle do osi, czyli nie dotyczyła pojedynczych parametrów, co byłoby niekorzystne gdyby funkcja celu była np. obróconą wersją funkcji zależnej wprost od parametrów), mutuje się naraz wszystkie geny (i wtedy stosujemy rozkład normalny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

- **Absorb**: Illegal mutant values are truncated to the nearest boundary.
- **Repeat**: Mutant values are repeatedly generated, until a legal value is obtained.
- **Replace**: Any offspring for which illegal trait values are generated is replaced by a new offspring, re-choosing parents.
- **Ignore**: Mutation events which transgress legal bounds are ignored. Rather than inherit an illegal mutant value, offspring inherit the parental value.
- **Reflect**: Mutant values lying a distance of d above (or below) the legal range are replaced by values a distance of d below (or above) the nearest boundary.
- **Wrap**: The trait is treated as if it were periodic. The edges of its legal range “wrap” around. Mutant values are calculated modulo the trait’s range.

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:

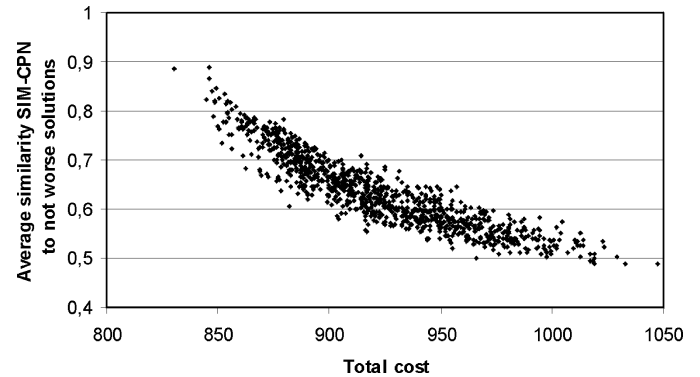
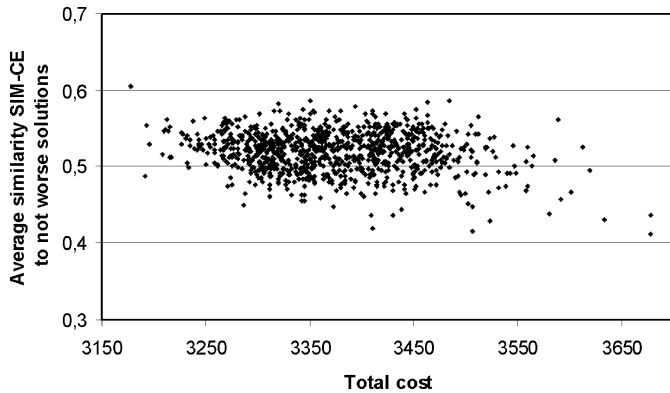


To, że w metodach **Ignore**, **Reflect**, **Wrap**, uzyskano taki sam rozkład jak we **Flat** nie oznacza, że metody te zachowują się tak samo. Faktycznie zasada ich działania jest zupełnie inna. Przykład: metoda **Ignore**. W okolicy granic przedziału więcej mutacji będzie nielegalnych i ignorowanych. 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 **Ignore**, **Reflect**, **Wrap** 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 **Absorb** 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.

Przegląd operatorów mutacji i krzyżowania dla problemów numerycznych zawierają książki-zbiory [Gwi07a; Gwi07b].



Rysunek 2.7: Podobieństwo rozwiązań a podobieństwo ich ocen – przykład z [Kub04].

2.5.2 Krzyżowanie i mutacja a globalna wypukłość

Globalna wypukłość: dobre rozwiązania są do siebie bardziej podobne, niż do gorszych. Da się obiektywnie i ilościowo zmierzyć! (*fitness-distance correlation*, FDC). Jeśli wysoka, to warto zaczynać optymalizację z dobrych rozwiązań, a dywersyfikacja może polegać na niewielkich zaburzeniach rozwiązania (zamiast zaczynać algorytm zupełnie od nowa).

Krzyżowanie zachowujące odległość (*distance-preserving crossover*, DPX):

$\text{dist}(\text{parent1}, \text{parent2}) = \text{dist}(\text{parent1}, \text{child}) = \text{dist}(\text{parent2}, \text{child})$.

Świadome tworzenie skutecznych operatorów krzyżowania:

- pomyśl, jakie cechy rozwiązania wpływają na wartość funkcji celu,
- stwórz miary odległości bazujące na tych cechach,
- zbadaj, czy te miary determinują globalną wypukłość (Rys. 2.7),
- jeśli tak, wykorzystaj te cechy budując operator DPX.

Analogicznie postępujemy, gdy chcemy stworzyć skuteczny operator mutacji (sąsiedztwa) – opieramy się na takich aspektach podobieństwa rozwiązań, które ujawniają powyższą korelację.

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 ścisłością – rozdział 2.2.2,
- szacowania różnorodności w populacji i oceny zbieżności,

- analizy struktury populacji; analizy skupień w zbiorze rozwiązań,
- utrzymywania „gatunków” podczas ewolucji – rozdział 2.7.3,

oraz wszędzie tam, gdzie występuje potrzeba wyznaczenia różnicy dwóch rozwiązań, np.

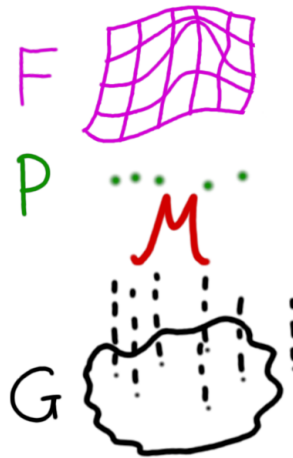
- w ewolucji różnicowej – rozdział 2.4,
- przy krzyżowaniu simpleksowym – rozdział 2.5.1.

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 edycyjnej¹⁵ oraz odległości spychaczowej¹⁶ (tłum. MK).

2.5.3 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?

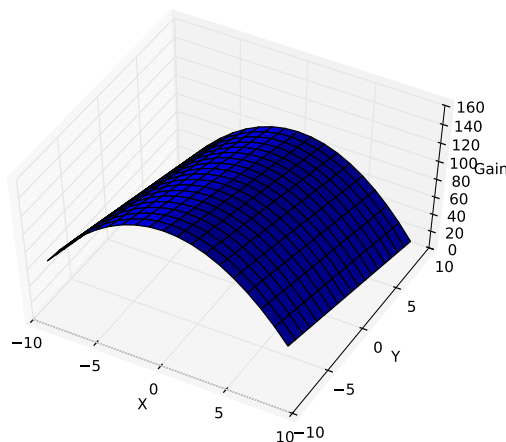
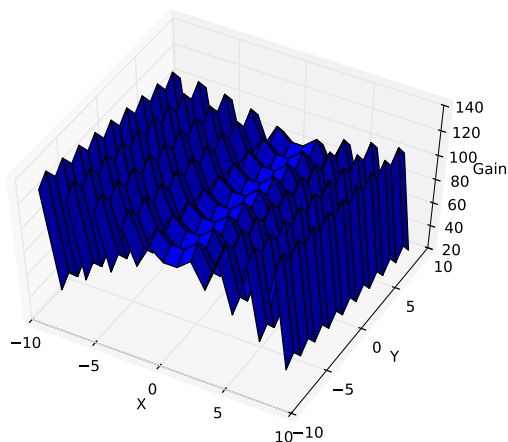


Przypomnij sobie RGB $\leftrightarrow$ HSL, sygnał $\leftrightarrow$ widmo, kran  $\leftrightarrow$ , ...

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?

¹⁵https://en.wikipedia.org/wiki/Edit_distance

¹⁶https://en.wikipedia.org/wiki/Earth_mover%27s_distance



Pomyśl teraz o naturze, o organizmach żywych i o biologicznym mapowaniu genotyp $\rightarrow$ 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 (Rys. 2.8). 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 $\rightarrow$ 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.

Możliwe powody skłaniające do stosowania nietrywialnego mapowania [Ben99]:

- ograniczenie (zmniejszenie) przestrzeni przeszukiwania (rekurencyjne, hierarchiczne, itd.),
- lepsze próbkowanie przestrzeni przeszukiwania (tworzące topologię zwiększającą FDC – opis w rozdziale 2.5.2),
- bardziej złożone rozwiązania w przestrzeni fenotypów („procedura wzrostu” zapisana w genotypie),
- lepsza obsługa ograniczeń (mapowanie **każdego** genotypu na poprawny fenotyp),

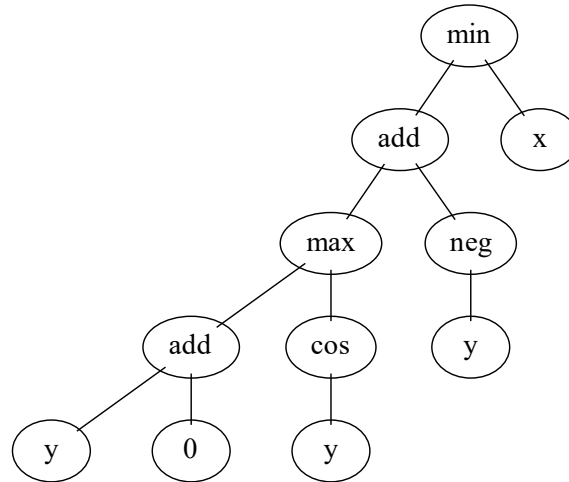
oraz:

- 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).

ale:

- 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 niezmiennie, zaprojektowane przez człowieka reguły odwzorowujące genotyp w jego znaczenie (*external embryogeny*). Więcej informacji – patrz rozdział ?? o optymalizowaniu konstrukcji/ewolucyjnym projektowaniu.



Rysunek 2.9: Wyrażenie $\min(\text{add}(\text{max}(\text{add}(y, 0), \cos(y)), \text{neg}(y)), x)$ czyli $\min(\text{max}(y + 0, \cos(y)) + (-y), x)$ czyli $\min(x, \text{max}(y, \cos(y)) - y)$.

2.6 Programowanie genetyczne (*genetic programming*)

Programowanie genetyczne¹⁷ służy do optymalizacji wyrażeń i programów. Cechą charakterystyczną jest drzewiasta reprezentacja rozwiązań pozwalająca na zakodowanie programów – Rys. 2.9, choć istnieje też mniej popularny wariant z kodowaniem liniowym.¹⁸

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 być funkcjami lub akcjami bezargumentowymi.

Pożądane są dwie cechy zbiorów F i T :

1. *domknięcie* (ang. *closure*) – każda funkcja działa dla dowolnych wartości i typów

¹⁷Darmowa książka: http://www0.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/poli08_fieldguide.pdf

¹⁸https://en.wikipedia.org/wiki/Linear_genetic_programming

argumentów zwracanych przez dowolną funkcję lub terminal,

2. *wystarczalność* (ang. *sufficiency*) – dostępne w obu zbiorach elementy pozwalają na zbudowanie rozwiązania stawianego problemu.

```
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')
```

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... (tutaj wspomnienie długiej symulacji numerycznej 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)
```

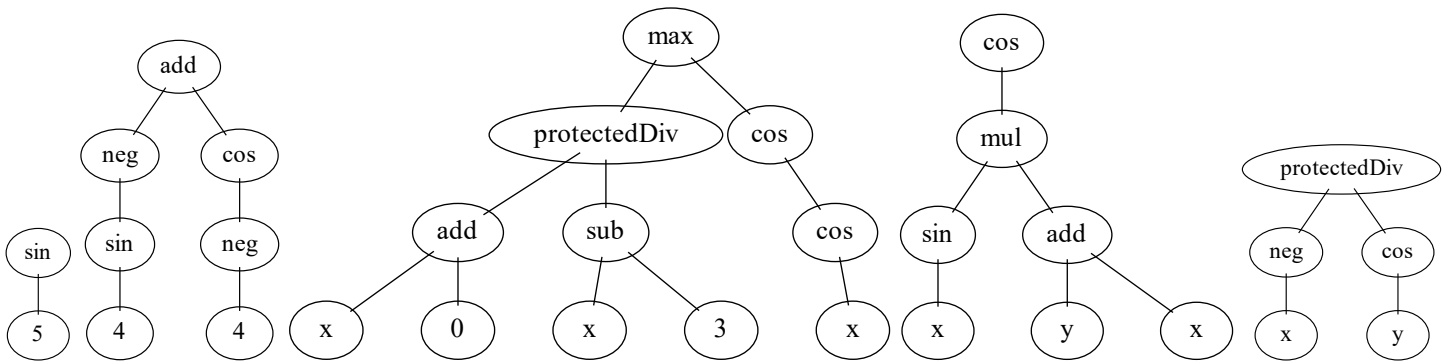
Jeśli nie zapewnimy *wystarczalności*, GP będzie starało się znaleźć (najlepsze) przybliżenie rozwiązania za pomocą dostępnych środków.

Podstawowe metody tworzenia populacji początkowej:

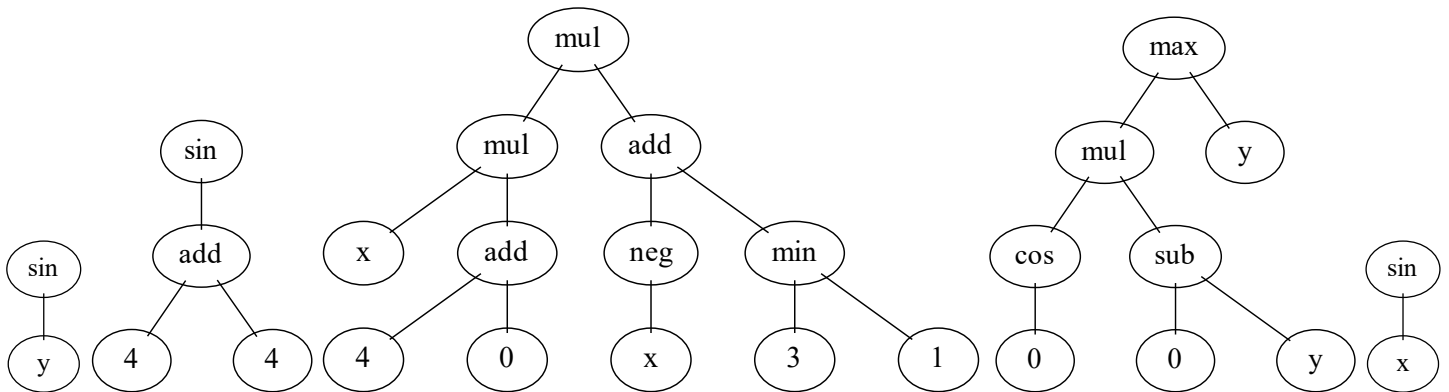
- *Full*: wybieraj węzły z **F** jeśli głębokość jest poniżej ustalonego progu, a jeśli nie, to z **T**. Wszystkie drzewa będą miały taką samą głębokość – przykłady na Rys. 2.10.

- *Grow*: wybieraj węzły z $F \cup T$ jeśli głębokość jest poniżej ustalonego progu, a jeśli nie, to z T . Drzewa będą miały różną głębokość i kształt – przykłady na Rys. 2.11.
- *Ramped half-and-half*: połowa populacji metodą *full*, połowa metodą *grow* – zapewnia zróżnicowanie populacji początkowej.

```
toolbox.register("expr", gp.genHalfAndHalf, pset=pset, min_=1, max_=2) #
    tree height range
toolbox.register("individual", tools.initIterate, creator.Individual,
    toolbox.expr)
toolbox.register("population", tools.initRepeat, list, toolbox.individual
    )
```



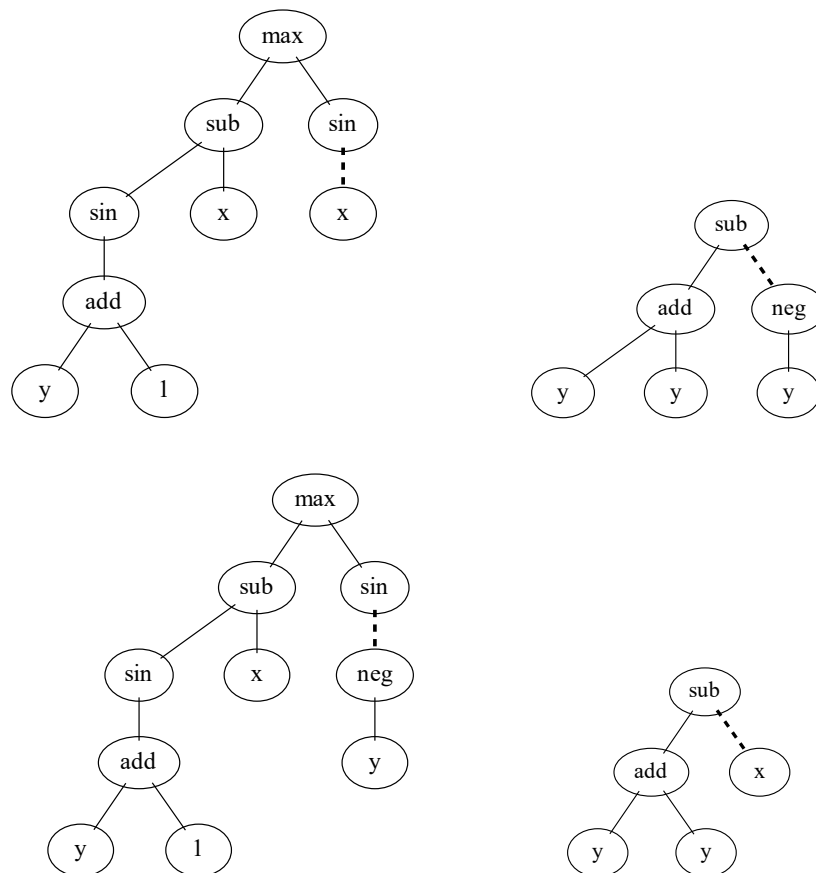
Rysunek 2.10: Pięć osobników wygenerowanych metodą *Full*, `gp.genFull(pset,1,3)` (DEAP wymaga dwóch parametrów, nie jednego) dla $T=\{x, y, 0, 1, 2, 3, 4, 5\}$.



Rysunek 2.11: Pięć osobników wygenerowanych metodą *Grow*, `gp.genGrow(pset,1,3)`, dla $T=\{x, y, 0, 1, 2, 3, 4, 5\}$. 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 w GP to najczęściej wymiana losowo wybranych poddrzew u obu rodziców (Rys. 2.12).

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



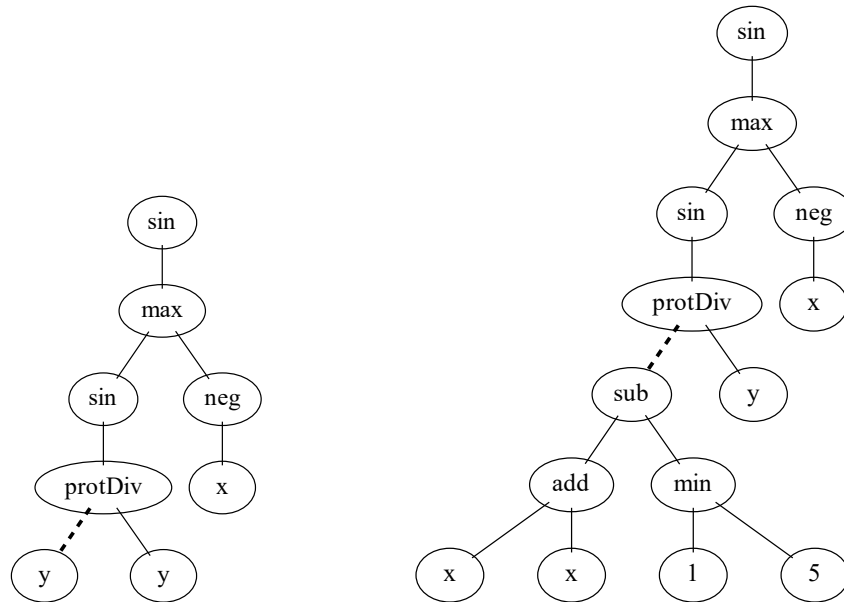
Rysunek 2.12: Krzyżowanie w GP. U góry rodzice wygenerowani metodą `gp.genGrow(pset,2,4)`. Na dole dzieci utworzone metodą `gp.cxOnePoint(parent1,parent2)`.

Typowa mutacja to wybranie losowego miejsca w oryginalnym drzewie i zastąpienie poddrzewa nowym, wygenerowanym jedną z powyżej opisanych metod (Rys. 2.13).

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

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))
```



Rysunek 2.13: Mutacja w GP. 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)`.

```
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.

2.6.1 Regresja symboliczna

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 funkcji¹⁹, a nawet szukać pewnych klas funkcji lub dowolnych funkcji (stąd ta metoda regresji nazywana jest *symboliczną*).

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

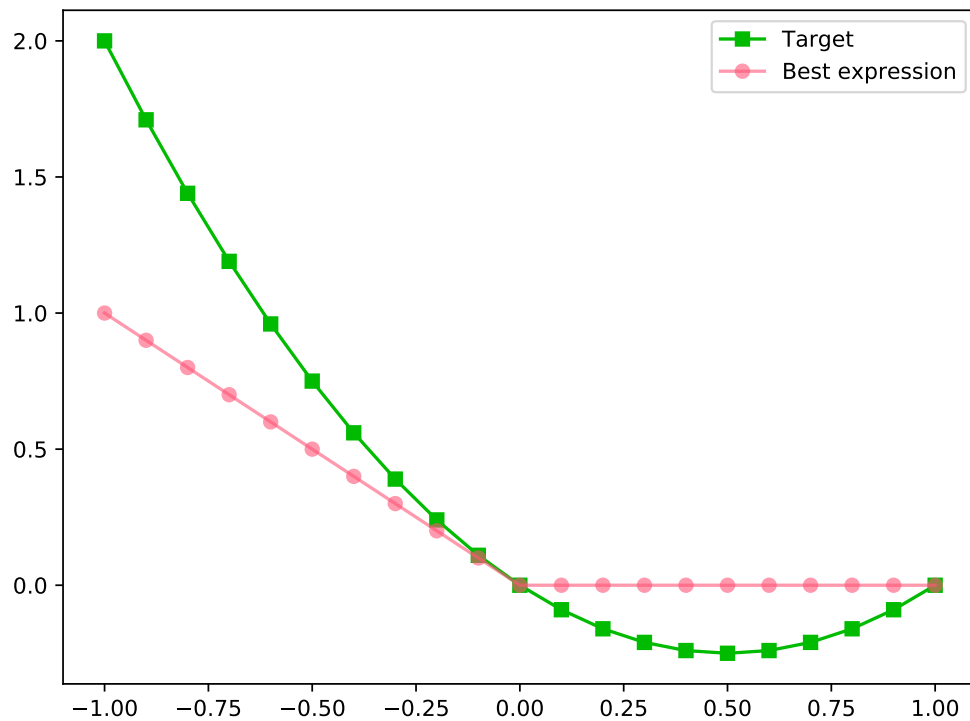
def eval_expr(individual, points):
    # transform the tree expression into a callable function
    func = toolbox.compile(expr=individual)
    # evaluate the mean squared error between the expression and the
    target function
    sqerrors = ((func(x) - target_function(x))**2 for x in points)
    return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

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

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

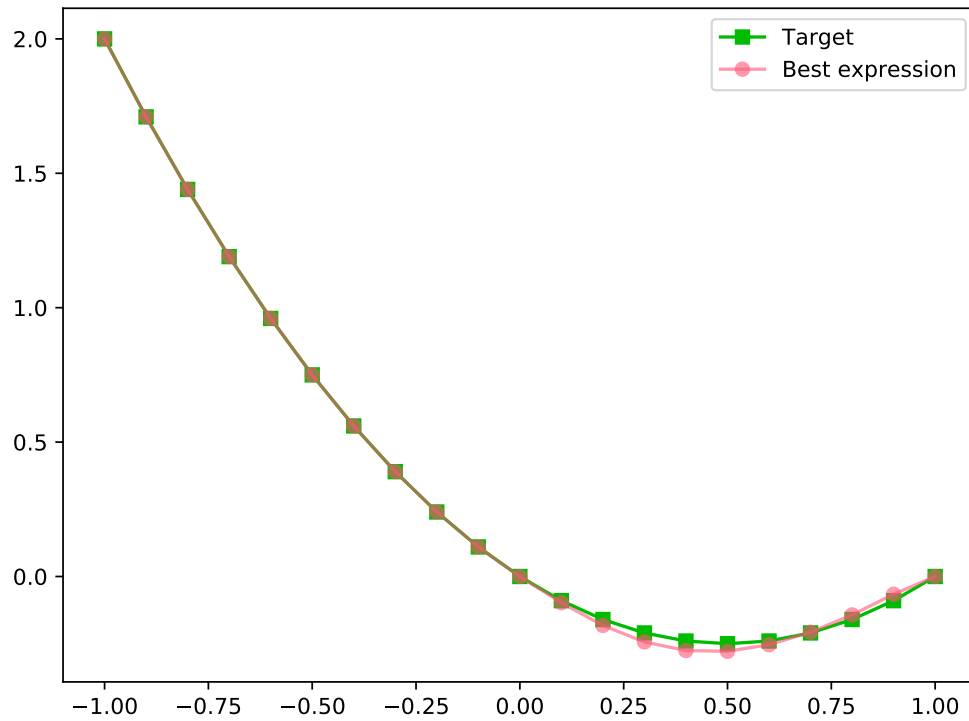
Przykładowy eksperyment #1: Znajdź wyrażenie najlepiej opisujące zbiór punktów należących do funkcji $f(x) = x^2 - x$. Pamiętajmy, że w praktyce funkcja ta jest nieznana i chcemy ją odkryć! Do dyspozycji GP dajemy to co w przykładowych źródłach powyżej, czyli oprócz x operatory neg, +, −, *, /, max,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

¹⁹<http://www.alife.pl/programowanie-genetyczne>



Rysunek 2.14: Rozwiązanie najlepsze w pierwszym pokoleniu (czyli w losowo wygenerowanej populacji).

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



Rysunek 2.15: 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)))
```

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 $\{x_1, x_2, y\} = \{(0, 0, 0); (0, 1, 1); (1, 0, 1); (1, 1, 0)\}$.

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

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

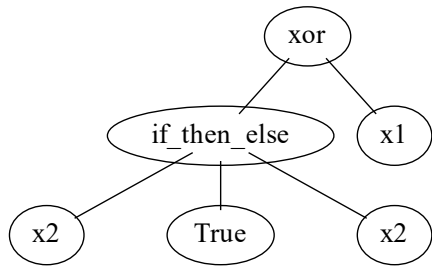
pset = gp.PrimitiveSetTyped("main", [bool, bool], bool) # let's use
    strongly-typed GP as an example
pset.addPrimitive(operator.xor, [bool, bool], bool)
pset.addPrimitive(operator.or_, [bool, bool], bool)
pset.addPrimitive(operator.and_, [bool, bool], bool)
pset.addPrimitive(operator.not_, [bool],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

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,
```

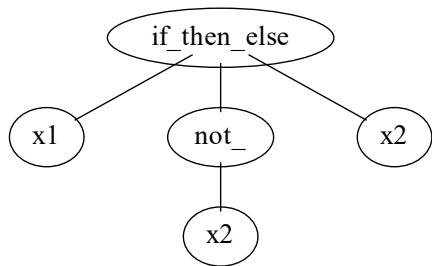
W tym eksperymencie GENERATIONS=100 oraz POPSIZE=150, a w przypadku niepowodzenia – kolejna próba z POPSIZE=1500.

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



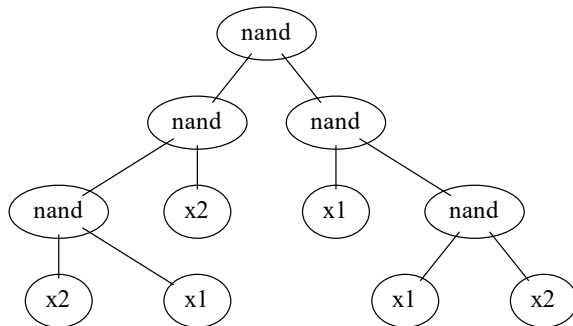
- Samo if-then-else: brak idealnego rozwiązania (najmniejszy błąd = 1)

- Tylko if-then-else oraz not:
if_then_else(x1, not_(x2), x2)

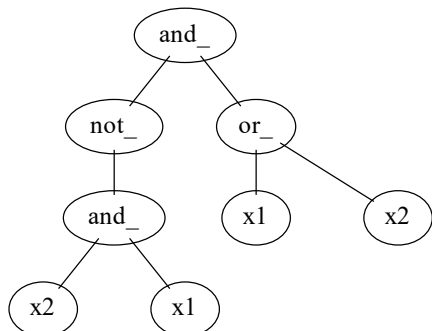


- Tylko not oraz and: brak idealnego rozwiązania (najmniejszy błąd = 1)

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

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

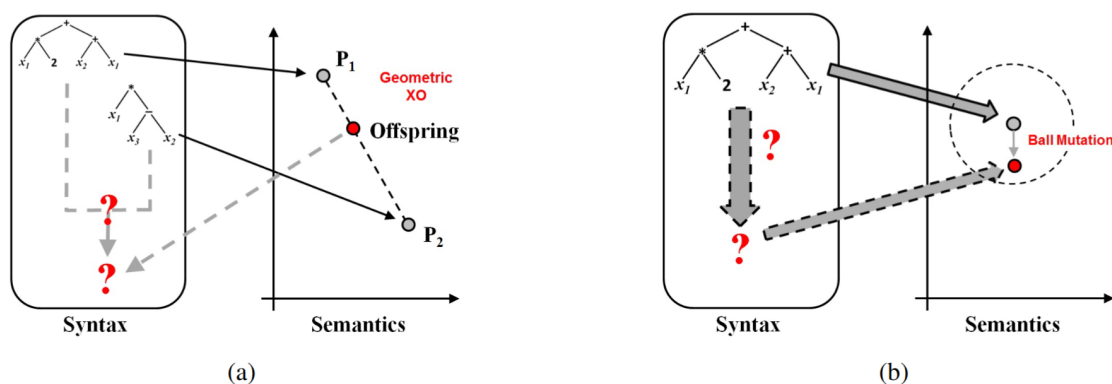


Figure 1: Geometric semantic crossover (plot (a)) (respectively geometric semantic mutation (plot (b))) performs a transformation on the syntax of the individual that corresponds to geometric crossover (respectively geometric mutation) on the semantic space. In this figure, the unrealistic case of a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

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

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

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

Odkrycia ewolucji – Rys. 6:

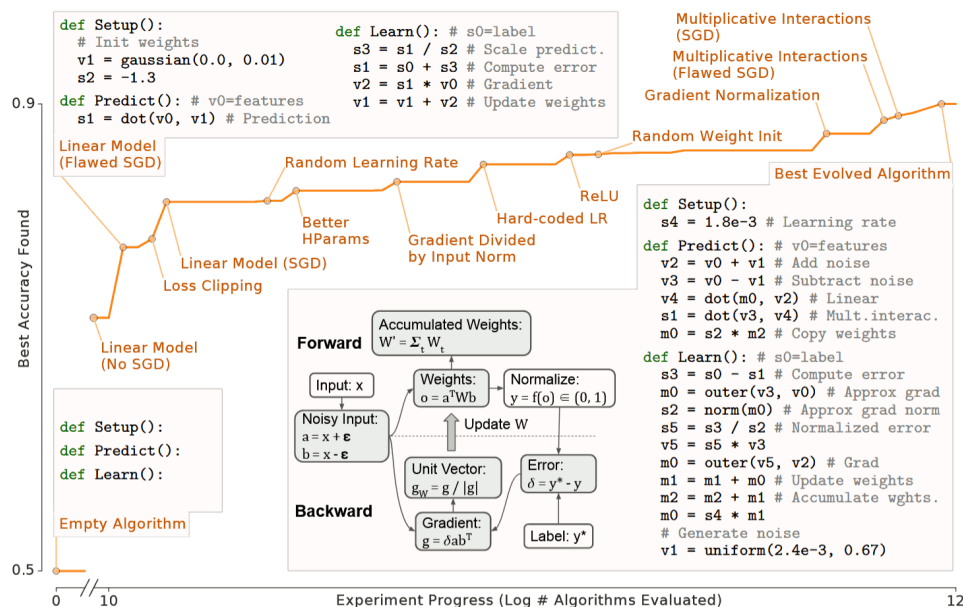


Figure 6: Progress of one evolution experiment on projected binary CIFAR-10. Callouts indicate some beneficial discoveries. We also print the code for the initial, an intermediate, and the final algorithm. The last is explained in the flow diagram. It outperforms a simple

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

2.6.2 Hiperheurystyki i samo-programujące się algorytmy

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 hiperheurystyki²⁰ przeszukujące przestrzeń heurystyk i ich kombinacji [Ros05; ÖBK08; Bur+10].

Przykładowe pytania

Przypomnij sobie warianty algorytmów ewolucyjnych przedstawione we wcześniejszych rozdziałach oraz ich skrótowe oznaczenia i upewnij się, że odróżniasz je od siebie – przypomnij sobie ich wyróżniające cechy.

²⁰<http://en.wikipedia.org/wiki/Hyper-heuristic>

2.7 Mechanizmy inspirowane naturą

2.7.1 Reprezentacja

Powielanie genów

Dotychczas mówiliśmy o genotypach haploidalnych. W przyrodzie występują również genotypy diploidalne, składające się z jednej lub wielu par chromosomów (u człowieka są **23 pary**), oraz **poliploidalne**. Taka reprezentacja wymaga specjalnych operacji, takich jak **segregacja** (wybór chromosomów z par), **translokacja** (przemieszczenie genów do innego chromosomu), **duplikacja** (podwojenie genu i umieszczenie kopii w chromosomie) i **delecja** (usunięcie zduplikowanego egzemplarza genu). Ponadto występują problemy związane z interpretacją osobnika (z powodu nadmiarowości informacji w genotypie). W naturze istnieje mechanizm **dominacji** rozstrzygający o tym, które geny uzewnętrzniają swoje znaczenie; dominacja może też zachodzić nie tylko pomiędzy chromosomami, ale również wewnątrz nich.

Dyskusja: kiedy nadmiarowość informacji w genotypie może być zaletą i pomagać w optymalizacji?

x'_1	x'_2	x'_3	x'_4	...
x''_1	x''_2	x''_3	x''_4	...
...				

W AE nadmiarowość informacji to pamięć genetyczna: choć niektóre cechy nie objawiają się zewnętrznie, są przechowywane i przekazywane. Przyroda przechowuje rozwiązania, które kiedyś okazały się korzystne, w postaci ukrytych genów, które ulegają dominacji. Geny takie w przyszłości, przy odpowiednich warunkach, mogą zacząć same dominować.

Zastosowanie podobnego mechanizmu w AE ma sens, kiedy funkcja celu jest zmienna – niestacjonarna (odpowiada to zmiennym warunkom środowiska). Stwierdzono eksperymentalnie [SW15, Fig. 2, 3], że w takim przypadku algorytm z diploidalną reprezentacją osobników jest w stanie szybciej dostosować się do zmian środowiska (szczególnie okresowych) znajdując dobre rozwiązania.

Istnieje wiele mechanizmów dominacji: stałe, zmienne, losowe, deterministyczne, dominowanie lepszego chromosomu itd. W tym ostatnim przypadku przerobienie tradycyjnego, haploidalnego algorytmu na poliploidalny jest bardzo proste: to co było haploidalnym genotypem staje się chromosomem, genotyp składa się z wielu chromosomów i jego jakość (dla selekcji) jest jakością najlepszego chromosomu, selekcja działa na całych genotypach, a mutacja i krzyżowanie operują na chromosomach tak, jak dawniej na haploidalnych genotypach.

Relacja gen–fen

W naturze występują:

- geny nadmiarowe: nie mają wpływu na fenotyp osobnika,
- efekt plejotropowy: pojedynczy gen może wpływać na wiele cech fenotypowych,
- efekt poligeniczny: pojedyncza charakterystyka fenotypowa osobnika może być determinowana przez jednocześnie oddziaływanie wielu genów.

Tymczasem, w klasycznym AG: jeden gen — jedna cecha. Można zaimplementować nadmiarowość, plejotropowość i/lub poligeniczność o ile jest się dobrze świadomym konsekwencji oraz wad i zalet każdego z tych mechanizmów – przypomnij sobie pojęcie epistazy ze str. 25.

2.7.2 Operatory

Inwersja (odwrócenie kolejności losowego podciągu genotypu) pozwala na odkrycie korzystnych ustawień i połączeń genów (w sensie ich znaczenia). Chodzi tu o współpracę z operacją krzyżowania, która rozcina łańcuchy genów, rozcinając zarazem długie schematy (zob. twierdzenie o schematach, rozdział 2.2.7). Fragmenty genotypu związane ze sobą nieliniowo i leżące w dużej odległości od siebie będą zwykle rozdzielane przy krzyżowaniu, choć razem mogą stanowić istotne i korzystne połączenie. Operacja krzyżowania w obecności inwersji nie jest już jednak tak oczywista: ponieważ przodkowie posiadają różną kolejność znaczeń genów, po rozcięciu i zamianie odcinków potomkowie zwykle nie mają pełnego garnituru genów (przypomnij sobie, dlaczego taki problem nie występuje przy opisanej w rozdziale 2.2.4 operacji „tasowania”, ale występuje w nieporządnym AG z rozdziału 2.2.9). Dlatego ogranicza się zbiór rodziców, którzy mogą być ze sobą skrzyżowani (do takich, którzy dadzą poprawne potomstwo), lub stosuje się różne złożone operacje krzyżowania.

Dopełnienie częściowe polega na zastąpieniu w wybranych osobnikach pewnej części genów wartościami przeciwnymi – dopełnieniami.²¹ Celem jest zwiększenie różnorodności osobników i ochrona przed zbytnim ukierunkowaniem populacji. Choć cel zostaje osiągnięty, to odbywa się to kosztem spadku prędkości zbieżności algorytmu genetycznego.

Dodatkowe mechanizmy dotyczące operatorów:

²¹Pierwszy raz zaproponowane w [Fra72, str. 70], ta operacja została nazwana przez autora „migracją”, ponieważ odpowiada napływowi osobników tego samego gatunku z zewnątrz demu (imigranci) do tego demu (tubylcy). [https://pl.wikipedia.org/wiki/Dem_\(biologia\)](https://pl.wikipedia.org/wiki/Dem_(biologia)) Osobniki w demie i poza nim są zgodne genetycznie, ale są przystosowane do innych środowisk, zatem allele imigrantów nie są przystosowane do warunków demu, do którego napływają – czyli różnią się od alleli tubylców. To powoduje wzrost różnorodności genetycznej w demie po krzyżowaniu imigrantów z tubylcami.

- Zmienne (adaptacyjne) prawdopodobieństwa operatorów: szczególnie przydatne przy niestacjonarnych (zmiennych) funkcjach celu! [\[przykład stałej funkcji\]](#)
- Adaptacyjne operatory: są umieszczane w genotypach osobników i wraz z nimi podlegają ewolucji.
- Krzyżowanie z uwodzeniem: preferencje osobnika zapisane w jego genotypie i ewoluujące.

Pomysły testowane w [Kwa97, opis na str. 142]:

- Tranzycja: przeniesienie pojedynczej kopii genu z jednego genotypu do innego, gdzie umieszczany jest jako gen nadmiarowy.
- Transpozycja: gen nadmiarowy zamienia się miejscem z genem fenotypowym (tj. mającym wpływ na fenotyp).
- Rekrudescencja: u niewielkiej liczby osobników w każdym pokoleniu występuje zwiększone prawdopodobieństwo mutacji i transpozycji. Po rekrudescencji występują znaczne zmiany fenotypowe i większość osobników jest kiepska, ale czasem pojawiają się „*hopeful monsters*”.
- Kryzys (katastrofa): w losowym momencie czasu w całej populacji zachodzi znaczna reorganizacja genotypów. Skutki – wymarcie populacji lub możliwość zasiedlenia nowego szczytu w krajobrazie przystosowania.
- Różna intensywność mutacji („makromutacje” i „mikromutacje”).

Naśladując naturę w implementacjach należy pamiętać, że nie wszystkie jej mechanizmy zostały poznane i wyjaśnione. Istnieją liczne problemy w analizie ewolucji naturalnej – grupy naukowców posiadają własne, rozbieżne teorie: *punctuated equilibria*²², jak powstają nowe gatunki, jak działa ewolucja i na jakim poziomie (poziomach?) – gen, osobnik, grupa osobników/stado, mem, populacja/gatunek.

2.7.3 Funkcja celu i logika algorytmu

Zmienna liczebność populacji (np. o rząd wielkości) – np. czas istnienia osobnika w populacji zależy od jego wartości funkcji oceny. Samodostrajanie wielkości populacji; „eksplozje demograficzne” – szukanie optimów.

Algorytmy kulturowe²³ – „ewolucja w ewolucji”. Osobniki poprawiają swoje możliwości

²²https://en.wikipedia.org/wiki/Punctuated_equilibrium

²³https://en.wikipedia.org/wiki/Cultural_algorithm

uczenia się przez doświadczenie wyniesione z procesu ewolucyjnego (jak ewolucja kulturowa społeczeństw ludzkich²⁴, czego dowodem miałyby być ponadwykładniczy wzrost poziomu rozwoju ludzkości).

Starzenie się osobników – np. usuwanie najstarszych osobników, żeby zrobić miejsce dla nowych. Dyskusja: pomyśl, jaki to będzie miało efekt dla procesu przeszukiwania w porównaniu do usuwania najgorszych osobników i do usuwania losowych osobników?

Nazwana przez autorów „ewolucją regularyzowaną” [Rea+19], może ograniczać problemy wynikające z niedeterministycznej (zazsumionej) oceny („szczęściarze” i „pechowcy” poddawani selekcji) zwiększając różnorodność i sprzyjając eksploracji [Yin+19].

Klonowanie – inna prosta metoda radzenia sobie z niedeterministyczną (zazsumioną) oceną. Żeby nie tracić zasobów obliczeniowych na wielokrotne ocenianie każdego osobnika i uśrednianie ocen, można powtarzać ocenianie proporcjonalnie do jakości osobnika: oprócz mutacji i krzyżowania – operator klonowania i uśrednianie oceny klonów, zobacz [KR01, rys. 12].

Specjalizacja i specjacja

Dyskusja: jaką znasz odpowiedź na pytanie „po co jest podział na płcie w naturze?”

Szczególnym przypadkiem specjalizacji jest w naturze **zróżnicowanie płciowe osobników**. Ponieważ dotychczas nie ma w biologii jednego, pewnego wyjaśnienia tego fenomenu²⁵ ²⁶, rozważmy teoretycznie bardzo prosty model [Gol03, str. 196], aby zrozumieć wady i zalety zróżnicowania płciowego. W modelu tym zakłada się, że przeżycie potomków s zależy od tego, jaki procent czasu rodzice poświęcą na dwa rodzaje zadań (np. polowanie h i opiekę n): $s = n \cdot h$. Czas, którym dysponuje rodzic jest stały i może być dowolnie dzielony między te zadania. Wprowadzamy dodatkowo opcjonalny (regulowany parametrem a) koszt czasowy związany z „przełączaniem się” rodzica między zadaniami: anh . Podział czasu rodzica spełnia równanie $n + h + anh = 1$.

Gdy nie ma zróżnicowania płciowego, najlepiej jest rodzicowi podzielić czas po równo pomiędzy czynności, $n = h = \frac{1}{2}$. Wzrost a pogarsza s . Z kolei gdy występuje zróżnicowanie płciowe, para rodziców zajmuje się potomkiem i $a = 0$, nie ma znaczenia jak dokładnie oboje rodzice podzielią własne czasy – maksymalne s osiągną dając wspólnie potomstwu taki sam czas n i h . Gdy $a > 0$, natychmiast optymalne stają się tylko takie sytuacje, kiedy każdy z rodziców poświęca cały swój czas na inną czynność (a więc maksymalna specjalizacja). Dzięki eliminacji kosztu „przełączania się” między zadaniami, przypadek zróżnicowania

²⁴https://pl.wikipedia.org/wiki/Koewolucja_genetyczno-kulturowa

²⁵https://en.wikipedia.org/wiki/Evolution_of_sexual_reproduction

²⁶<https://web.archive.org/web/20200128050806/http://archiwum.wiz.pl/1999/99113000.asp>

płciowego rodziców pozwala uzyskać wyższe s , niż odpowiadający mu przypadek braku zróżnicowania.²⁷

Zróżnicowanie płciowe to specjalizacja dwóch rodzajów osobników. W przyrodzie specjalizacja występuje również pod postacią podziału organizmów na **gatunki** (**specjacja**), z których każdy ma swoje miejsce (**niszę**) i zadania [Gol03, str. 200]. Podobne zjawisko może być korzystne w AE: jeśli funkcja posiada kilka *optimów*, możemy oczekiwać, by pod koniec ewolucji osobniki nie gromadziły się w jednym z nich, a dzieliły się równo pomiędzy *optima* (albo proporcjonalnie do jakości lokalnych *optimów*). Osobniki podobne do siebie, zajmujące otoczenie pewnego ekstremum mogą być nazwane gatunkiem.

Efekt powstawania i utrzymania gatunków można uzyskać na wiele sposobów, na przykład:

- stosując zastępowanie przez potomka najbardziej podobnego osobnika (opisana w rozdziale 2.2.2 selekcja według modelu ze współczynnikiem zatłoczenia),
- modyfikując wartość **przystosowania** w zależności od **podobieństwa** osobników.
nowa ocena := pierwotna ocena / (suma podobieństw do pozostałych osobników)
Popularna, prosta i skuteczna metoda! Więcej na temat podobnych metod w rozdziale o sterowaniu różnorodnością.

Utrzymywanie gatunków potrafi ostatecznie doprowadzić do odkrycia lepszego rozwiązania, niż w standardowej ewolucji bez modyfikacji wartości przystosowania. Ponadto jest ono korzystne przy optymalizacji wielokryterialnej, kiedy chcemy, by osobniki-rozwiązania pokryły równomiernie front rozwiązań niezdominowanych. Przy stosowaniu specjacji zalecane może być ograniczenie krzyżowania do osobników tego samego gatunku, jeśli krzyżowanie osobników bardzo odmiennych daje słabe rozwiązania.

Koewolucja

Naśladownictwo natury przejawia się w eksperymentach z systemami **współewoluującymi**, w których istnieje kilka populacji wzajemnie na siebie wpływających. Ocena rozwiązań jednej populacji może zależeć od rozwiązań znajdujących się w innej populacji. Koewolucja może posłużyć do wyznaczenia optymalnej strategii, która jest zależna od strategii innych populacji – wszystkie populacje dążą do przewyższenia pozostałych lub współpracy. Taka sytuacja prowadzi do powstania zachowań typowych dla pasożytów, drapieżników, a także wprowadza element konkurencji lub kooperacji (→ rozdział o algorytmach koewolucyjnych, 2.12).

²⁷Szwecja: do 1 500 € premii za podział urlopu rodzicielskiego po równo między rodziców.

2.8 Systemy klasyfikatorowe (CFS/LCS/GBML)

LCS is a simple example of a *cognitive architecture*. A cognitive architecture can mean:

- a theory about the structure of the human mind,
- an implementation of such a theory (used in AI) – a cognitive system or agent.

Possible functional criteria of such architectures include flexible behavior, real-time operation, rationality, large knowledge base, learning, development, adaptation, modularity, linguistic abilities, and self-awareness. Competencies and behaviors demonstrated by such systems include – similarly to AGI – perception, memory, attention, actuation, social interaction, planning, motivation, emotion, development and using knowledge efficiently to perform new tasks. Components include memory storage, control components, data representation, and input/output devices [KT20]. More in Sect. ??.

John Holland envisioned a cognitive system [HR78] capable of classifying the goings on in its environment, and then reacting to these goings on appropriately²⁸. To build such a system²⁹ (see Fig. 2.16) we need

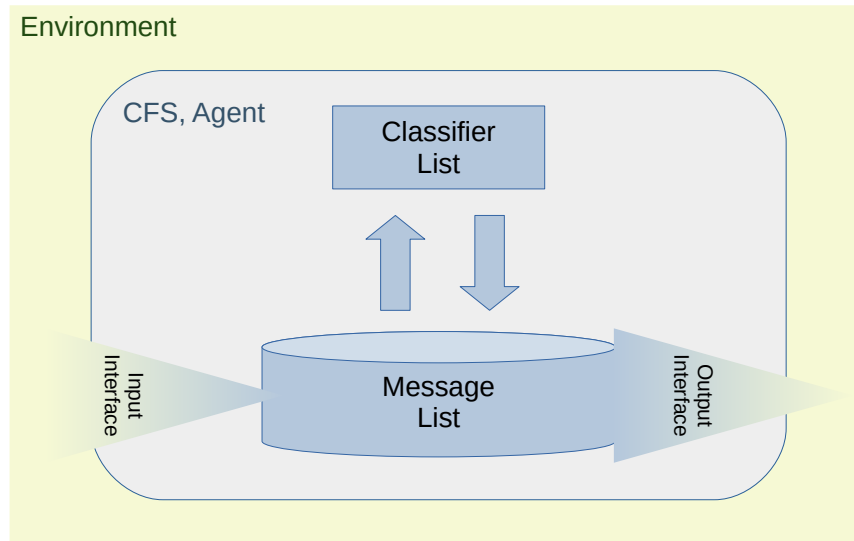
- (1) an environment;
- (2) receptors/sensors that tell our system about the goings on;
- (3) effectors/actuators that let our system manipulate its environment; and
- (4) the system itself that has (2) and (3) attached to it, and “lives” in (1).

CFS is quite a general and versatile architecture – consider the following three examples:

- (4) can be a real or simulated robot or creature: (1) is a world with “food” (something beneficial, reward) and “poison” (something detrimental, penalty), and a robot walking (3) across this environment and trying to learn to distinguish (2) between these two items, and to survive while maximizing reward.
- (4) can be a computer. It has inputs (2), outputs (3), and a message passing system in-between, converting certain input messages into output messages, according to a set of rules, usually called a *program*.
- (4) can be a machine learning (ML) algorithm. Inputs (2) provide values of conditional attributes, outputs (3) send out a value of the decision attribute, and a message passing

²⁸With minor updates and corrections, from <https://www.cs.cmu.edu/Groups/AI/html/faqs/ai/genetic/part2/faq-doc-5.html> and from no longer available parts of online slides by Riccardo Poli.

²⁹https://en.wikipedia.org/wiki/Learning_classifier_system



Rysunek 2.16: The architecture of the classifier system.

system in-between is the ML model that maps inputs to outputs (predicts outputs based in inputs).

The input interface (2) generates messages – strings of symbols that are written on the message list. Then these messages (internal and external) are matched against the condition-part of all classifiers (“if-then” rules), to find out which actions are to be triggered. The message list is then emptied, and the encoded actions, themselves just messages, are posted to the message list. Then, the output interface (3) checks the message list for messages concerning the effectors. Then the cycle restarts.

You may start from scratch (from tabula rasa – without any knowledge) using a randomly generated classifier population, and let the system learn its program by induction. The input stream are input patterns that must be repeated over and over again, to enable the agent to classify its current situation/context and react on the goings on appropriately, as in the example below:

```

IF small, flying object to the left THEN send @
IF small, flying object to the right THEN send %
IF small, flying object centered THEN send $
IF large, looming object THEN send !
IF no large, looming object THEN send *
IF * and @ THEN move head 15 degrees left
IF * and % THEN move head 15 degrees right
IF * and $ THEN move in direction head pointing
IF ! THEN move rapidly away from direction head pointing

```

Classifier list is a list of classifiers:

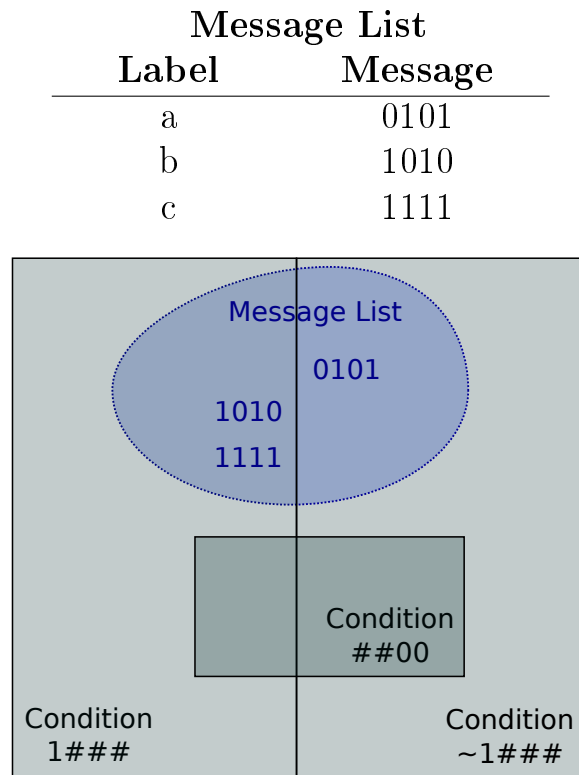
```
IF cond-1 AND cond-2 ... AND cond-N THEN action
```

```
cond-1, cond-2, ... cond-N / action
```

(we use a shorter notation, “,” means “AND”).

For now, let us assume the simplest language for messages and conditions – they will be encoded by $\{0, 1, \#\}$. In a real application, we would use the natural language (set of symbols) – this is analogous to the GA/EP difference.

A message matches a condition if all its 0's and 1's are in the same positions as in the condition string. A negated condition is satisfied if no message in the message list matches it:



Condition	Matched by	Satisfied	Negation satisfied
0101	a	Yes	No (~ 0101)
1101		No	Yes (~ 1101)
#101	a	Yes	No ($\sim \#101$)
1####	b, c	Yes	No ($\sim 1####$)
##00		No	Yes ($\sim ##00$)
#####	a, b, c	Yes	No ($\sim #####$)

In CFSs actions are strings of fixed length built from characters in the alphabet $\{0, 1, \#\}$; their length is usually the same as that of messages. Action strings can be interpreted as parameterized assertions (messages) that go into the message list. When a classifier is activated, a message is built using the following procedure:

- 0's and 1's in the action string are simply copied in the message
- #'s are substituted by the corresponding characters in the message that matches the first condition in the condition part. For this reason the # character is also called the pass-through operator.

Example:

Message List	
Label	Message
a	0101
b	1010
c	1111

Classifier List	
Label	Classifier
i	#11#, ~#110 / 00##
ii	###1, ~#110 / ###0
iii	##1#, ~1110 / 0##0

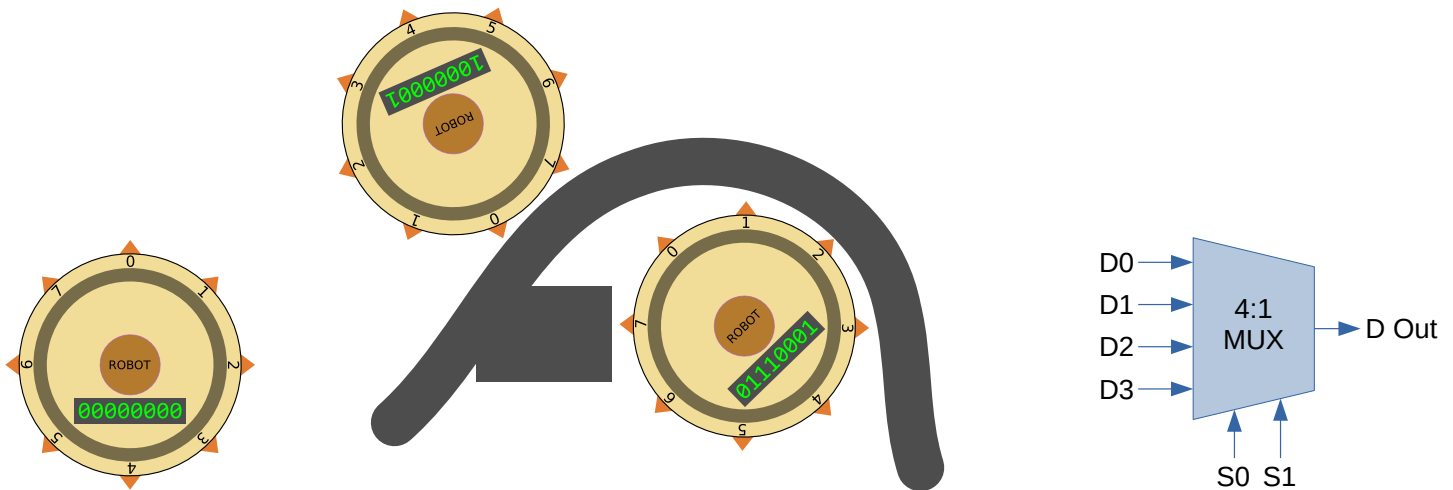
The following set of messages will be produced:

Message	Reason
0011	Posted by i, c matches cond-1
0100	Posted by ii, a matches cond-1
1110	Posted by ii, c matches cond-1
0010	Posted by iii, b matches cond-1
0110	Posted by iii, c matches cond-1

The only actions allowed in the basic CFS are assertions, so messages (facts) cannot be explicitly deleted. However, as the message list is of finite size, old messages can be overwritten. Many classifiers can be activated in parallel by the messages in the message list. A classifier can post as many messages as the number of messages matching its first condition. Conflict resolution is only necessary if the active classifiers can produce more messages than entries in the message list.

2.8.1 Input and output interfaces

The input interface can be thought of as a mechanism by which the CFS can obtain information about the environment. The messages posted by the input interface are often descriptions of the state of a set of (binary) detectors that can sense various features of the environment.



Rysunek 2.17: Simple examples of CFS control: a robot with eight proximity sensors and a 4:1 multiplexer.

The output interface can be realized by any procedure capable of selecting (deleting) and using some messages in the message list. Usually we imagine that the bits in a message picked up by the output interface represent (and control) the state of a set of effectors which act on the environment, for example to control the actions of a robot.

In any case the output interface must be able to recognize which messages are input messages posted by the input interface, which are internal messages posted by classifiers, and messages meant to be output messages. This is obtained by using tags usually consisting of two additional bits in the messages that are interpreted in a special way. An example convention:

Tag	Interpretation
01	Internal Message
11	Internal Message
00	Output Message
10	Input Message

Or:

bit 1: Input/Output (from/to)

bit 2: Inside/Outside

For example, a CFS which controls a robot (Fig. 2.17, left) might have some classifiers devoted to obstacle avoidance, like

```
10 1##### / 00 0100 0000
10 ##1##### / 00 0001 0000
```

```
10 ####1### / 00 1000 0000
10 #####1# / 00 0010 0000
```

The conditions of these simple classifiers just check whether there is an obstacle in front, on the right, on the back, on the left, respectively. The action-strings have the following interpretation:

- The first two bits indicate that the messages are for the effectors in the output interface
- The following four bits indicate which movement (forward, backward, right, left) is appropriate to avoid the obstacle
- The other bits are padding.

In the case of the 6-bit 4:1 multiplexer (Fig. 2.17, right), the following classifier list would provide the correct behavior:

```
10 00 0### / 00 0 00000
10 00 1### / 00 1 00000
10 01 #0## / 00 0 00000
10 01 #1## / 00 1 00000
10 10 ##0# / 00 0 00000
10 10 ##1# / 00 1 00000
10 11 ###0 / 00 0 00000
10 11 ###1 / 00 1 00000
```

- The first two bits of the conditions mean “input message”
- The next two bits of the conditions are interpreted as address (i.e., selector) bits
- The other characters of the conditions as checks for the state of the data lines
- The first two bits of the action are interpreted as the “output message” tags
- The third bit as the output of the multiplexer
- The remaining bits as padding.

2.8.2 Main cycle

1. Activate the input interface and post the input messages it generates to the message list.
2. Perform the matching of all the conditions of all classifiers against the message list.

3. Activate the fireable classifiers (those whose conditions are satisfied) and add the messages they generate to the message list.
4. Activate the output interface, i.e. remove the output messages from the message list and perform the actions they describe; go to 1.

In the previous two examples we have considered non-overlapping rulesets, i.e. sets of classifiers in which one and only one classifier is active in the presence of a given message in the message list. An alternative, more parsimonious way of using classifiers is to organize them to form a default hierarchy (pol. *hierarchia domniemań*), in which some very general classifiers provide the default behavior for the system. Other, more specific classifiers refine such a behavior in the presence of messages improperly handled by the default ones.

For example, for the 6-bit multiplexer we could use:

```
10 00 0### / 00 0 00000
10 01 #0## / 00 0 00000
10 10 ##0# / 00 0 00000
10 11 ###0 / 00 0 00000
10 ## #### / 00 1 00000
```

“Unless there is a 0 on the data line currently addressed, set the output to 1”

Default hierarchies are not only more parsimonious ways of programming CFSs, they also make the “search” for a good program much easier (for example we can successively refine the hierarchy). This is very important when we introduce learning and rule discovery in CFSs.

2.8.3 *Learning* Classifier Systems (LCS)

The real power of CFSs derives from the possibility of adding adaptation mechanisms to the basic architecture, so that they can learn to behave appropriately in the environment³⁰ or, more simply, to perform useful tasks. There are two ways in which we can adapt (i.e. improve the performance of) a CFS:

- Adaptation by **credit assignment**: changing the way existing classifiers are used.
- Adaptation by **rule discovery**: introducing new classifiers in the system.

³⁰<http://www.alife.pl/adaptacyjny-system-klasyfikatorowy>

2.8.4 Good and bad classifiers

Not all the classifiers active at a given cycle will in general produce a message which will lead (directly or after additional processing) to a good action. For example, if a CFS controls an autonomous agent who has to find a source of energy (i.e. food) to survive, some classifiers will post actions which will lead to get some food; others will post actions which will delay the search for food. In summary, in a CFS there are usually some classifiers which are better than the others.

2.8.5 The need for competition

Unfortunately, the basic CFS is absolutely, blindly fair and gives to all the fireable classifiers the same chances of posting messages, and therefore of influencing the overall behavior of the system. To maximize the performance of a CFS it would be nice to give higher priority to the messages posted by good classifiers and low priority to the others. Even better – to prevent low quality classifiers from posting their messages at all if other, better classifiers are fireable. This behavior could be obtained if the classifiers had to compete to post their messages, basing on some measure of quality of classifiers.

2.8.6 Quality of classifiers

There may be several properties of classifiers on which a quality measure can be based. The two most important ones are:

- The usefulness of the classifier in determining the good performance of the whole system: *strength*.
- The relevance of a classifier in a particular situation: the *specificity* of the classifier = its (length – number_of_#’s) / length.

Strength and *specificity* are usually combined into a single measure, the *bid* a classifier makes in the auction (competition):

$$bid = k \cdot strength \cdot specificity \quad (k \text{ is a constant } \approx 0.1)$$

- To maintain parallelism, in the auction there must be more than one winner.
- To avoid premature convergence, we use a noisy (probabilistic) auction, in which classifiers have a bid-proportionate winning probability.

- As a classifier's specificity is a constant, the strengths associated to classifiers are the only quantities that can be varied to influence the auction and therefore the behavior of a CFS.

2.8.7 Adaptation by credit assignment

A learning algorithm for CFS should be capable of modifying the *strengths* of classifiers to optimize the behavior of the system as a whole. To do that, the algorithm will need to have some kind of information about the quality of the behavior of the system. This information will come from the environment, e.g. from an external observer (a teacher), or from some other part of the system, e.g. from an internal variable representing the level of energy of the system. The simplest (more biologically plausible) form of behavioral-quality information would be a scalar value, termed *reward*, whose sign tells the learning algorithm whether the actions of the system are good (positive reward) or bad (negative reward or punishment) and whose magnitude may be fixed (e.g. +1, -1, 0) or variable. If rewards only are available, the learning algorithm will have to solve the so-called credit assignment problem: which classifiers are responsible (and to which extent) for the good or bad overall behavior of the system?

2.8.8 The Bucket Brigade algorithm

The bucket brigade (pol. *brygada kubelkowa/wiaderkowa*) algorithm is a parallel, domain-independent, local credit-assignment-based learning algorithm:

1. If there is a reward (or punishment), add it to the strength of all the classifiers active in the current major cycle.
2. Make each active classifier pay its bid to the classifiers that prepared the stage for it (i.e. posted messages matched by its conditions).

The stage-preparing classifiers had to pay (invest) their bids in the previous cycle (when they were active). In this cycle they get back their "money". In turn, the classifiers that prepared the stage for the stage-preparing classifier received some money two cycles earlier, and so on. Good classifiers are rewarded often so their strengths tend to grow. So, they will make bigger bids, and so they will pay more to their stage-preparing classifiers. In turn those classifiers will be able to pay more to their stage-preparing classifiers, and so on. During time, strength is propagated backwards, and each classifier receives the correct share of credit for the good (or bad) behavior of the system as a whole. With time, strengths reach a (nearly) constant equilibrium value.

Układ oceniający w zadaniu klasyfikowania

Układ oceniający spełnia podstawową rolę w procesie uczenia, gdyż od jego działania zależy szybkość i skuteczność tego procesu. Zadaniem układu oceniającego jest określenie jakości poszczególnych reguł w systemie. Z ocen tych korzysta AG przy tworzeniu następnej populacji reguł, a także użytkownik, przy wykorzystywaniu otrzymanych reguł. Ocena zależy od dwóch czynników: od tego, czy dana reguła dopasowała się do kolejnego przetwarzanego komunikatu i od „nagrody” jaką system otrzymał od środowiska w momencie gdy określona przez niego klasa decyzyjna pokrywała się z decyzją przewidzianą dla danego przykładu uczącego w zbiorze uczącym.

Algorytm oceniania zastosowany w systemie nosi nazwę *bucket brigade*. Algorytm ten można porównać do rynku usług informacyjnych. Reguły nabywają i sprzedają prawo do obrotu informacją. W takim układzie owe reguły tworzą „łańcuszek pośredników informacji” od detektorów (ze środowiska) do efektorów (do środowiska). Tłumaczy to nazwę algorytmu.

W algorytmie występują dwa zasadnicze elementy. Są to instytucja przetargu i instytucja izby rozrachunkowej. Jeżeli reguła dopasuje się do nadchodzącego komunikatu, oznacza to, że mogłaby zostać uaktywniona. Jednak tak się nie dzieje: reguła nabywa jedynie prawo do udziału w przetargu. Z każdą regułą związana jest informacja o jej sile (nabywczej) – *strength*. Każda reguła, która została dopasowana do komunikatu, składa swoją ofertę (*bid*), proporcjonalną do swojej siły. Reguły dobrze dostosowane (mające dużą siłę) składają odpowiednio wyższą ofertę. W instytucji przetargu wyłania się regułę (reguły), która będzie mogła wysłać swój komunikat (wykonać akcję, podać decyzję, zaklasyfikować – zależnie od interpretacji komunikatu).

Po wybraniu zwycięskiej reguły (reguł) musi nastąpić uregulowanie płatności w izbie skarbowej. Każda zwycięska reguła wpłaca do izby wielkość swojej oferty (*bid*). Odpowiednio maleje jej siła. Izba skarbowa natomiast rozdysponowuje tę ofertę pośród wszystkich *reguł, które nadały komunikaty uaktywniające regułę zwycięską*. Zapobiega to monopolizacji zbioru reguł przez jeden ich typ, co pozwala utrzymać pewną podpopulację reguł dobrych, choć nie podobnych do siebie. Analogią tego zjawiska w przyrodzie (i w AE) jest współzycie różnych gatunków w swoich niszach ekologicznych. Takie postępowanie jest także zgodne z metodologią uczenia maszynowego, gdzie nie próbuje się szukać jednej uniwersalnej reguły, lecz ich dobrego, użytecznego zbioru.

Dla opisanej ogólnej zasady potrzebne są konkretne metody wykorzystywane w systemie klasyfikującym. Pierwszą jest wypłacanie nagrody (przychodu) ze środowiska. Ma to miejsce wówczas, gdy decyzja jaką podjęła dana reguła dla nadchodzącego komunikatu jest zgodna z rzeczywistą decyzją (podaną w zbiorze uczącym) dla tego komunikatu. Odbyna się to poprzez zwiększenie siły danego klasyfikatora. Drugim elementem upodabniającym całość

układu oceniającego do rynku są podatki. Równanie opisujące siłę S reguły w kolejnym cyklu przetwarzania $t + 1$ możemy zapisać jako funkcję jej dotychczasowej siły $S(t)$, wypłaty na rzecz innych reguł B , podatków T , oraz przychodów R :

$$S(t + 1) = S(t) + R(t) - T(t) - B(t)$$

Wielkość R jest jednym z parametrów systemu, a początkowa wartość $S(0)$ jest cechą każdej z reguł które są początkowo umieszczone w zbiorze reguł. Wyjaśnienia wymagają wielkości B (oferta) oraz T (podatek). Oferta zależy od siły:

$$B = C_{bid} \cdot S$$

C_{bid} jest parametrem systemu i określa, jaka część siły staje się ofertą (np. 10%). Bardziej skomplikowane wygląda sprawa podatku. Rolą podatków jest usuwanie z pamięci takich reguł, które są bezproduktywne. Nigdy się nie dopasowują, ale mając pewną siłę początkową cały czas istnieją w systemie. To, że nigdy się nie dopasowują do przykładów uczących może świadczyć o ich zbyteczności. Są dwa rodzaje podatków – obrotowy i stały (od istnienia). Ten pierwszy płacią tylko te reguły, które się dopasowały. Drugi podatek jest płacony przez wszystkie reguły. Jeżeli reguła płaci podatek przez cały czas, a nie ma możliwości zwiększenia swojej siły (poprzez otrzymanie części oferty zwycięskiego klasyfikatora lub nagrody od środowiska), wówczas jej siła zmaleje do 0. Podatki obrotowy i stały są pewną częścią siły reguły, S . Jaka to część, jest określone parametrem systemu.

Jeśli razem z CFS stosuje się AE, wtedy selekcja zależy od siły reguł, zatem reguły z zerową siłą nie zostaną wylosowane przy tworzeniu nowej populacji.

Chociaż widzimy wiele elementów w formule na siłę $S(t + 1)$, udowodniono, że wszystkie są niezbędne – stanowią minimalny zestaw, który zapewnia skuteczne działanie mechanizmu ustalania sił reguł.

2.8.9 Adaptation by rule discovery

In addition to credit assignment, in order to learn we need a way to introduce new classifiers to the system. Evolutionary algorithm can be used to optimize and adapt a CFS in two ways:

- Considering the classifier list as a single individual whose chromosome is obtained by concatenating the conditions and actions of all classifiers (the “Pittsburgh” (“Pitt”) approach, De Jong)
- Considering each classifier as a separate individual (the “Michigan” approach, Holland).

In the Pittsburgh approach, the fitness of each CFS is determined by observing the behavior of the system for a certain amount of time or on some test data. The EA optimizes the CFS by breeding and making *compete* different sets of classifiers. The Michigan approach requires a fitness measure for each classifier. If used with the bucket brigade algorithm, the strength of the classifier can be taken as its fitness. In this case, the EA optimizes the CFS by breeding and making *compete* and *co-operate* different classifiers.

Algorytm ewolucyjny (w podejściu Michigan)

LCS to CFS z mechanizmami ewolucyjnymi, które optymalizują zestaw reguł. Algorytm ewolucyjny stosowany w takim systemie jest dość typowy, jednak posiada pewne cechy potrzebne z punktu widzenia systemu uczącego się. Głównym zadaniem AE jest zasilanie zbioru reguł nowymi, lepszymi regułami.

Sila reguły pełni dla AE rolę oceny (*fitness*) reguły. Ocena ta jest wykorzystywana przy selekcji. Dodatkowo można stosować mechanizm „ścisku” (zob. rozdział 2.2.2) – eliminuje on z populacji reguły identyczne poprzez zastępowanie regułami nowymi istniejących już reguł im podobnych. Po dokonaniu selekcji reguł, wykonujemy na nich krzyżowanie oraz mutację. Zwykle nie zmieniamy całej populacji, a jedynie jej część. Ma to zapobiegać wymianie całego materiału genetycznego i służy utrzymaniu części reguł niezmienionych. Dzięki temu możemy zachować wysoki poziom bieżącej efektywności w czasie, gdy system uczy się sprawniejszych wzorców zachowania – nie interesuje nas rozwiązanie zupełnie nowe, a jedynie poprawa rozwiązania dotychczasowego. W systemach uczących się nie chodzi o znalezienie jednej reguły, lecz ich zbioru (podczas gdy typowy AE jest nastawiony na zbieżność do jednego, najlepszego rozwiązania). Mamy wpływ na to, jaka część populacji zostanie wymieniona: odpowiedni parametr nazywa się współczynnikiem wymiany i określa jaka część populacji pierwotnej zostanie zastąpiona nowymi chromosomami (np. 20%).

Kolejnym parametrem wpływającym na działanie AE jest okres jego wywoływania. Tradycyjnie, gdy AE jest częścią większego systemu i służy optymalizacji jakiejś funkcji, wywołujemy go w każdym cyklu działania tego systemu. W systemie uczącym, AE wywołujemy co kilka cykli działania (deterministycznie lub średnio) lub przy zajściu określonych zdarzeń (np. spadek trafności klasyfikowania). Jeden cykl działania systemu uczącego się oznacza przetworzenie jednego komunikatu nadchodzącego ze środowiska. Choć teoretycznie można by wywoływać AE w każdym kroku działania systemu uczącego, takie postępowanie nie przynosi nadzwyczajnych wyników – a dość istotnie zwiększa nakłady obliczeniowe, ponieważ AE w porównaniu do pozostałych elementów systemu uczącego bywa kosztowny obliczeniowo.

2.8.10 Podsumowanie

Powyżej opisana została klasyczna idea GBML (analogiczna w swojej prostocie do AG). W praktyce elementy systemu i całą jego architekturę dostosowuje się do konkretnego problemu. Modyfikacje i ulepszenia systemów LCS dotyczą języka przekazu (wiele symboli to duża siła wyrazu gramatyki), zasad reprezentacji (złożone reguły), operatorów genetycznych, mechanizmów nagradzania (np. integracja z algorytmami uczenia ze wzmocnieniem), itd. Zastosowania są bardzo różnorodne ze względu na uniwersalność samej idei. Przykładowo można tu wymienić eksplorację nieznanego środowiska przez robota, systemy sterowania, trudne gry (np. poker), budowę sieci semantycznych, wyuczenie się reguł postępowania w diagnostyce medycznej i leczeniu, i wiele innych.

Ogólnie, dla złożonych problemów, systemy LCS/GBML zachowują się lepiej od tradycyjnych systemów klasyfikujących i tradycyjnych algorytmów uczenia maszynowego. W ich działaniu widzimy analogie do głębokich i rekurencyjnych sieci neuronowych, jednak LCS oferują symboliczną, zatem w pewnych zastosowaniach łatwiejszą do interpretacji niż sieci neuronowe, formę wiedzy.

2.9 Inne techniki w AE

2.9.1 Wiele kryteriów

Multiple objective GA/EA (MOGA/MOEA):

- randomized weighting (utility function),
- Pareto optimality (dominance relation) and ranking. Memory (“archive”) helps build better Pareto fronts,
- dedicated selection techniques [Kun05] such as SPEA (strength Pareto EA) and NSGA (non-dominated sorted GA), and the “crowding distance” operator; <https://www.youtube.com/watch?v=Q8KvtVCTjEw>.

2.9.2 Wzbogacanie wiedzę

You get some specific problem to optimize (e.g., optimize satellite trajectory to maximize image collection (Earth coverage) per day). How do you customize the algorithm taken from some evolutionary library/toolkit/framework? [discussion]

The problem turns out to be more difficult than expected. The algorithm seems to get stuck and does not produce satisfactory results. What do you do to help? [discussion]

Incorporating knowledge into EA:

- use many fitness functions (multiple objectives) for additional guidance,
- use knowledge-rich representations and complex genotype-phenotype mappings,
- use knowledge seeding (include good solutions in the initial population),
- to evaluate individuals, compare them with a database of good/bad cases (case-based knowledge).

Wzbogacanie wiedzę jest szczególnie potrzebne w skomplikowanych problemach optymalizacji – przykładem jest *evolutionary design* czyli projektowanie ewolucyjne omawiane w rozdziale ??.

2.9.3 Sterowanie różnorodnością i techniki jakość–różnorodność

We learned how to preserve diversity by using the crowding model or the convection selection (Sect. 2.2.2), and by speciation (Sect. 2.7.3). One recent example of optimization where maintaining diversity of solutions was important was the Michigan-style LCS (Sect. 2.8).

Controlling the diversity in fitness. In the order of complexity:

- FUSS (Fitness Uniform Selection Scheme) [HL06].
- FUDS (Fitness Uniform Deletion Scheme) [LH05].
- Convection selection – already discussed.
- HFC (Hierarchical Fair Competition) [Hu+05].

A more complex diversification mechanism (albeit requiring the introduction of additional evaluation criteria) is MAP-Elites [MC15]:

- Define your fitness as usually (for example: maximize velocity of a robot).
- Define other features of each solution (for example: robot size, weight, energy consumption).
- Discretize ranges of these other features, thus creating a multi-dimensional grid with “cells”.
- During evolution, select an individual from a random cell, mutate/crossover it, evaluate its fitness and features, and put into the appropriate cell.
- In each cell keep only one solution with the best fitness found so far.

Variants:

- Keep more than one solution in each cell.
- Start with a coarse discretization of features and gradually increase the number of intervals (resolution).

Controlling the diversity in solution content.

- Niching/speciation (already discussed).
- Novelty search (i.e., niching but disregarding the fitness component).
- Two-criteria optimization of fitness and diversity (no aggregation like in niching).
- NSLC (Novelty Search with Local Competition).

2.9.4 Obsługa ograniczeń

Constraints (hard, soft): when solutions violate them,

- correct
- penalize (pressure)
- remove (prevention)

Think about advantages and disadvantages of each approach. When each approach is suitable and when it is not?

2.10 Równoległe algorytmy ewolucyjne

Parallel EA (PEA), Równoległe AE [Pod97]. Zaleta – przyspieszenie obliczeń, ponadto wpływ na zbieżność, eksplorację i eksploatację.

- I. Algorytmy scentralizowane. Ocena osobników równoległe, ale podział zadań między procesami w schemacie master-slave (por. [KU17]). LAN, topologia gwiazdy.
- II. Algorytmy o gruboziarnistej równoległości (alg. subpopulacyjne a.k.a. model wyspowy). Równoległe procesy przetwarzają autonomiczne subpopulacje. Wymiana najlepszych osobników. Duża moc komputerów, niewielka komunikacja.
 - (a) Subpopulacje są początkowo losowane z równym rozkładem z całej dziedziny.
 - (b) Subpopulacje dzielą dziedzinę.

- (c) Subpopulacje dzielą zakres wartości funkcji celu (rozpraszanie konwekcyjne z rodzaju 2.2.2).

III. Algorytmy o drobnoziarnistej równoległości (alg. komórkowe). Każdy osobnik przetwarzany przez osobny proces połączony z procesami sąsiednimi (operacje genetyczne). Duża komunikacja!

PEA można też podzielić ze względu na komunikację – synchroniczną lub asynchroniczną.

ad. II.

Migracja osobników:

- model migracji: emigracja (osobnik opuszcza swoją subpopulację) i imigracja (kopia osobnika rozsyłana)
- topologia sieci połączeń komunikacyjnych: pierścień, torus, krata, hipersześcian, drabina... (zależnie od architektury sprzętowej)
- zasięg migracji
 - model globalny (*island model*): osobniki mogą migrować do dowolnej subpopulacji
 - model lokalny (*stepping stone model*): osobniki mogą migrować tylko do sąsiednich subpopulacji (wymaga zdefiniowania sąsiedztwa)

ad. III.

Maszyny o masowej równoległości. Każdy osobnik przetwarzany przez osobny proces, ewaluacja nie wymaga komunikacji, ale selekcja i rekombinacja – tak. Dlatego te dwie ostatnie operacje przeprowadza się na pewnym otoczeniu osobnika (by zmniejszyć nakłady na komunikację).

Topologia sieci ma duży wpływ na dynamikę i zbieżność. Najpopularniejsze topologie: krata, torus, hipersześcian, drabina.

Zaleta PEA (II i III) – utrzymanie większej różnorodności osobników, niż GA z pojedynczą populacją. Szybciej znajduje optima, nie dochodzi do przedwczesnej zbieżności całej populacji.

Problemy i kierunki badań:

ad. II.

- Lepsze wykorzystanie właściwości, poznanie, zrozumienie szczególnych cech tego typu PEA.
- Wyznaczenie optymalnych wartości parametrów opisujących architekturę tego typu PEA – np. na ile subpopulacji dzielić, jaka część osobników ma być wymieniana pomiędzy subpopulacjami itp.

ad. III.

- Opracowanie operatorów które nie wymagają tak dużych nakładów na komunikację.
- Przeciwdziałanie osłabieniu presji selekcyjnej, spowodowanej dużą decentralizacją.

2.11 Jak zrobić skuteczny AE? Na co zwrócić uwagę?

- Dobre kodowanie i operatory (wiedza dziedzinowa) wynikające z identyfikacji cech rozwiązania ujawniających globalną wypukłość.
Jeśli AE jest mocno dopasowany do problemu, nazywa się go silnie typowanym (*strongly typed*).
- Funkcja oceny – łagodne, ciągłe zmiany, dobrze dobrana. Lepsza „średnia odległość od źródła” niż „czy znaleziono źródło”. Należy uwzględnić spodziewany krajobraz ewolucyjny, myśleć kategoriami algorytmu. Mała zmiana w genotypie powinna powodować małą zmianę w fenotypie (małą zmianę wartości funkcji celu).
- Dobrze zdefiniowane środowisko, nie za trudne na początek – przykład: robot, symulacje, ...; możliwość koewolucji rozwiązania i środowiska.
- Populacja początkowa: czasem opłaca się zacząć od dobrych rozwiązań. Podczas działania warto rozważyć wykorzystanie prostej, lokalnej optymalizacji (wtedy powstanie „*genetic local search*” nazywany też czasem algorytmem memetycznym, por. rozdział ??).
- Dokładna ocena i analiza działania algorytmu: musimy wiedzieć co się dzieje, czy wystarczyło czasu na działanie optymalizacji, czy nie za silna zbieżność, czy nie marnujemy czasu, itd. Powinniśmy obserwować to na odpowiednich wykresach, w przeciwnym razie uruchomienie „czarnej skrzynki AE” nie ma sensu – z powodu źle dobranego pojedynczego parametru algorytmu możemy uzyskać beznadziejne wyniki nawet po długim czasie (podobnie jest z każdym sparаметryzowanym algorytmem optymalizacji). Właściwe, świadome wykorzystanie algorytmu jest szczególnie ważne gdy nie

znamy optimum (czyli w większości praktycznych sytuacji) – wtedy widzimy że „algorytm działa” i odkrywa coraz lepsze rozwiązania, ale nie mamy pojęcia na ile lepiej mógłby działać.

- Niedeterminizm oceny może mieć wiele źródeł. Rozważ następujące sytuacje, kiedy optymalizujemy:
 1. Model UM oceniany C-V: bardzo czasochłonna ocena, naturalny niedeterminizm, nie da się go usunąć (chyba, że olbrzymim kosztem obliczeniowym – wszystkie możliwe podziały C-V).
 2. Robota/konstrukcję 3D w symulacji: bardzo czasochłonna ocena, determinizm wyidealizowanej symulacji – wiemy, że on nie odpowiada rzeczywistości.
 3. Bota do [gry wieloosobowej](#). Jak dla przykładu robota powyżej – można łatwo zapewnić determinizm przeciwników, ale co to spowoduje...
 4. Trasę TSP: niedeterminizm czasów przejazdu w rzeczywistości, ale jeśli mamy średnie czasy w macierzy, to ocena permutacji jest błyskawiczna.

A jeśli nie mamy średnich czasów, tylko wyidealizowany symulator jeżdżącego agenta?

Kiedy w rzeczywistym środowisku występuje niepewność, niedeterminizm lub bardzo/nieskończenie wiele stanów wpływających na ocenę rozwiązania: podczas jego oceniania dodaje się w symulowanym środowisku losowy szum³¹ – w ten sposób rozwiązania stają się odporniejsze (*robust*), choć ich optymalizacja jest trudniejsza, bo ocena nie jest już deterministyczna. Takie podejście często stosuje się przy optymalizacji robotów i w projektowaniu ewolucyjnym.

- Jeśli ocena jest niedeterministyczna, to pomimo tego, że AE (bez elitaryzmu) są stosunkowo odporne na jej niedokładności („niepewność”), żeby wartość oceny była stabilniejsza stosuje się uśrednianie – wielokrotną ewaluację każdego rozwiązania. Selekcja promuje niestety „szczęściarzy” i wielokrotne ocenianie każdego osobnika, radykalnie zwiększając koszt obliczeniowy, jedynie zmniejsza skalę tego problemu. Ilukrotnie przy 100-krotnym powtórzeniu oceny osobnika? Przypomnienie: [KR01, rys. 12] oraz metody starzenia i klonowania z rozdziału 2.7.3.

³¹Nawiązując do dyskusji o roli i rodzajach losowości z rozdziału 2.2.2, rozważ i porównaj sytuacje: stałe kombinacje wartości parametrów oceny (regularna siatka), stałe kombinacje wartości parametrów oceny (nieregularne, chaotyczne), niedeterministyczne kombinacje (zmieniające się przy każdej ocenie).

2.12 Architektury koewolucyjne

Koewolucja: wiele gatunków (grup organizmów) – co najmniej dwie – wpływa nawzajem na swoje procesy ewolucyjne.

2.12.1 Kooperatywne

Osobna prezentacja bazująca na publikacji [PD00], por. https://deap.readthedocs.io/en/master/examples/coev_coop.html.

2.12.2 Konkurencyjne

Typowy przykład: koewolucja (optymalizacja) strategii [Elf+21]. Osobnik reprezentuje wiedzę zawartą w strategii (np. mogą być to wagi kryteriów używanych do oceny sytuacji na planszy w grze). Ocenę osobnika uzyskuje się np. przez rozegranie wielu partii z pozostałymi osobnikami (każdy gra według własnej strategii).

Dyskusja: czy włączając taki proces, jeśli odpowiednio długo poczekamy, otrzymamy mistrzowską strategię? Jeśli nie, to dlaczego? (podaj przyczyny – listę ewentualnych problemów).

Trudności: chcemy *arms race* (ciągła konkurencja), lecz możemy zostać w *MSS* – *Mediocre Stable State* (stagnacji – kiepskim, stabilnym stanie). Zbyt silny przeciwnik nie pozwoli różnić przeciętnego i złego rozwiązania; zbyt słaby – przeciętnego i dobrego. Ocena każdego zależy od innych (zewnętrzny, obiektywny „nauczyciel” rozwiązuje ten problem równocześnie eliminując zalety koewolucji). Ocena strategii może być nieprzechodnia.

- Dyskusja przykładowych sytuacji: GP (populacja wyrażeń i populacja testów), strategię gry w szachy, piłka nożna, tenis i nieprzechodniość „lepszości”, papier–kamień–nożyce, lokalna szkoła szermierki i różnorodność (por. *exploiter agents* w [AlphaStar](#)), natura.
- Pojęcia: *arms race*, Red Queen³², MSS.
- Problemy: brak lub utrata gradientu, zapętlenie (cykle, nieprzechodniość relacji porównania wynikającej z oceniania), brak monotoniczności (postępu) [Mic09].
- Rady: *competitive fitness sharing* (zwiększanie oceny tych rozwiązań, które wygrywają z testami (przeciwnikami) trudnymi dla pozostałych rozwiązań [RB95]), specjalny

³²https://en.wikipedia.org/wiki/Red_Queen_hypothesis

dobór zbioru testów, utrzymywanie *hall of fame* lub zbiorów Pareto-niezdominowanych rozwiązań i testów.

Bibliografia

- [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](https://doi.org/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](https://doi.org/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](https://doi.org/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.
- [CG97] Andrzej Chorażyczewski i Roman Galar. “Wizualizacja dynamiki ewolucji fenotypowej”. W: *Proceedings of the 2nd National Conference on Evolutionary Computation and Global Optimization*. Ryto: Oficyna Wydawnicza Politechniki Warszawskiej, wrz. 1997, s. 41–48. URL: <https://troja.uksw.edu.pl/pdf/kaeiog/KA Ei0G1997.41-48.pdf>.
- [Dav90] Yuval Davidor. “Epistasis variance: Suitability of a representation to genetic algorithms”. W: *Complex Systems* 4.4 (1990), s. 369–383. URL: <http://www.complex-systems.com/pdf/04-4-1.pdf>.
- [DJ75] Kenneth Alan De Jong. “Analysis of the behavior of a class of genetic adaptive systems”. Prac. dokt. University of Michigan, 1975. URL: <https://deepblue.lib.umich.edu/bitstream/handle/2027.42/4507/bab6360.0001.001.pdf>.
- [ECS89] Larry J. Eshelman, Rich Caruana i J. David Schaffer. “Biases in the crossover landscape”. W: *Proceedings of the 3rd International Conference on Genetic Algorithms*. Red. J. David Schaffer. Morgan Kaufmann, 1989, s. 10–19.
- [Elf+21] Ehab Z. Elfeky i in. “A systematic review of coevolution in real-time strategy games”. W: *IEEE Access* 9 (2021), s. 136647–136665. URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9548932>.

- [ENS99] Ewa Niewiadomska-Szynkiewicz. “Przegląd metod optymalizacji globalnej”. W: *Proceedings of the 3rd National Conference on Evolutionary Computation and Global Optimization*. Potok Żłoty: Oficyna Wydawnicza Politechniki Warszawskiej, 1999, s. 363–371. URL: <https://troja.uksw.edu.pl/pdf/kaeiog/KAeiOG1999.363-371.pdf>.
- [Fog00] David B. Fogel. *Evolutionary computation: toward a new philosophy of machine intelligence*. 2 wyd. IEEE Press, 2000.
- [Fra72] Daniel Raymond Frantz. “Non-linearities in genetic adaptive search”. Prac. dokt. University of Michigan, 1972. URL: <https://deepblue.lib.umich.edu/bitstream/handle/2027.42/4950/bac2356.0001.001.pdf>.
- [Gea+02] Nicholas Geard i in. “A comparison of neutral landscapes – NK, NKp and NKq”. W: *Proceedings of the 2002 Congress on Evolutionary Computation, CEC’02*. T. 1. IEEE. 2002, s. 205–210. URL: <https://eprints.soton.ac.uk/264208/1/cec1-comp.pdf>.
- [Gol+93] David E. Goldberg i in. *Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms*. Spraw. tech. 93004. 1993, s. 1–16. URL: <http://repository.ias.ac.in/81677/1/110-a.pdf>.
- [Gol03] David Edward Goldberg. *Algorytmy genetyczne i ich zastosowania*. Wydawnictwa Naukowo-Techniczne, 2003.
- [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>.
- [Gut94] Marek W. Gutowski. “Smooth genetic algorithm”. W: *Journal of Physics A: Mathematical and General* 27.23 (1994), s. 7893.
- [Gut97] Marek Gutowski. “Zdolność rozdzielcza gładkiego algorytmu ewolucyjnego”. W: *Proceedings of the 2nd National Conference on Evolutionary Computation and Global Optimization*. Ryto: Oficyna Wydawnicza Politechniki Warszawskiej, wrz. 1997, s. 73–79.
- [Gwi07a] Tomasz Dominik Gwiazda. *Algorytmy genetyczne – kompendium. Tom I. Operator krzyżowania dla problemów numerycznych*. PWN, 2007.
- [Gwi07b] Tomasz Dominik Gwiazda. *Algorytmy genetyczne – kompendium. Tom II. Operator mutacji dla problemów numerycznych*. PWN, 2007.
- [HL06] Marcus Hutter i Shane Legg. “Fitness uniform optimization”. W: *IEEE Transactions on Evolutionary Computation* 10.5 (2006), s. 568–589. URL: <https://arxiv.org/pdf/cs/0610126.pdf>.
- [HR78] John H. Holland i Judith S. Reitman. “Cognitive systems based on adaptive algorithms”. W: *Pattern-directed inference systems*. Elsevier, 1978, s. 313–329.
- [Hu+05] Jianjun Hu i in. “The Hierarchical Fair Competition (HFC) framework for sustainable evolutionary algorithms”. W: *Evolutionary Computation* 13.2 (2005), s. 241–277. URL: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=9a9418ae55bdf2f1a988effe855>
- [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>.

- [KM18] Maciej Komosinski i Konrad Miazga. “Comparison of the tournament-based convection selection with the island model in evolutionary algorithms”. W: *Journal of Computational Science* 32 (2018), s. 106–114. ISSN: 1877-7503. DOI: [10.1016/j.jocs.2018.10.001](https://doi.org/10.1016/j.jocs.2018.10.001). URL: <http://www.framsticks.com/files/common/ConvectionSelectionVsIslandModel.pdf>.
- [KR01] Maciej Komosinski i Adam Rotaru-Varga. “Comparison of different genotype encodings for simulated 3D agents”. W: *Artificial Life Journal* 7.4 (Fall 2001), s. 395–418. DOI: [10.1162/106454601317297022](https://doi.org/10.1162/106454601317297022). URL: <http://www.framsticks.com/files/common/ComparisonGeneticEncoding.pdf>.
- [KT20] Iuliia Kotseruba i John K. Tsotsos. “40 years of cognitive architectures: core cognitive abilities and practical applications”. W: *Artificial Intelligence Review* 53.1 (2020), s. 17–94. DOI: [10.1007/s10462-018-9646-y](https://doi.org/10.1007/s10462-018-9646-y).
- [KU17] Maciej Komosinski i Szymon Ulatowski. “Multithreaded computing in evolutionary design and in artificial life simulations”. W: *The Journal of Supercomputing* 73.5 (2017), s. 2214–2228. ISSN: 1573-0484. DOI: [10.1007/s11227-016-1923-4](https://doi.org/10.1007/s11227-016-1923-4). URL: <http://www.framsticks.com/files/common/MultithreadedEvolutionaryDesign.pdf>.
- [Kub04] Marek Kubiak. “Systematic construction of recombination operators for the vehicle routing problem”. W: *Foundations of Computing and Decision Sciences* 29.3 (2004). URL: http://www.cs.put.poznan.pl/mkubiak/publications/Kubiak_SystematicConstruction_FCDS2004.pdf.
- [Kun05] Daniel Kunkle. *A summary and comparison of MOEA algorithms*. Spraw. tech. 2005. URL: <https://www.ccs.neu.edu/home/kunkle/papers/techreports/moeaComparison.pdf>.
- [Kwa97] Halina Kwaśnicka. “Nadmiarowość genetyczna, poligeniczność i plejotropowość w algorytmach ewolucyjnych”. W: *Proceedings of the 2nd National Conference on Evolutionary Computation and Global Optimization*. Ryto: Oficyna Wydawnicza Politechniki Warszawskiej, wrz. 1997, s. 137–144.
- [LH05] Shane Legg i Marcus Hutter. “Fitness uniform deletion: A simple way to preserve diversity”. W: *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. 2005, s. 1271–1278. URL: <https://arxiv.org/pdf/cs/0504035.pdf>.
- [Mah92] Samir W. Mahfoud. “Crowding and preselection revisited”. W: *Parallel problem solving from nature*. Red. R. Männer i B. Manderick. T. 2. Elsevier, 1992, s. 27–36. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.3943&rep=rep1&type=pdf>.
- [MC15] Jean-Baptiste Mouret i Jeff Clune. “Illuminating search spaces by mapping elites”. W: *arXiv preprint arXiv:1504.04909* (2015). URL: <https://arxiv.org/pdf/1504.04909.pdf>.
- [Mic09] Thomas Miconi. “Why coevolution doesn’t “work”: superiority and progress in coevolution”. W: *12th European Conference on Genetic Programming – EuroGP*. Springer, 2009, s. 49–60. URL: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=c51f7914019c020d2386b4880cdc59b22dbbd7c4>.
- [Mic96] Z. Michalewicz. *Algorytmy genetyczne + struktury danych = programy ewolucyjne*. Wydawnictwa Naukowo-Techniczne, 1996.
- [NK00] B. Naudts i L. Kallel. “A comparison of predictive measures of problem difficulty in evolutionary algorithms”. W: *IEEE Transactions on Evolutionary Computation* 4.1 (2000), s. 1–15. DOI: [10.1109/4235.843491](https://doi.org/10.1109/4235.843491).

- [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.
- [PD00] Mitchell A. Potter i Kenneth A. De Jong. “Cooperative coevolution: an architecture for evolving coadapted subcomponents”. W: *Evolutionary computation* 8.1 (mar. 2000), s. 1–29. DOI: 10.1162/106365600568086. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.5861&rep=rep1&type=pdf>.
- [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>.
- [Pod97] Mariusz Podsiadło. “Równoległe algorytmy genetyczne – przegląd problematyki”. W: *Proceedings of the 2nd National Conference on Evolutionary Computation and Global Optimization*. Rytyro: Oficyna Wydawnicza Politechniki Warszawskiej, wrz. 1997, s. 219–227.
- [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.
- [RB95] Christopher Rosin i Richard Belew. “Methods for competitive co-evolution: finding opponents worth beating”. W: *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1995, s. 373–380. URL: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.9359&rep=rep1&type=pdf>.
- [Rea+19] Esteban Real i in. “Regularized evolution for image classifier architecture search”. W: *Proceedings of the AAAI conference on artificial intelligence*. T. 33. 1. 2019, s. 4780–4789. URL: <https://ojs.aaai.org/index.php/AAAI/article/download/4405/4283>.
- [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*. Red. 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.

- [RW95] Colin R. Reeves i Christine C. Wright. “Epistasis in Genetic Algorithms: An Experimental Design Perspective”. W: *ICGA*. 1995, s. 217–224. URL: https://www.researchgate.net/profile/Christine_Wright5/publication/2504851_Epistasis_in_Genetic_Algorithms_An_Experimental_Design_Perspective/links/0912f50bf2d3d75799000000.pdf.
- [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](https://doi.org/10.1023/A:1008202821328).
- [SW15] Boris Shabash i Kay C. Wiese. “Diploidy in evolutionary algorithms for dynamic optimization problems: A best-chromosome-wins dominance mechanism”. W: *International Journal of Intelligent Computing and Cybernetics* 8.4 (2015), s. 312–329. URL: https://www.researchgate.net/profile/Kay-Wiese/publication/283648490_Diploidy_in_evolutionary_algorithms_for_dynamic_optimization_problems/links/5eea7a3d92851ce9e7ec6cce/Diploidy-in-evolutionary-algorithms-for-dynamic-optimization-problems.pdf.
- [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*. Ryto: Oficyna Wydawnicza Politechniki Warszawskiej, wrz. 1997, s. 293–297.
- [Wei09] Thomas Weise. *Global Optimization Algorithms – Theory and Application*. Second. Self-published, 2009. URL: <http://www.it-weise.de/projects/book.pdf>.
- [Yin+19] Chris Ying i in. “NAS-Bench-101: Towards reproducible neural architecture search”. W: *International Conference on Machine Learning*. PMLR. 2019, s. 7105–7114. URL: <https://proceedings.mlr.press/v97/ying19a/ying19a.pdf>.

Cytowanie tego skryptu:

```
@booklet{MK-AlgEwolSkrypt,
  title = {Algorytmy ewolucyjne},
  author = {Maciej Komosiński},
```

```
year    = {2025},  
note    = {Skrypt do zajęć},  
url     = {http://www.cs.put.poznan.pl/mkomosinski/lectures/optimization/MK\_AlgEwolucyjne.pdf}  
}
```