ALGORYTMY EWOLUCYJNE
NIEKLASYCZNE KODOWANIE I MUTACJA*)

- Algorytmy ewolucyjne zaliczane są do grupy tzw. metaheurystyk, czyli uniwersalnych metod rozwiązywania szerokiej klasy problemów.
- Ceną za ich uniwersalność są często gorsze wyniki niż te uzyskiwane przez specjalizowane algorytmy zaprojektowane do określonego zadania, np. sortowanie łatwiej wykonać specjalizowanym algorytmem niż algorytmem genetycznym (choć to ostatnie też jest możliwe).

*) Źródła: *1. J. Cytowski, Algorytmy genetyczne, Podstawy i zastosowanie, Akademicka Oficyna Wydawnicza, Warszawa, 1996*

2. T. Białaszewski, Programowanie Genetyczne, materiały do wykładu, Internet, 2007

Lekarstwem na - szkodliwą z punktu widzenia skuteczności - nadmierną uniwersalność algorytmów ewolucyjnych może być:

- Stosowanie specjalizowanych, zaprojektowanych do konkretnego zadania optymalizacyjnego, sposobów kodowania. Zamiast stosować klasyczny algorytm ewolucyjny z kodowaniem binarnym lub zmiennoprzecinkowym, w którym część osobników może nie spełniać warunków (więzów) narzucanych na rozwiązania, należy tak dobrać kodowanie, by odtwarzało ono naturalną "strukturę" (rozumianą w kategoriach funkcji celu *f* i operatorów genetycznych) przestrzeni rozwiązań.

- Inną metodą może być zastosowanie algorytmu hybrydowego. . W tym celu należy znaleźć specjalizowany deterministyczny algorytm heurystyczny znajdujący w krótkim czasie jeden wynik, zwykle nieoptymalny, ale za to uzyskany szybko przy wykorzystaniu wiedzy o naturze problemu, który należy powiązać z niedeterministycznym algorytmem ewolucyjnym, działającym wolniej, przeszukującym przestrzeń rozwiązań i zawężającym ją do mniejszej sub-przestrzeni, w której można oczekiwać poszukiwanego rozwiązania optymalnego.

Wadą algorytmów hybrydowych jest ich zwykle dłuższy czas działania. W jednym kroku należy nie tylko przeprowadzić selekcję, uruchomić mutację i krzyżowanie, ale też wielokrotnie uruchamiać dodatkowy algorytm heurystyczny, by policzyć funkcję dopasowania.

Omawiane dotychczas algorytmy genetyczne i ewolucyjne są najczęściej wykorzystywane z uwagi na ich prostotę i stosowalność od dłuższego już czasu (przyzwyczajenie użytkowników do tego co jest już od dawna z powodzeniem stosowane).

Z drugiej jednak strony, jak wykazują badania, większość praktycznych zastosowań algorytmów ewolucyjnych opiera się na innych niż klasyczne sposobach kodowania:

- Przykładowo, jeżeli mamy zoptymalizować funkcję *f(x,y)* zdefiniowaną w domenie będącą pewnym kole, możemy albo dopuszczać *(x,y)* pochodzące z kwadratu zawierającego to koło i odrzucać osobniki, które wyjdą poza koło, albo zmienić parametryzację kartezjańską na biegunową *(x,y) -> (a,r)*.

- W przypadku problemów kombinatorycznych wygodniej jest przechowywać w chromosomie np. permutacje zmiennych projektowania, drzewa czy nawet bardziej złożone struktury, przystosowywując odpowiednio algorytmy krzyżowania i mutacji.

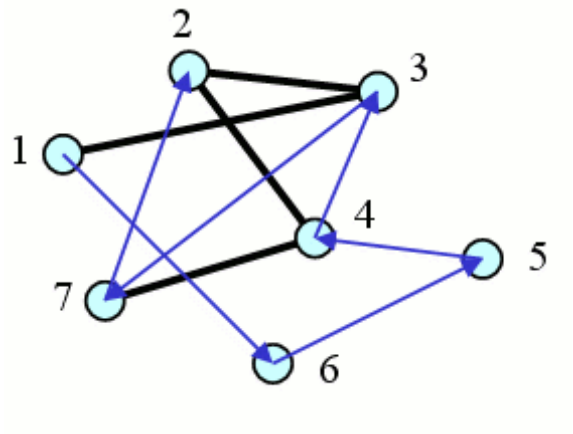
- Krzyżowanie i mutacja muszą prowadzić do powstawania prawidłowo zbudowanych osobników, które da się rozkodować i zinterpretować jako potencjalne rozwiązania problemu.

- Operator mutacji powinien pozwalać na przekształcenie dowolnego osobnika w dowolnego innego (być może w kilku krokach), co może umożliwić szybsze i łatwiejsze osiągnięcie optymalnego rozwiązania globalnego a nie jednego z możliwych rozwiązań optymalnych lokalnie.

Kodowanie permutacji

W pewnych zastosowaniach kodowanie binarne jest mniej naturalne niż inne sposoby kodowania. Na przykład w problemie komiwojażera najwygodniej jest trasy przedstawiać jako permutacje miast, które ma odwiedzić komiwojażer, wracając na końcu do punktu (miasta) startu. Poszukiwane rozwiązanie optymalne ma wyznaczyć kolejność odwiedzanych miast tak, aby przebyta łączna droga osiągnęła swoje minimum.

- Permutacje miast (czyli ich kolejność odwiedzania) są, bez dodatkowego przekodowania, zapisane jako np. liczby całkowite, w chromosomach populacji. Alternatywne rozwiązanie, w którym te permutacje kodujemy łańcuchami binarnymi, jest trudniejsze i kosztowniejsze.



Przykładowe chromosomy mogą mieć np. postać:

[1 6 5 4 3 7 2] i [1 2 4 7 6 5 3].

W przypadku typowego krzyżowania jednopunktowego powstały-by np. chromosomy potomne o postaci:

$$[1\ 6\ 5\ 4\ 3\ 7\ 2]\ \text{i}\ [1\ 2\ 4\ 7\ 6\ 5\ 3] \Rightarrow [1\ 6\ 5\ 4\ 6\ 5\ 3]\ \text{i}\ [1\ 2\ 4\ 7\ 3\ 7\ 2],$$

które nie są prawidłowymi permutacjami ciągu $1 \div 7$.

Widać zatem, że w takim problemie należy operatory genetyczne mutacji i krzyżowania zaprojektować tak, aby zawsze w wyniku ich działania dawały poprawną permutację.

- W literaturze opisano szereg metod krzyżowania, zachowujących własność poprawnej permutacji w chromosomach pochodnych. Za przykład niech posłuży opracowany w 1985 r. operator krzyżowania PMX (*partially-mapped crossover)*:

  Dla dwóch chromosomów $v_1 = [i_1, i_2, .., i_n]$ i $v_2 = [j_1, j_2, .., j_n]$ wyznaczane są segmenty określone dwoma parametrami *k* i *m*, takimi, że:

  $$v_1 = [i_1, i_2, .., i_k, i_{k+1}, .., i_{k+m}, .., i_n]\ \text{i}\ v_2 = [j_1, j_2, .., j_k, j_{k+1}, .., j_{k+m}, .., j_n],$$

W obu chromosomach wykonuje się przekształcenie $i_{k+l} \leftrightarrow j_{k+l}$ dla $l = 0,1,..,m,$
w wyniku czego powstają dwa chromosomy potomne o postaci:

$$w_1 = [i_{r_1}, i_{r_2}, .., i_{r_{k-1}}, j_k, j_{k+1}, .., j_{k+m}, .., i_{r_{n-m-1}}]$$
$$w_2 = [j_{p_1}, j_{p_2}, .., j_{p_{k-1}}, i_k, i_{k+m}, , .., j_{p_{n-m-1}}]$$

W chromosomie w_1 geny i_{r_l} o wartościach różnych od $j_{k+1}, , .., j_{k+m}$ pokrywają się z genami i_l, natomiast wartości pozostałych genów wynikają z przekształceń. Podobnie określane są geny j_{p_l} chromosomu w_2.

- Mutacja w sytuacji kodowania permutacyjnego polega, na przykład, na wykonaniu losowej transpozycji genów, czyli wybraniu losowej pary numerów odwiedzanych miast i ich zamianie miejscami:

[1 6 5 4 3 7 2] $\Rightarrow$ [1 6 7 4 3 5 2]

Kodowanie strategii postępowania w algorytmach ewolucyjnych

Wspomniany algorytm hybrydowy stanowił pewien przełom metodologiczny, w porównaniu z klasycznymi algorytmami genetycznym lub ewolucyjnym. O ile te ostatnie po prostu szukały rozwiązania problemu, o tyle metody hybrydowe szukają raczej *przepisu* (algorytmu) uzyskania rozwiązania analizowanego problemu lub klasy problemów.

W podobnym kierunku idą zastosowania związane z projektowaniem strategii, w której kod genetyczny to po prostu zapis pewnego algorytmu (programu) w prostym języku. Strategia może być, na przykład, rozumiana jako przepis (algorytm) mówiący, jak rozwiązywać problem, w którym na wejściu dostajemy historię poprzednich podobnych rozwiązań, a na wyjściu otrzymujemy odpowiedź jak rozwiązać problem aktualnie analizowany.

Takie podejście do algorytmów genetycznych i ewolucyjnych prowadzi do tzw. *Programowania Genetycznego* (*Genetic Programming*).

‚PROGRAMOWANIE GENETYCZNE

Oprócz prostych ciągów binarnych i bardziej skomplikowanych struktur kombinatorycznych, zawartość chromosomu mogą stanowić programy komputerowe.

Oczywiście wiele jeszcze nam brakuje do sytuacji, w której nowe oprogramowanie konsumenckie będzie „hodowane”, a nie pisane, jednakże w pewnych najprostszych sytuacjach ewolucyjne (samoistne) tworzenie programów jest realne.

Warunkiem jest odpowiednie ograniczenie składni i wielkości programu, a także możliwość eksperymentalnego zbadania jego skuteczności.

Istnieją problemy, w których rozwiązania nie można uzależnić od wektora zmiennych decyzyjnych. Naturalnym określeniem ich rozwiązania jest podanie strategii lub programu.

J. Koza [Koza 1992, *Genetic Programming: On the programming of computers by means of natural selection*, MIT Press, 1992] zaproponował schemat włączenia algorytmów genetycznych do modyfikacji strategii lub programów. Taki sposób rozwiązywania problemów nazywany jest programowaniem genetycznym.

Definicja Programowania Genetycznego, sformułowana w Wikipedii, brzmi następująco:

Programowanie genetyczne, GP – zautomatyzowana metoda mająca na celu tworzenie programów komputerowych w oparciu o ogólną definicję problemu. Innymi słowy programowanie genetyczne pozwala, w oparciu o wysokopoziomową definicję mówiącą co ma być zrobione, automatycznie stworzyć program, który owo zagadnienie rozwiąże.

Jako język zapisu programów służących do tego celu, twórca *programowania genetycznego* J. Koza, przyjął język programowania LISP (*LISt Proccesing*).

Kilka uwag o języku programowania LISP:

1. Jeden z pierwszych języków wspierających programowanie symboliczne - czyli operacje na abstrakcyjnych obiektach i strukturach danych zamiast zmiennych liczbowych i tablic z wartościami numerycznymi. Za pomocą list i symboli można było budować drzewa i inne skomplikowane struktury.

2. Lisp jest drugim z kolei pod względem wieku językiem programowania wysokiego poziomu pozostającym w użyciu (starszy jest tylko Fortran).

3. Podobnie jak Fortran, Lisp ulegał na przestrzeni czasu licznym zmianom. Powstało również wiele jego dialektów. Dziś do najpopularniejszych należą trzy: *Common Lisp*, *Scheme* i *Clojure*.

4. Lisp powstał jako wygodna matematyczna notacja dla programów komputerowych. Szybko został najchętniej wybieranym językiem do badania i rozwoju sztucznej inteligencji. Wywodzi się z niego wiele technik programistycznych, takich jak struktury drzewiaste, odśmiecanie pamięci, dynamiczne typowanie czy nowe koncepcje w programowaniu obiektowym (*Common Lisp Object System*).

5. Nazwa Lisp pochodzi od LISt Processing. Podstawową strukturą danych w Lispie jest lista; kod źródłowy programów w Lispie składa się z list. Dzięki temu język jest homoikoniczny (*zaletą homoikoniczności jest prostsze rozszerzenie języka o nowe koncepcje programistyczne)*, tzn. programy w Lispie mogą manipulować kodem źródłowym jak zwykłą strukturą danych. Umożliwia to pisanie makr, pozwalających programiście tworzyć nową składnię lub nawet małe, zagnieżdżone w Lispie, języki DSL(*domain-specific language*), *DSL* – język programowania przystosowany do rozwiązywania określonej dziedziny problemów, określonej reprezentacji problemu lub określonej techniki ich rozwiązywania).

Podstawowe konstrukcje języka LISP:

Niech F będzie zbiorem symboli funkcyjnych, T zbiorem stałych i zmiennych. Podstawowym pojęciem LISP-u jest lista określona przez następujące warunki:

1. t jest listą dla dowolnego $t \in T$.
2. Jeżeli $f \in F$ jest symbolem funkcyjnym dla funkcji n argumentowej oraz $l_1, \ldots, l_n$ są listami to $(fl_1, \ldots, l_n)$ jest także listą.

Z każdą listą l można związać funkcję f w sposób następujący:

1. Jeżeli $l = t$ dla $t \in T$ to:
   (a) f jest funkcją stałą, gdy t jest stałą;
   (b) f jest funkcją identycznościową zmiennej t, gdy t jest zmienną.
2. Jeżeli f jest postaci: $l = (fl_1, \ldots, l_n)$

oraz listy l_i (dla $i = 1,..., n$) są związane z funkcjami: $g_i(x_1^{(i)}, \ldots, x_k^{(i)})$ dla $i = 1,..., n$, to z listą l związana jest następująca funkcja:

$$f(g_i(x_1^{(i)}, \ldots, x_{k_1}^{(i)}), \ldots, g_n(x_1^{(n)}, \ldots, x_{k_n}^{(n)}))$$

Założymy dla. przykładu, że zbiory F oraz T są następujące:

$$F = \{+, \cdot, -, /, \sin\},$$

$$\mathrm{T} = \{x,\ y,\ 50.0,\ 5.0,\ 10.0\ ,1.5\ ,\pi,\ 2000.0,\ 64.0,\ 16.0,\ 0.185\}.$$

Przyjmiemy też , że $+, \cdot, -, /$ są dwuargumentowe, a funkcja $\sin$jest jednoargumentowa. Wtedy następująca lista:

$$(-(-(50.0)(\cdot(5.0)(\sin(10.0)(\pi x))))(\cdot(10.0)(\cdot(-x)(1.5))(-x(1.5)))) \qquad (*)$$

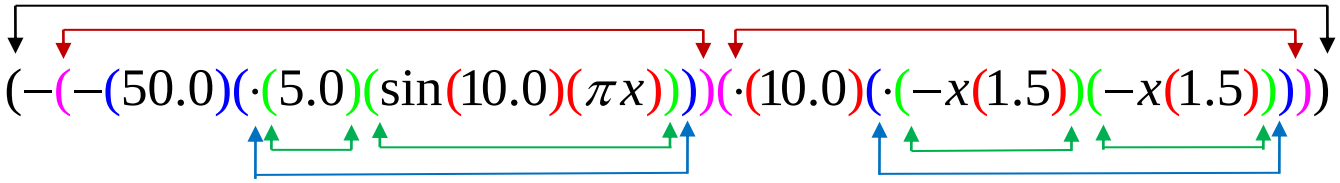
jest tożsama z funkcją o postaci:

$$f(x) = \ 50.0\ -\ 5.0 \cdot \sin(10.0 \cdot \pi \cdot x)\ -10.0 \cdot (\mathrm{x}\ -1.5)^2.$$

Oryginalna reprezentacja w postaci listy symboli funkcyjnych F, zbioru T stałych i zmiennych oraz nawiasów jest bardzo użyteczna do zapisów postaci funkcji w kodzie źródłowym języka LISP.

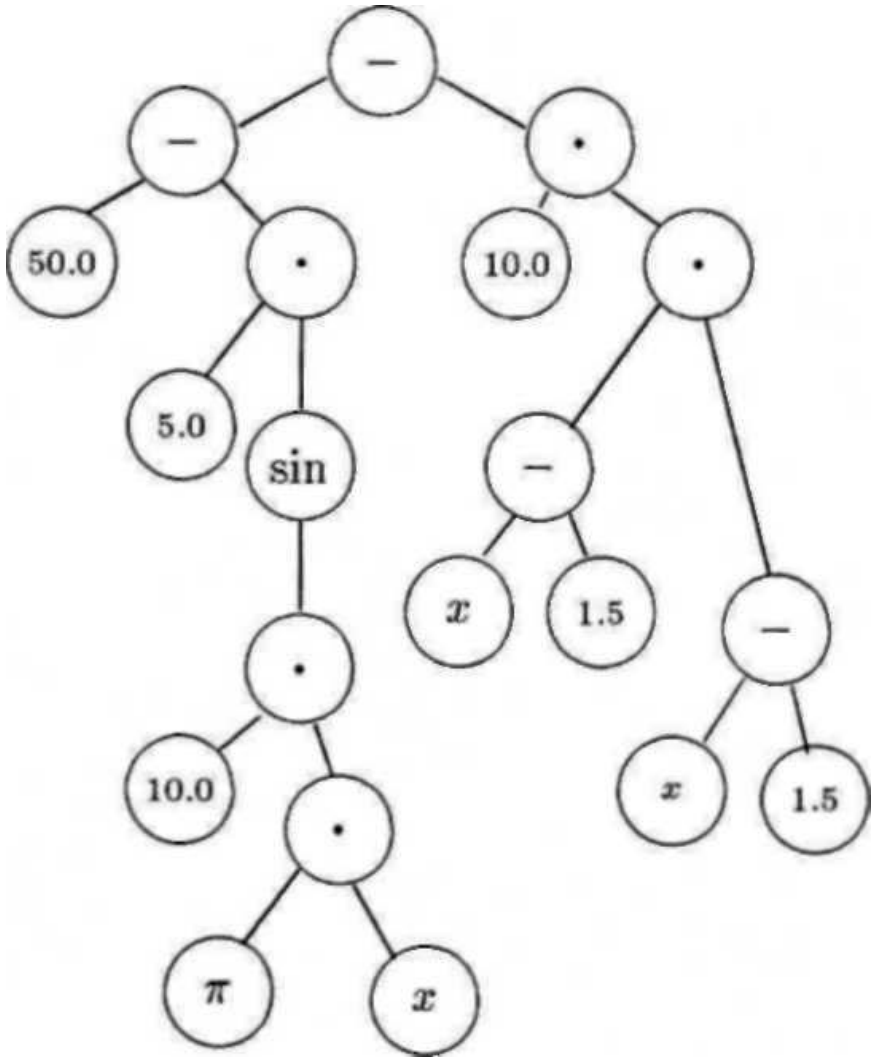
Istotną cechą takiej listy jest sposób zapisu działań arytmetycznych w postaci *operator(argumenty operatora)* oraz działania wykonywane z pierwszeństwem działań w nawiasach.

Spójrzmy raz jeszcze na listę (*)



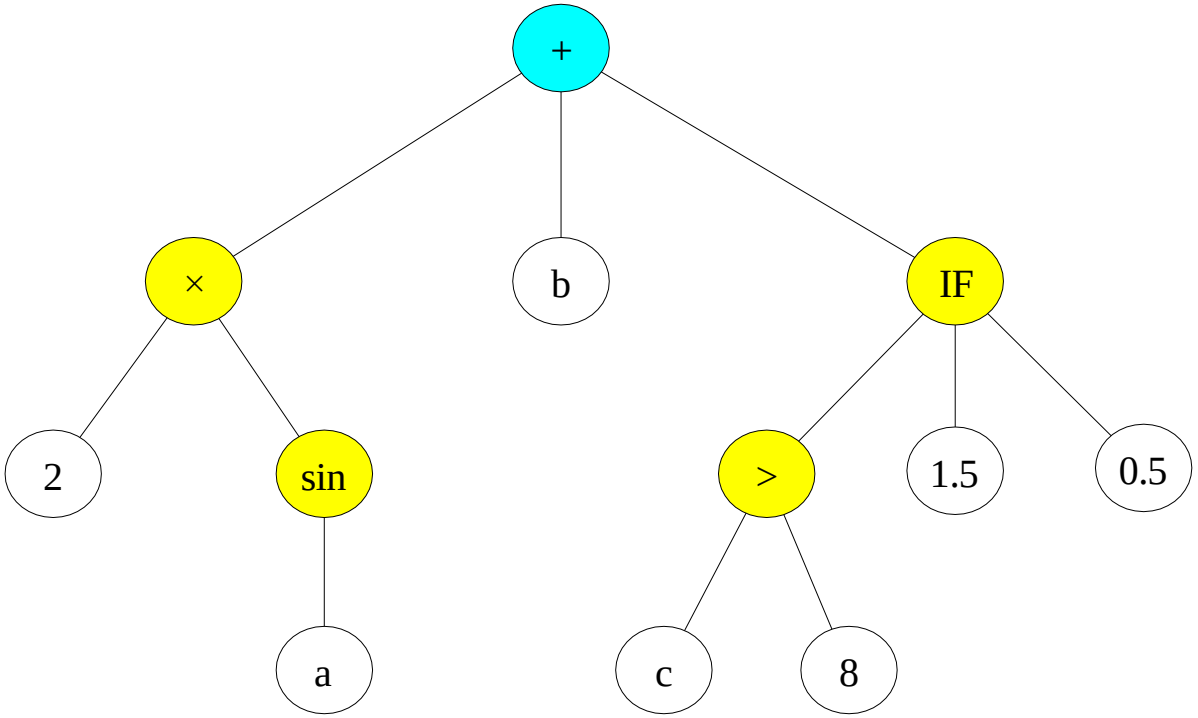
$$f(x) = 50.0 - 5.0 \cdot \sin(10.0 \cdot \pi \cdot x) - 10.0 \cdot (x - 1.5)^2$$

Bardziej przejrzystą reprezentację listy można uzyskać w postaci drzewa. Węzłami takiego drzewa są stałe, zmienne lub symbole funkcyjne. Węzły symboli funkcyjnych są połączone z węzłami ich argumentów. Na następnej stronie przedstawione jest drzewo dla listy odpowiadającej opisanej powyżej funkcji *f(x)*.



$$f(x) = 50.0 - 5.0 \cdot \sin(10.0 \cdot \pi \cdot x) - 10.0 \cdot (x - 1.5)^2$$

Jako drugi przykład rozpatrzmy drzewo następujące:



Reprezentuje ono następującą funkcję: $f(x) = (2 \cdot (\sin a) + b + x$ gdzie x=1.5 gdy c>8 w przeciwnym razie x=0.5. Odpowiednia lista będzie miała postać:

$$(+(+(+(\cdot(2\ \sin(a))(b))x)x)(\cdot(\text{if}(>((c)(8))(1.5)(0.5)))$$

- Zatem każda lista zawierająca symbole funkcyjne, zmienne i stałe może być traktowana jako **chromosom** o szczególnej strukturze, tzn. strukturze drzewiastej zawierającej węzły połączone krawędziami. W takim drzewie mogą występować:
  - **węzły terminalne**, tzn. nie zawierające węzłów podrzędnych, tzw. **liście**;
  - **węzły pośrednie**, tzn. nie będące węzłami terminalnymi;
  - dokładnie jeden **węzeł nadrzędny**, tzw. **korzeń drzewa**;
  - **krawędzie** łączące węzły i opisujące relacje między nimi.

- Oznacza to, że w chromosomie jest zapisana w postaci listy postać pewnej funkcji zawierającej informację o „stopniu rozwiązania" analizowanego problemu przez daną listę.

- Zbór tak zdefiniowanych chromosomów tworzy populację funkcji, które mogą być rozważane jako potencjalne rozwiązania (dobre lub złe lub pośrednie w tym samym sensie)

- , Populacja chromozomów, a więc list różnych postaci funkcji, podlega, podobnie jak w typowym algorytmie genetycznym lub ewolucyjnym, operacjom selekcji, mutacji i krzyżowania oraz wyznaczeniu populacji początkowej

- Zasadniczym celem programowania genetycznego jest takie przetworzenie list, aby stworzyć listę optymalną lub prawie optymalną, która opisuje funkcję przystosowania rozwiązującą „prawie najlepiej” zbiór pewnej klasy problemów, dla których nie potrafimy *a priori* sformułować poprawnej postaci funkcji przystosowania, lecz posiadamy np. informacje, uzyskane z rzeczywistych eksperymentów lub obserwacji, wiążące znane parametry wejściowe z parametrami wyjściowymi.

  ```W takiej sytuacji funkcją dopasowania może być błąd stanowiący różnicę pomiędzy wartościami parametrów wyjściowych uzyskanych dla chromosomu ”optymalnego” i otrzymanych w drodze realnego eksperymentu dla tych samych wartości parametrów wejściowych.
  ```

Zasadniczy schemat programowania genetycznego składa się z następujących elementów:

1. Wybór odpowiednich zbiorów F oraz T.
2. Wyznaczenie populacji początkowej.
3. Wybór funkcji dopasowania, oceniającej jak dobrze dana lista spełnia zadany problem.
4. Wybór procedury selekcji (zazwyczaj jest to standardowa, procedura **metody ruletki**, **metody turniejowej** lub **rankingowej**.
4. Zdefiniowanie operatora, krzyżowaniu
5. Wybór warunków zakończenia procesu przetwarzania list.

- Pierwszą operacją inicjującą PG jest wybór skończonych zbiorów postaci funkcji oraz zbiorów zmiennych i stałych tak aby wartości i typy danych wyznaczone przez funkcje były akceptowane przez wszystkie funkcje oraz znajdowały się w zbiorze stałych i zmiennych.

- Drugim ważnym postulatem jest to by wybór zbiorów F oraz T umożliwiał rozwiązanie postawionego problemu.
  Na przykład zbiór funkcji (węzłów pośrednich) może mieć postać:

$$F = \{+, -, *, /, \text{IF}, \text{THEN}, \text{AND}, \sin(), \text{FOR}\},$$

  zaś przykładowy zbiór T węzłów terminalnych:

$$T = \{x, y, \text{a}, \text{b}, \text{wartości stałe}\}$$

- Kolejnym problemem przy projektowaniu programu genetycznego jest wybór populacji początkowej. Losowy wybór tej pierwszej populacji musi uwzględniać specyfikę tworzenia list LISP-owych. Jeżeli na przykład ze zbioru $F \cup T$ wylosowany został symbol funkcyjny odpowiadający funkcji n argumentowej to następnych n losowań dotyczy wyboru argumentów tej funkcji. Argumenty mogą być znowu funkcjami i losowanie ich argumentów będzie ponownie wymagało kolejnych losowań. Może to prowadzić do list o bardzo dużej długości. Z tego względu w procesie wyboru populacji początkowej zakładana jest maksymalna głębokość drzewa związanego z generowaną list.

Ocena jakości chromosomów

Poszukiwania optymalnych osobników w programowaniu genetycznym mogą być prowadzone według :

- skalarnej funkcji kryterialnej (optymalizacja jednokryterialna);
- wektorowej funkcji kryterialnej (optymalizacja wielokryterialna).

Operator selekcji chromosomów

Selekcja przeprowadzana według mechanizmów stosowanych w AG i AE, tzn.:

- metodą proporcjonalną;
- metodą turniejową;
- metodą rankingową.

Operator krzyżowania

Zadaniem krzyżowania jest wymiana składowych list LISP-owych. Zasadę takiego krzyżowania łatwiej wyjaśnić można wykorzystując reprezentacji w postaci drzew.

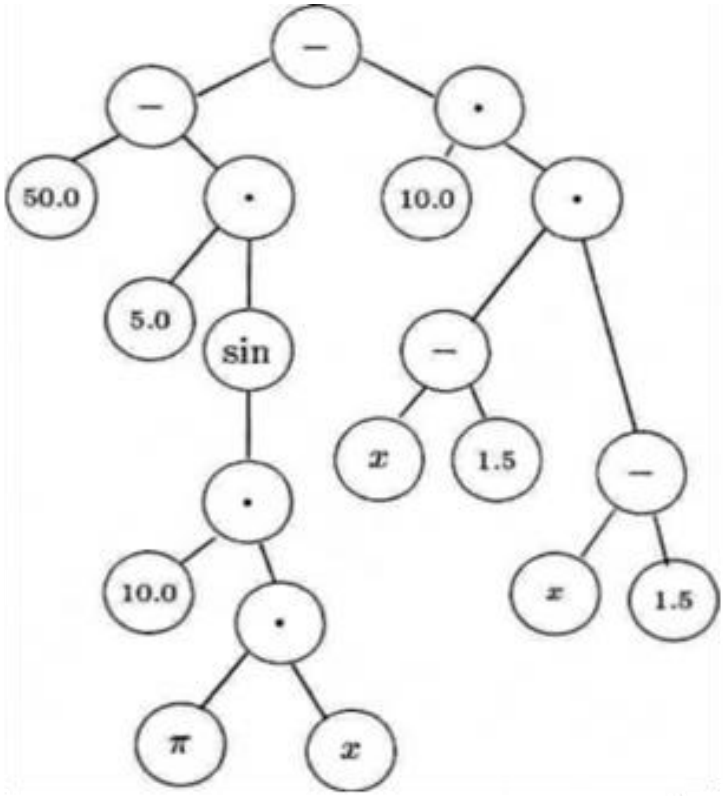
Początkową fazą krzyżowania jest losowy wybór list podlegających przekształceniu, a następnie w każdej z nich losowy wybór węzłów (w drzewach odpowiadających listom). W **wylosowanych węzłach** następuje cięcie i następnie wymiana odpowiednich fragmentów obu list (czyli chromosomów wybranych w procesie selekcji).

Rozpatrzmy zatem drzewa dwóch chromosomów wybranych do krzyżowania. Drzewa te reprezentują listy następujących funkcji:

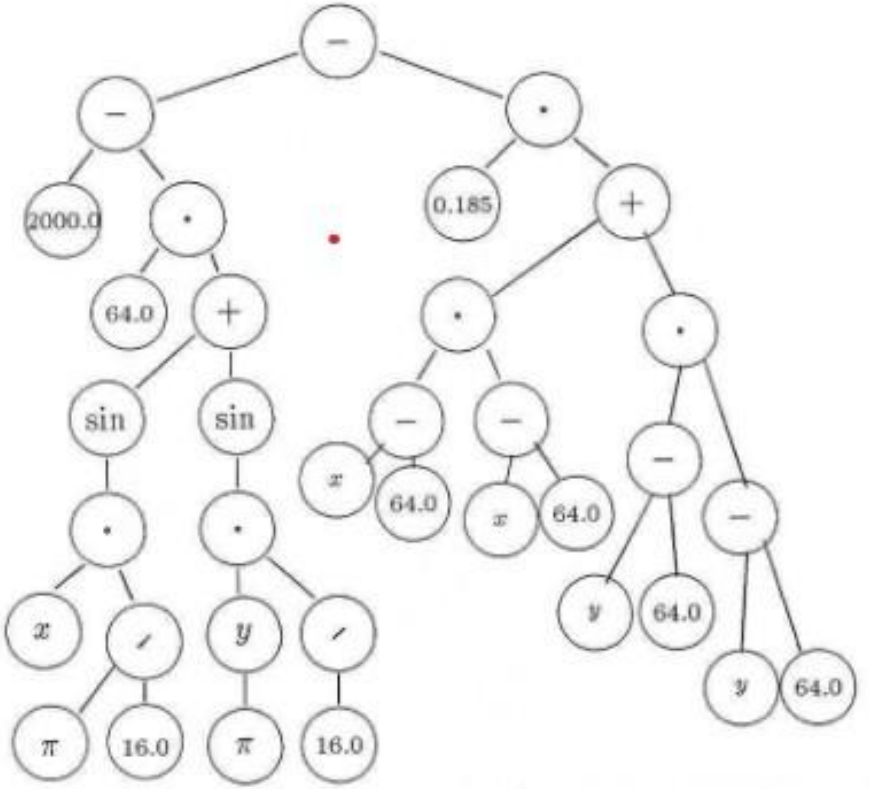
Chr.1: $f(x) = 50.0 - 5.0 \cdot \sin(10.0 \cdot \pi \cdot x) - 10.0 \cdot (\mathrm{x} - 1.5)^2$

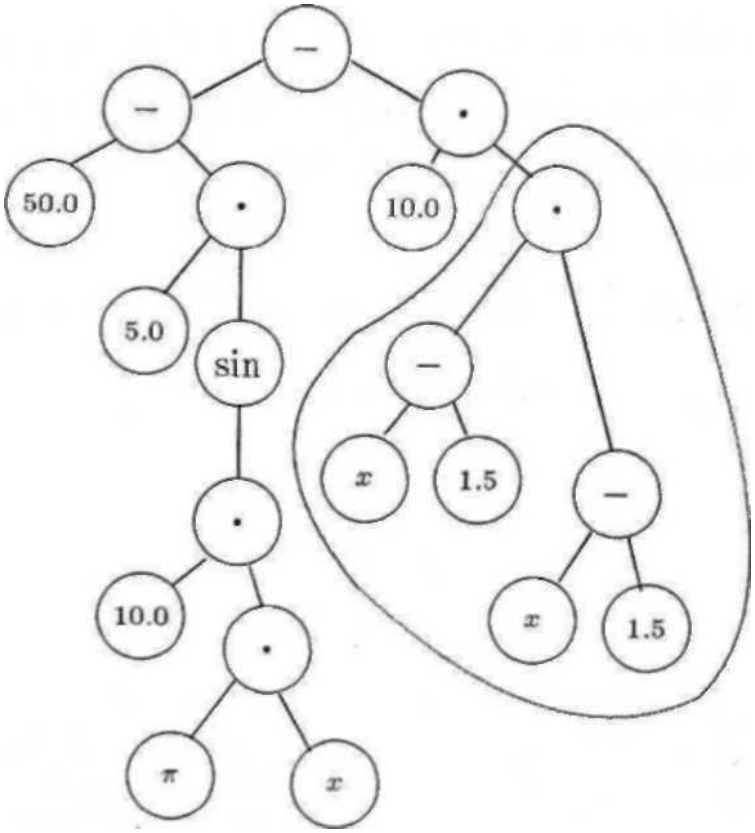
Chr 2: $f(x) = 2000.0 - 64.0 \cdot (\sin(\frac{x}{16.0\pi}) + \sin(\frac{y \cdot \pi}{16.0})) - 0.185 \cdot ((x - 64.0)^2 + (\mathrm{y} - 64.0)^2)$

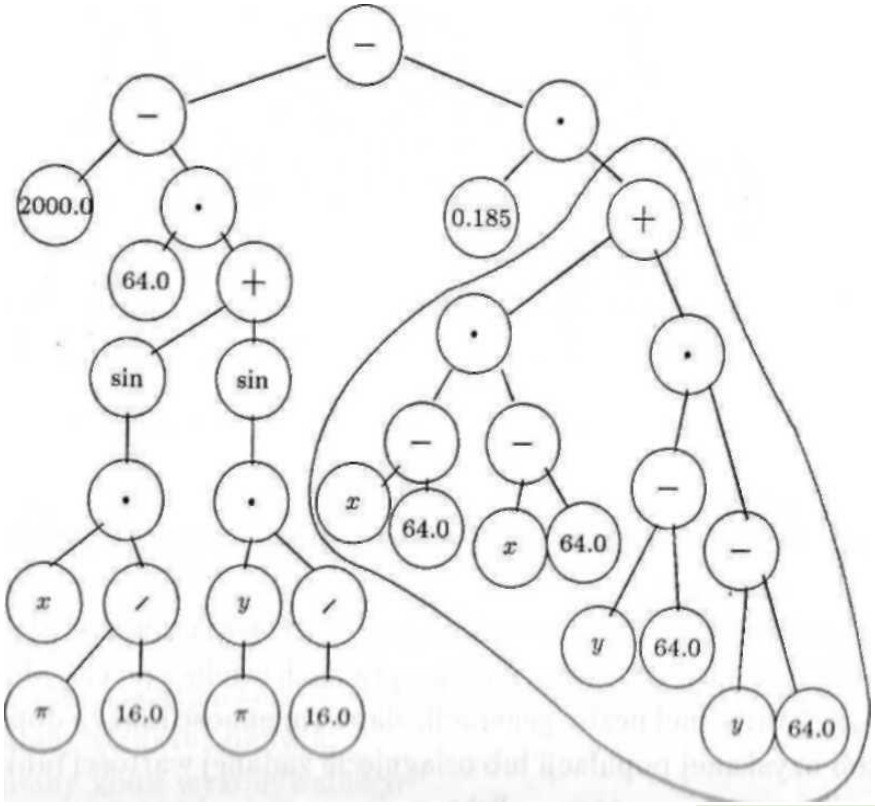
Ch. 1


50.0
5.0
sin
10.0
π
x
10.0
x
1.5
x
1.5

Ch. 2

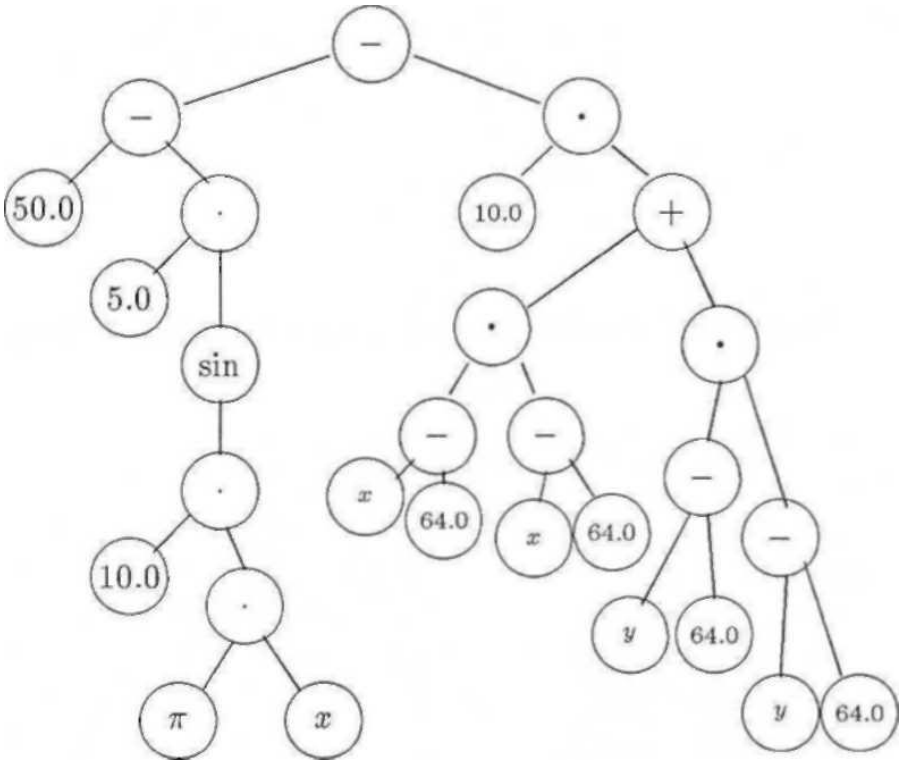

2000.0
64.0
sin
sin
x
π
16.0
y
π
16.0
0.185
x
64.0
x
64.0
y
64.0
y
64.0



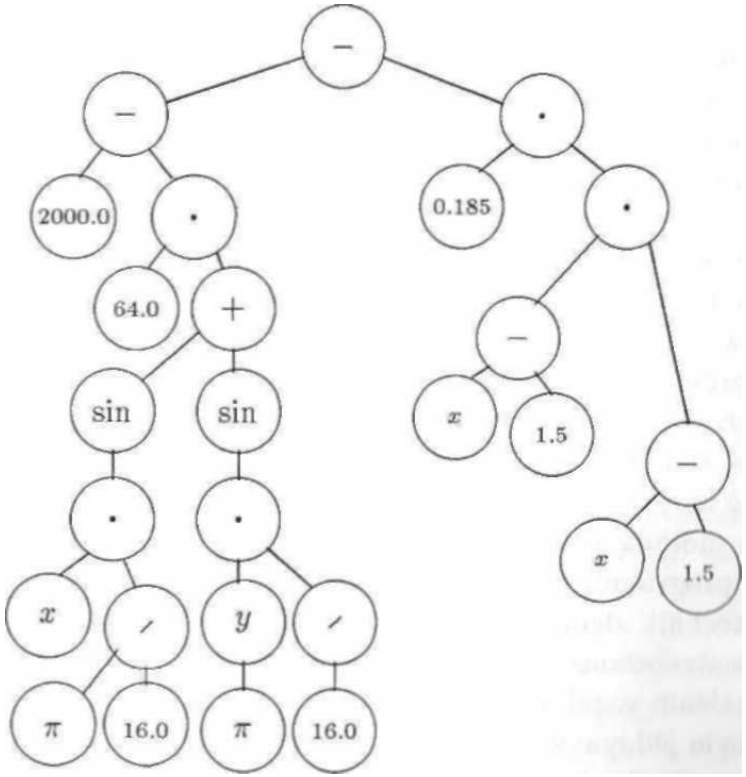


Na rysunkach powyżej zaznaczono fragmenty obu drzew przeciętych w węzłach. Wymieniając zaznaczone fragmenty pomiędzy chromosomem 1 i 2 otrzymamy dwa chromosomy potomne.

Potomek 1



Potomek 2



Funkcje reprezentowane przez potomków mają teraz postać:

P1: $f(x) = 50.0 - 5.0 \cdot \sin(10.0 \cdot \pi \cdot x) - 10.0 \cdot ((x - 64.0)^2 + (y - 64.0)^2)$

P2: $f(x) = 2000.0 - 64.0 \cdot (\sin(\frac{x}{16.0\pi}) + \sin(\frac{y \cdot \pi}{16.0})) - 0.185 \cdot (x - 1.5)^2$

Podsumowywując:

R1: $f(x) = 50.0 - 5.0 \cdot \sin(10.0 \cdot \pi \cdot x) - 10.0 \cdot (x - 1.5)^2$

R2: $f(x) = 2000.0 - 64.0 \cdot (\sin(\frac{x}{16.0\pi}) + \sin(\frac{y \cdot \pi}{16.0})) - 0.185 \cdot ((x - 64.0)^2 + (y - 64.0)^2)$

P1: $f(x) = 50.0 - 5.0 \cdot \sin(10.0 \cdot \pi \cdot x) - 10.0 \cdot ((x - 64.0)^2 + (y - 64.0)^2)$

P2: $f(x) = 2000.0 - 64.0 \cdot (\sin(\frac{x}{16.0\pi}) + \sin(\frac{y \cdot \pi}{16.0})) - 0.185 \cdot (x - 1.5)^2$

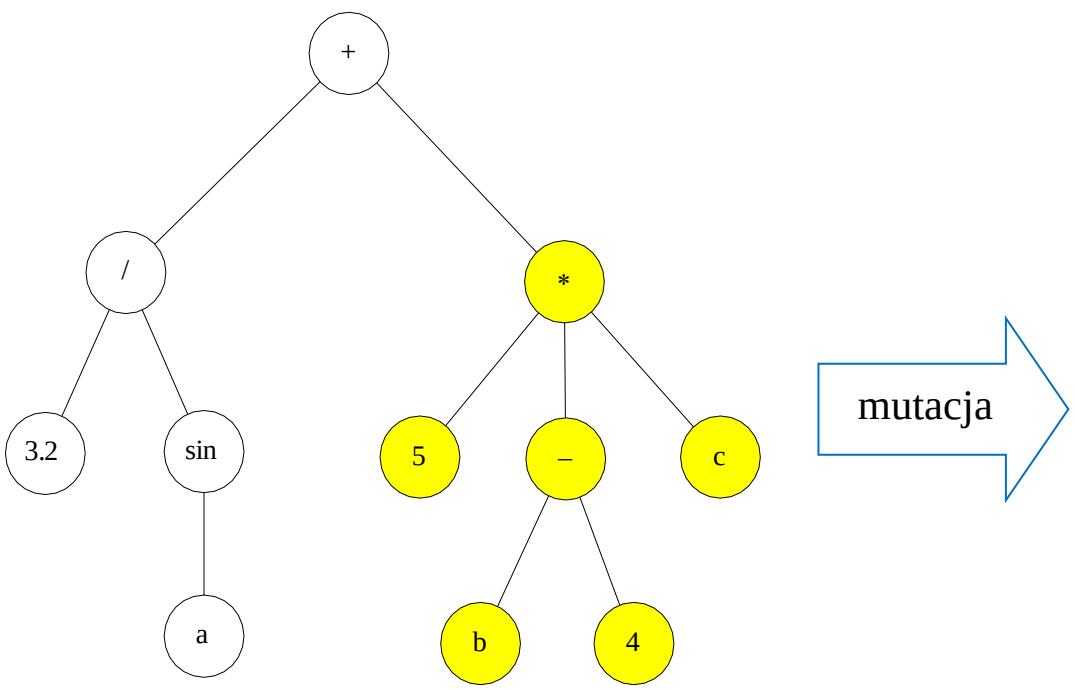
Powstały zatem dwa nowe chromosomy reprezentujące sobą dwie nowe funkcje, generujące nowe wartości ich przystosowania do rozwiazywania aktualnego problemu.

Operator mutacji

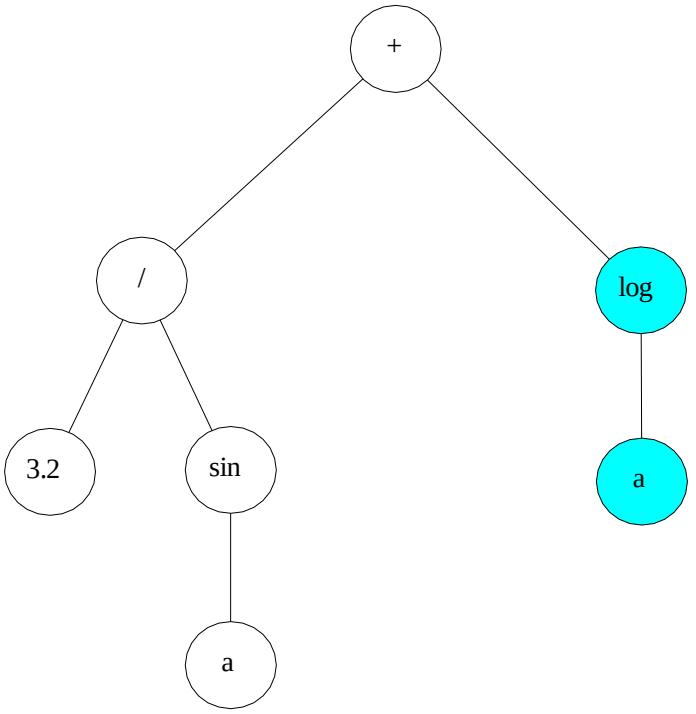
Mutacja osobnika kodowanego drzewem może być przeprowadzona w następujących trzech odmianach poprzez:

- zmianę poddrzewa
- zmianę węzła pośredniego
- zmianę węzła terminalnego
- reorganizację drzewa

- Zmiana poddrzewa



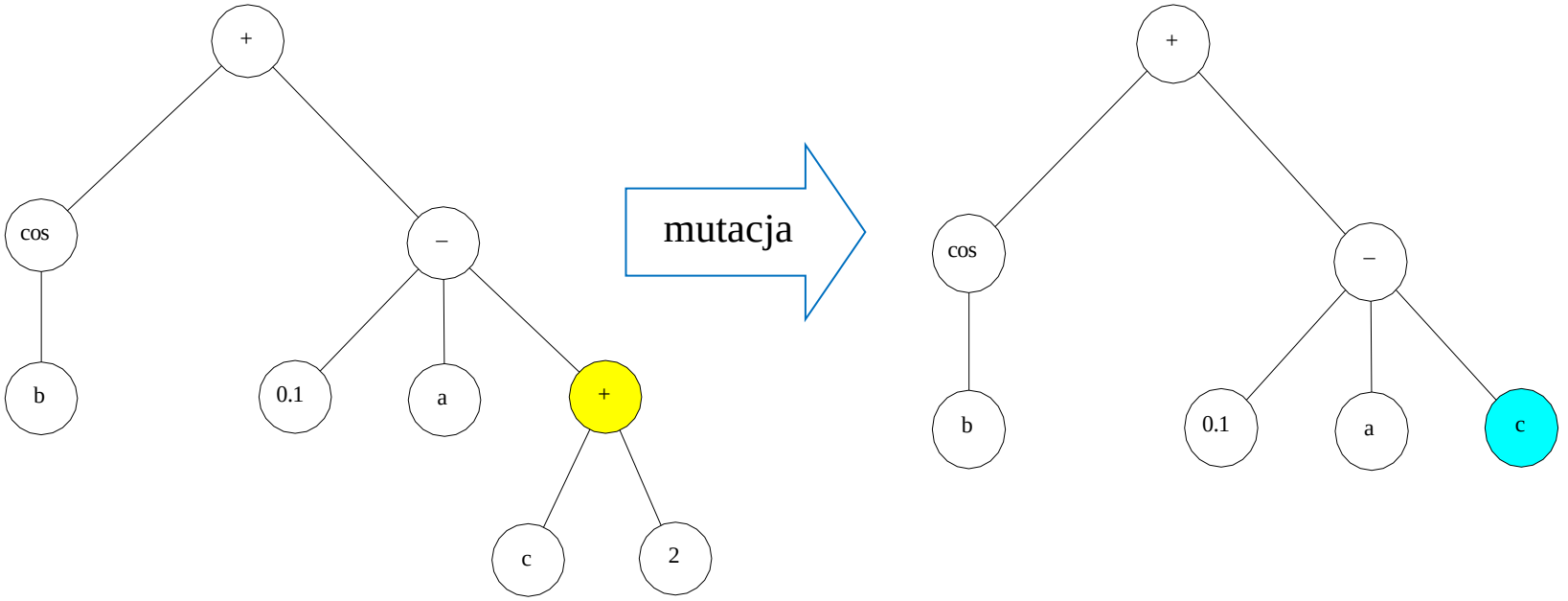
mutacja



osobnik
(+ (/ 3.2 (sin b)) (5 (– b 4) c))
3.2/sin(b)+5c(b–4)

osobnik zmutowany
(+ (/ 3.2 (sin b)) (log a))
3.2/sin(b)+log(a)

Zmiana węzła pośredniego



osobnik

(+ (cos b) (– 0.1 a (+ c 2)))

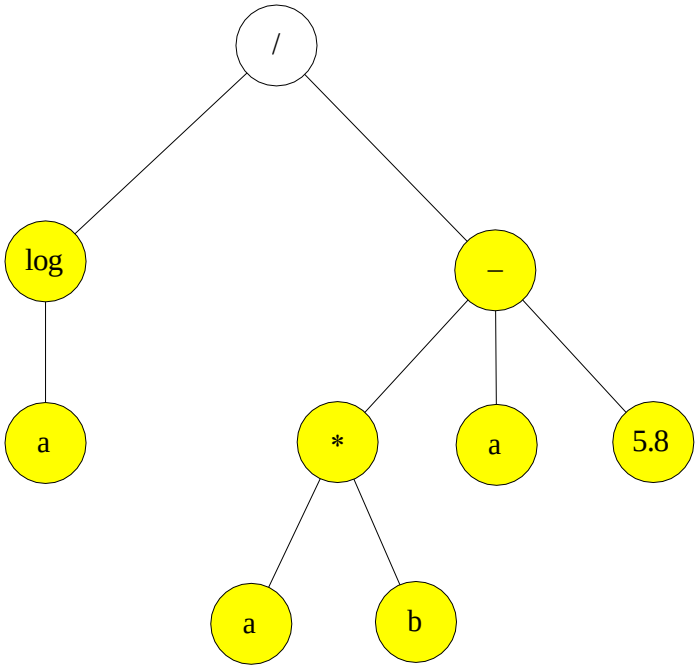
cos(b)+ 0.1– a–(c+2)

osobnik zmutowany

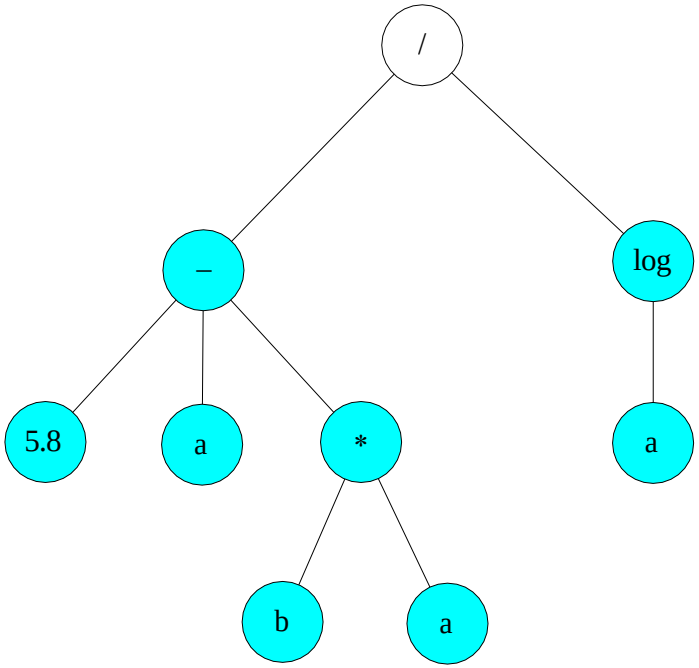
(+ (cos b) (– 0.1 a c))

cos(b)+ 0.1– a–c

Zmiana węzła terminalnego



mutacja



osobnik
(/ (log a) (– (∗ a b) a 5.8))
log(a)/(ab–a–5.8)

osobnik zmutowany
(/ (– 5.8 a (∗ b a)) (log a))
(ab–a–5.8)/log(a)

KONIEC WYKŁADÓW

POWODZENIA NA ZALICZENIU.