Politechnika Wrocławska

Programowanie genetyczne

Dariusz Banasiak
Katedra Informatyki Technicznej
Wydział Elektroniki

Programowanie genetyczne

Programowanie genetyczne jest rozszerzeniem klasycznego algorytmu genetycznego i jest wykorzystywane do automatycznego generowania programów komputerowych.

W programowaniu genetycznym dokonywana jest ewolucja programów komputerowych w celu znalezienia programu optymalnego dla danego zadania.

Populacja programów podlega działaniu odpowiednich operatorów genetycznych (reprodukcja, krzyżowanie itd.).

- Michael L. Cramer - A Representation for the Adaptive Generation of Simple Sequential Programs (1985)
- John R. Koza - *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (1992)

Programowanie genetyczne

Początkowo programowanie genetyczne dotyczyło syntezy programów w języku LISP (J. Koza 1992, 1994). Obecnie programowanie genetyczne jest stosowane dla programów napisanych w innych językach np. C, Pascal.

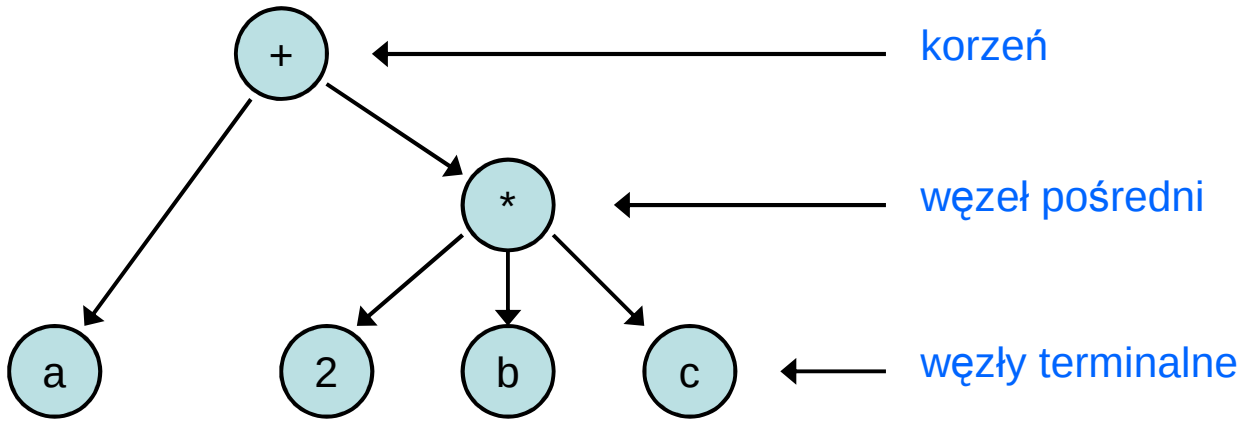
John Koza zaproponował w swojej pracy przedstawienie programów w postaci hierarchicznej, drzewiastej struktury.

W związku z tym w programowaniu genetycznym chromosomy kodowane są jako drzewa składające się z węzłów i krawędzi. Podstawowa informacja o programie zawarta jest w węzłach. Krawędzie określają jedynie wzajemne relacje między węzłami.

Wyróżnia się następujące typy węzłów:

- korzeń
- węzły pośrednie
- węzły terminalne

Przykład drzewa dla wyrażenia *a+2*b*c*:



Programowanie genetyczne

Programowanie genetyczne rozpoczyna się od utworzenia w sposób losowy początkowej populacji programów złożonych z funkcji i terminali właściwych dla obszaru zastosowania. Funkcjami mogą być:

- standardowe operacje arytmetyczne,
- standardowe operatory programowe,
- standardowe funkcje matematyczne,
- funkcje logiczne,
- funkcje specjalizowane w danej domenie zastosowań.

Zależnie od zastosowania przetwarzane mogą być wartości typu boolowskiego (logiczne), całkowite, rzeczywiste, zespolone, wektorowe, symboliczne lub złożone struktury.

Każdy program w populacji należy ocenić w jakim stopniu realizuje postawione zadanie. Do tej oceny służy miara dopasowania (*fitness measure*). Jej natura jest uzależniona od problemu. Może to być:

- liczba zdobytych punktów w grze (programy realizujące strategie gier)
- liczba prawidłowo rozpoznanych wzorców (programy klasyfikujące)
- czas (paliwo, pieniądze) niezbędne aby doprowadzić system do pożądanego stanu (programy sterowania)

Może to być też minimalizacja błędów generowanych przez program, zwięzłość zapisu, efektywność wykonywania programu i inne.

W typowym przypadku ocena przeprowadzana jest poprzez uruchomienie każdego programu dla grupy przypadków testowych (*fitness cases*).

Miarą dopasowania jest wówczas suma lub średnia wyników dla poszczególnych przypadków.

Przypadki testowe to na przykład różne wartości zmiennych lub różne stany początkowe systemu.

Programy w populacji początkowej będą miały bardzo słabe dopasowanie. Zasada przeżycia i reprodukcji najlepszych, a następnie krzyżowania służy do tworzenia populacji potomnych.

Programy w populacji różnią się rozmiarem i formą. Potomkowie będą składać się z fragmentów pochodzących od rodziców takich jak:

- poddrzewa
- podprogramy
- elementarne bloki składowe

Struktura i rozmiar potomków są najczęściej różne niż u rodziców.

Przez rekombinację losowo wybranych fragmentów częściowo efektywnych programów można uzyskać program bardziej efektywny.

Ważną cechą jest dynamiczna zmienność programów. Określenie z góry kształtu czy rozmiaru programu byłoby trudne i nienaturalne. Mogłoby to doprowadzić do takiego zawężenia obszaru poszukiwań które uniemożliwiłoby znalezienie rozwiązania.

Po reprodukcji i krzyżowaniu powstaje nowa populacja. W każdej fazie procesu ewolucji jego stan jest zawarty tylko w bieżącej populacji. W kolejnych pokoleniach widać jednak tendencję wzrostową średniego dopasowania programu.

W typowym przypadku jako rezultat uzyskany w wyniku programowania genetycznego przyjmowany jest najlepszy osobnik (program) uzyskany we wszystkich pokoleniach.

Kolejne operacje w programowaniu genetycznym

1. Wygenerowanie początkowej populacji programów.
2. Wykonanie iteracyjne następujących akcji:
   - uruchomienie każdego programu z populacji i określenie dla niego wartości funkcji przystosowania
   - utworzenie nowej populacji programów przez zastosowanie dwóch podstawowych operacji:
     - kopiowanie istniejącego programu do nowej populacji,
     - tworzenie nowego programu przez krzyżowanie (rekombinację losowo wybranych części dwóch istniejących programów)
3. Najlepszy program który pojawił się w dowolnym pokoleniu jest rozwiązaniem optymalnym (lub bliskim optymalnego).

Reprezentacja programów

Konwencjonalne algorytmy genetyczne operują na chromosomach w postaci łańcuchów o stałej długości.

Reprezentacja taka nie jest odpowiednia do zakodowania rozwiązania w postaci programu.

Program jest strukturą o zmiennej postaci i rozmiarze, parametry te nie są znane z góry. Ponadto są one zmienne dla różnych programów, które mogą powstać w trakcie ewolucji.

Wybór długości ciągu kodowego ograniczyłby z góry liczbę możliwych stanów wewnętrznych programu, a w konsek-wencji możliwość osiągnięcia rozwiązania.

W programowaniu genetycznym rozmiar, forma i zawartość struktur reprezentujących programy komputerowe może zmieniać się dynamicznie w czasie ewolucji.

Zbiór możliwych struktur jest zestawem wszystkich możliwych funkcji utworzonych rekurencyjnie ze zbioru F:

$$F = \{ f_1, f_2, \ldots , f_{Nfunc}\}$$

oraz zbioru terminali T:

$$T = \{a_1, a_2, \ldots , a_{Nterm}\}.$$

Każda z funkcji f_i w zbiorze funkcji F ma określoną liczbę $z(f_i)$ argumentów.

Funkcjami w zbiorze F mogą być:

- operatory arytmetyczne (+, -, *, itp.)
- funkcje matematyczne (np. sin, cos, exp, log, itd.)
- operatory logiczne (np. AND, OR, NOT, itp.)
- operatory warunkowe (np. IF_THEN_ELSE)
- funkcje realizujące iteracje (np. DO-UNTIL)
- funkcje realizujące rekurencję
- inne zdefiniowane, specyficzne dla dziedziny zastosowania.

Terminalami są typowo:

- zmienne
- stałe.

Zmienne reprezentują stan wejść, sensorów lub zmienne określające stan systemu.

Stałymi mogą być liczby (np. 3), stałe logiczne (np. 0 i 1).

Niekiedy terminalami są funkcje bez argumentów, których rola polega na oddziaływaniu na stan systemu.

Przykład

Zbiór funkcji: F = {AND, OR, NOT}
Zbiór terminali: T = {D0, D1}

D0 i D1 - zmienne logiczne (argumenty funkcji).

Można utworzyć połączony zbiór funkcji i terminali:

$C = F \cup T = \{AND, OR, NOT, D0, D1\}$

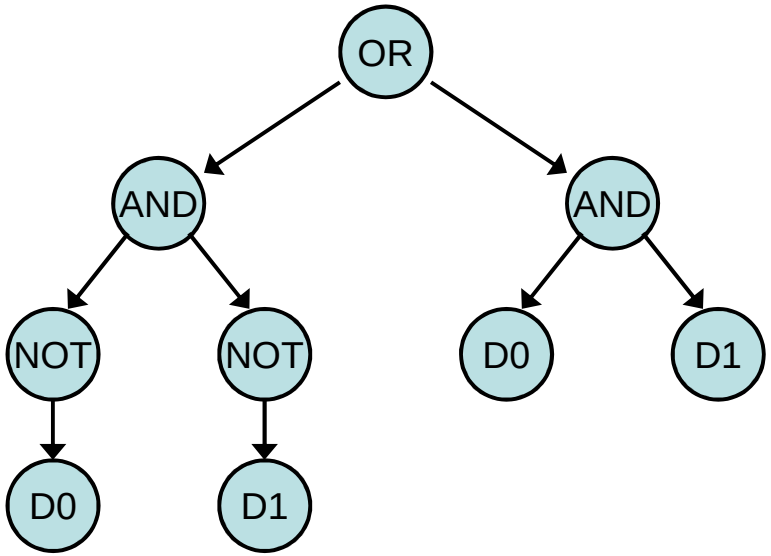
Terminale w tym zbiorze mogą być traktowane jako funkcje z zerową liczbą argumentów. Liczba argumentów dla kolejnych elementów zbioru C wynosi wówczas:

2, 2, 1, 0, 0

Funkcję EXCLUSIVE-NOR (detekcja parzystości) można wówczas przedstawić w postaci wyrażenia w LISP:

(OR (AND (NOT D0) (NOT D1)) (AND D0 D1))

lub w postaci drzewa:



Domknięcie zbiorów funkcji i terminali

Własność domknięcia wymaga, aby każda funkcja mogła akceptować jako swoje argumenty:

- dowolną wartość i typ danych zwracane przez inną funkcję
- dowolną wartość i typ danych terminala.

Gdy zbiór funkcji składa się z funkcji logicznych, a zbiór terminali ze zmiennych logicznych (o wartościach 0 i 1) własność domknięcia jest w oczywisty sposób spełniona.

Większość programów komputerowych używa jednak danych różnego typu (zmienne liczbowe, zmienne logiczne, operatory porównania i warunkowych rozgałęzień, itd.).

Wystarczalność zbiorów funkcji i terminali

Wymaganie to żąda aby zbiory funkcji i terminali były odpowiednie dla wyrażenia rozwiązania problemu.

Przykład (trzecie prawo Keplera):

Kwadrat odległości planety od słońca jest proporcjonalny do pierwiastka okresu obiegu planety wokół słońca.

Terminale: odległość_PS, okres_PS, ...

Funkcje: mnożenie, potęgowanie, pierwiastkowanie, ...

W wielu przypadkach konieczna jest więc wiedza specyficzna dla danej dziedziny.

Postać końcowa rozwiązania zadania zwykle zależy od przyjętego zbioru funkcji. Dla funkcji boolowskich możemy przyjąć jeden z zestawów:

{AND, OR, NOT}, {IF, AND, OR, NOT}, {NAND}, {NOR}

Postać rozwiązania będzie różna w każdym przypadku.

Wpływ nadmiarowych funkcji na rozwiązanie jest złożony. Przyjmuje się, że powodują one pogorszenie efektywności dochodzenia do rozwiązania (niekiedy jest odwrotnie).

Wpływ nadmiarowych terminali jest bardziej jednoznaczny. Prawie zawsze powodują one pogorszenie wydajności.

Struktury początkowe

Populacja początkowa składa się z losowo wygenerowanych wyrażeń (drzew). Z zestawu funkcji losowany jest korzeń drzewa (unikamy powstania struktury składającej się tylko z jednego terminala).

Następnie dla każdej gałęzi jest wybierany losowo ze zbioru $C = F \cup T$ element umieszczany na końcu gałęzi.

Podstawowe metody generowania drzew:

- metoda pełna (full method)
- metoda wzrostu (grow method)
- metoda mieszana

Głębokość drzewa – największa długość ścieżki (bez powrotów) od korzenia drzewa do węzła terminalnego.

W metodzie pełnej generowane są drzewa dla których długość każdej ścieżki między korzeniem i węzłem terminalnym jest równa określonej maksymalnej głębokości.

W metodzie wzrostu uzyskiwane są drzewa o różnym kształcie. Długość ścieżek jest nie większa niż przyjęte maksimum. Stosunek liczebności zbiorów F i T określa oczekiwaną długość ścieżki między korzeniem i węzłami terminalnymi.

W metodzie mieszanej tworzone są jednakowe liczby drzew o głębokości zawartej pomiędzy 2 i określoną maksymalną głębokością (np. dla maksimum równego 6 20% drzew będzie miało głębokość 2, 20% - 3 i tak dalej). Dla każdej wartości głębokości 50% drzew jest tworzonych przy użyciu metody pełnej, a 50% metodą wzrostu.

Metoda pozwala utworzyć drzewa o dużej różnorodności.

Powtarzające się w populacji egzemplarze mogą być eliminowane i zastępowane wygenerowanymi na nowo (nie zawsze jest to konieczne).

Możliwe jest również umieszczenie w populacji początkowej określonych osobników.

Dopasowanie – funkcja przystosowania

Dopasowanie jest oceniane zwykle na grupie przypadków testujących. Są one różnymi, reprezentatywnymi sytuacjami (zbiorami danych), dla których można uzyskać wystarczającą ocenę osobników w populacji.

Stanowią zwykle tylko małą próbkę całej przestrzeni problemu (zwykle bardzo dużej lub nieskończonej).

W niektórych przypadkach (funkcje logiczne z małą liczbą argumentów) obejmują całą przestrzeń poszukiwań.

Zbiór przypadków testowych można modyfikować w kolejnych pokoleniach.

Dopasowanie pierwotne - błąd będący sumą wartości bezwzględnych różnic między rozwiązaniem optymalnym a uzyskanym dla wszystkich przypadków testowych:

$$r(i,t) = \sum_{j=1}^{N_c} | S(i,j) - C(j) | \text{ , gdzie:}$$

i – numer osobnika,

j – numer przypadku testowego,

N_c – liczba przypadków testowych,

t – numer pokolenia,

C(j) – prawidłowa wartość dla przypadku j,

S(i,j) – dopasowanie dla osobnika i na przypadku testowym j.

Dopasowanie standaryzowane - wartość s(i,t) dopasowania pierwotnego r(i,t) przekształconego tak, aby mniejsza wartość liczbowa reprezentowała zawsze lepszą wartość:

$s(i,t) = r(i,t)$

$s(i,t) = r_{max} - r(i,t)$, gdzie r_{max} - najlepsza możliwa wartość r(i,t)

Dopasowanie skompensowane - dopasowanie przeskalowane do zakresu (0,1) w ten sposób, że większa wartość oznacza lepsze dopasowanie:

$a(i,t) = 1/(1 + s(i,t))$

Dopasowanie znormalizowane – wykorzystywane, gdy do utworzenia nowej populacji jest stosowana selekcja proporcjonalna do dopasowania:

$$n(i,t) = a(i,t) / \sum_{k=1}^{M} a(k,t) \text{ , gdzie}$$

M – wielkość populacji.

Dopasowanie to ma następujące, korzystne własności:

- zakres wartości (0,1)
- większe wartości dla lepszych osobników
- suma wartości znormalizowanych dla całej populacji wynosi 1.

Tworzenie nowych populacji

Podstawowe operatory do tworzenia nowej populacji to:

- reprodukcja
- krzyżowanie.

Reprodukcja

Reprodukcja polega na wyborze osobnika (programu) z danej populacji i przekopiowaniu go (bez zmian) do nowej populacji. Najbardziej popularna jest selekcja proporcjonalna do dopasowania (np. dopasowanie znormalizowane).

Stosowane są również inne metody np. metoda rang lub metoda turniejowa.

Krzyżowanie

Dwa osobniki, które będą pełnić funkcję rodziców, wybierane są na podstawie ich funkcji dopasowania.

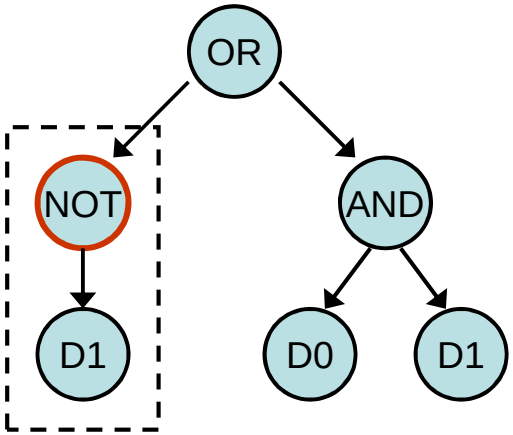
Następnie w każdym z rodziców wybierany jest (niezależnie) w sposób losowy punkt krzyżowania.

Fragment używany przy krzyżowaniu jest poddrzewem o korzeniu w punkcie krzyżowania i zawierającym cały fragment poniżej punktu krzyżowania.

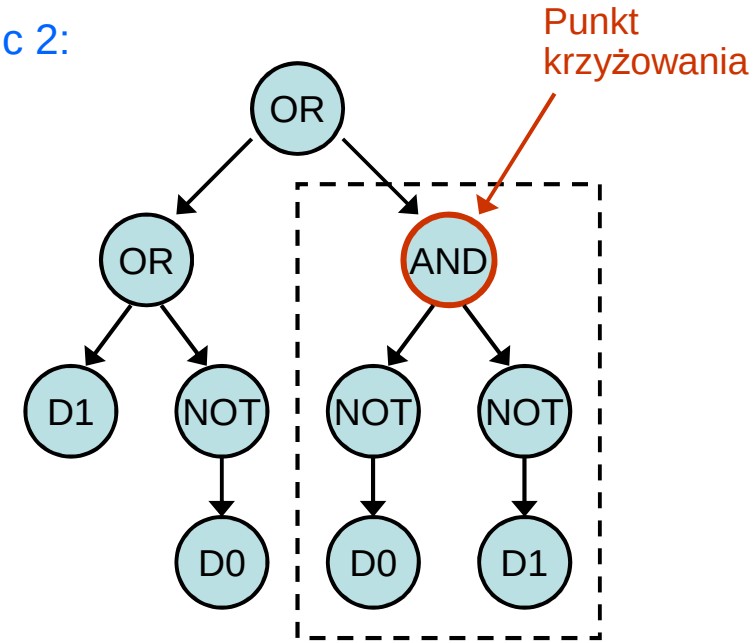
Poddrzewo może zredukować się do jednego tylko terminala lub stanowić całe drzewo dla danego rodzica.

Przykład

Rodzic 1:

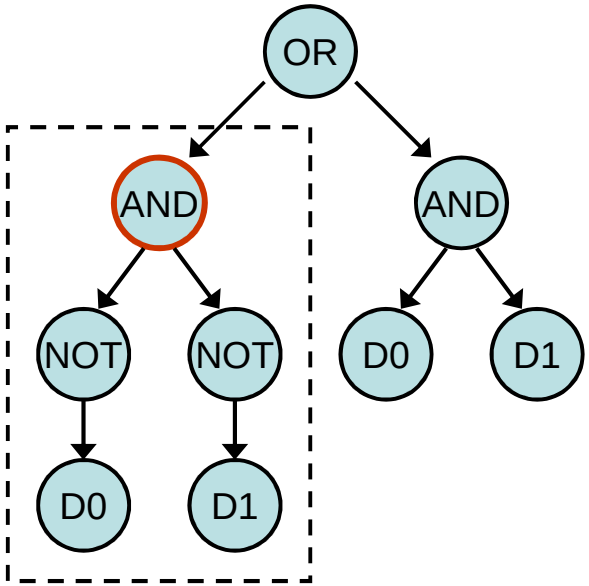


Rodzic 2:

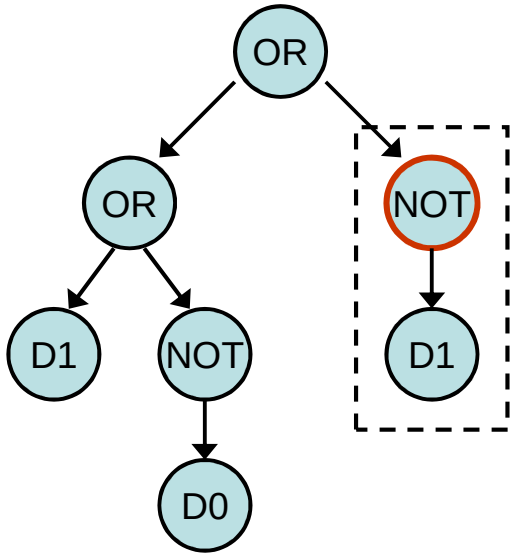


Wynik krzyżowania:

Potomek 1:



Potomek 2:



Uzyskany przez krzyżowanie potomek może być drzewem o większej głębokości niż przyjęta wartość maksymalna.

Wówczas taki osobnik jest odrzucany, a jego miejsce zajmuje pierwszy z rodziców.

Koza w swoich eksperymentach przyjął maksymalną głębokość równą 17.

Największy możliwy program (funkcje dwuargumentowe) zawiera wówczas $2^{17} = 131072$ funkcji i terminali.

Zakładając przelicznik 4 funkcje i terminale na jedną linię kodu daje to program w konwencjonalnym języku programowania o długości około 33000 linii.

Operacje pomocnicze w programowaniu genetycznym:

- mutacja
- permutacja
- edycja
- enkapsulacja
- dziesiątkowanie (przerzedzanie)

Kryteria zakończenia przebiegu algorytmu:

- wykonanie określonej liczby przebiegów (wygenerowanie określonej liczby pokoleń)
- spełnienie określonego warunku przez najlepsze rozwiązanie.

Metody określania rozwiązania optymalnego:

- najlepszy dotychczasowy rezultat (w każdym pokoleniu wybierany i zapamiętywany jest najlepszy osobnik)
- najlepszy w populacji końcowej.

Podstawowe parametry

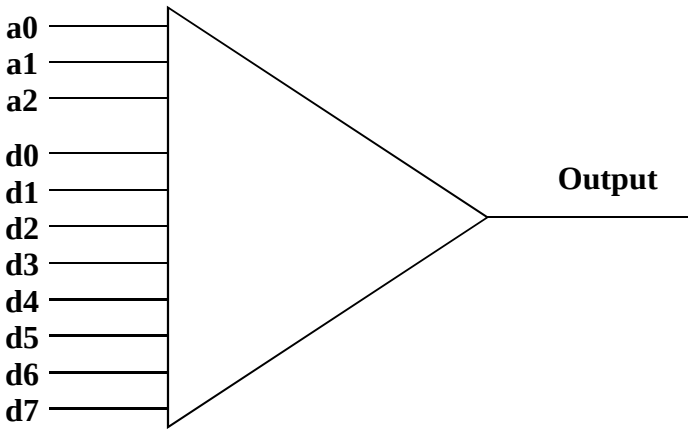
John Koza dla większości przedstawianych w pracy przykładów stosował jednakowe parametry:

- wielkość populacji: M = 500
- liczba pokoleń: 51 (pokolenie początkowe + 50 kolejnych)
- prawdopodobieństwo krzyżowania $p_c = 0.9$
- prawdopodobieństwo reprodukcji $p_r = 0.1$ (dla M = 500 jest to 50, dopuszczalny wielokrotny wybór)

- dla operacji krzyżowania rozkład wyboru punktów wewnętrznych (funkcyjnych) i zewnętrznych (terminali) wynosi 90% i 10% - promuje to rekombinacje większych struktur
- maksymalny rozmiar (głębokość drzewa) w operacji krzyżowania $D_{created} = 17$
- maksymalny rozmiar (głębokość drzewa) dla populacji początkowej $D_{initial} = 6$
- prawdopodobieństwa mutacji, permutacji, enkapsulacji $p_m = p_p = p_{en} = 0$
- częstotliwość edycji $f_{ed} = 0$
- brak operacji dziesiątkowania.

Przykład zastosowania

Synteza multipleksera o 8 liniach wejściowych i 3 liniach adresowych:



Dla multipleksera o k liniach adresowych:

- **n = k+2k wejść,**
- **o = 2^n kombinacji argumentów (liczba wierszy w tabeli prawdy)**
- **f = 2^o różnych możliwych funkcji.**

Dla k=3 otrzymujemy:

- **n = 3 + 23 = 11**
- **o = 2^{11} = 2048**
- **f = $2^{2048} \approx 10^{616}$**

W pełni poprawne rozwiązanie uzyskane w 9 pokoleniu: