Anna Paszyńska

Projektowanie wspomagane komputerowo a problemy zbieżności algorytmów genetycznych

Promotor:
dr hab. Ewa Grabska

Spis treści.

1 Wstęp

1.1 Cel i motywacja pracy

Pojawienie się komputerów doprowadziło do wielkiego postępu w dziedzinie projektowania. Niemal od początku ich istnienia zadaniem pewnej klasy programów była poprawa jakości i szybkości projektowania z jednoczesnym obniżeniem kosztów. Obecnie istnieje tendencja obejmowania wspomaganiem komputerowym całego procesu projektowania, wytwarzania i eksploatacji produktów. Systemy wspomagające projektowanie są wykorzystywane w architekturze, projektowaniu inżynierskim, wzornictwie przemysłowym, sztuce użytkowej, sztuce komputerowej, reklamie oraz modzie.

W kontekście informatyki projektowanie można traktować jako poszukiwanie optymalnych rozwiązań danego problemu w przestrzeni możliwych rozwiązań. Jedną z metod poszukiwania są algorytmy ewolucyjne [9, 11, 22, 23, 35].

Ze względu na udany kompromis pomiędzy złożonością obliczeniową a zdolnością eksploracji wielkich zbiorów, można je stosować do zadań trudnych, w których funkcja celu jest wielomodalna, przy braku informacji o ilości oraz przybliżonej lokalizacji rozwiązań. Dzięki powyższym zaletom, algorytmy ewolucyjne są stosowane w wielu dziedzinach nauki, takich jak: uczenie się maszynowe [35, 27, 42, 93, 97], systemy sterowania [52], sztuczne życie, teoria gier [3], robotyka [20], komponowanie muzyki [43, 56, 79], mechanika konstrukcji [70] czy zarządzanie. Coraz częściej są też stosowane w różnych dziedzinach projektowania [9, 15, 16, 17, 18, 24, 26, 33, 34, 45, 48, 62, 71, 88, 100, 104].

Algorytmy ewolucyjne od momentu powstania budzą duże zainteresowanie nie tylko praktyków, którzy próbują je zastosować w różnych dziedzinach, ale również teoretyków, zmierzających do zbadania złożonej natury ich działania. W zależności od różnych podejść, możemy wyróżnić kilka głównych teoretycznych nurtów w algorytmach ewolucyjnych: mikroskopowe modele systemów dynamicznych (modele markowowskie), krajobrazy poszukiwań (ang. fitness landscapes), analiza komponentów, teoria schematów czy NFL (ang. No Free Lunch Theorems).

Aktualnie uwaga teoretyków skupia się głównie na modelach markowowskich [21, 61, 85, 86, 87, 101, 102]. Istniejące modele matematyczne oparte na łańcuchach Markowa zostały stworzone

dla prostego algorytmu genetycznego z kodowaniem binarnym [61, 101], a następnie zostały rozszerzone na przypadek algorytmu genetycznego z kodowaniem w systemie o większej liczbie cyfr [49]. Znane jest tylko jedno uogólnienie modelu markowowskiego dla bardziej złożonych algorytmów ewolucyjnych [76] – model łańcuchów markowa dla programowania genetycznego i algorytmów genetycznych z krzyżowaniem homologicznym. Jedynym rezultatem teoretycznym otrzymanym na podstawie tego uogólnienia było obliczenie macierzy przejścia algorytmu genetycznego.

Celem pracy jest zbudowanie modelu matematycznego umożliwiającego badanie własności algorytmów genetycznych bazujących na strukturach hierarchicznych. Praca prezentuje rozszerzenie modelu markowowskiego stworzonego przez Vose'a dla prostego algorytmu genetycznego o nowe operatory: operatory przesunięcia i permutacji. Na podstawie tego modelu została udowodniona ergodyczność łańcucha markowa opisującego dynamikę algorytmu genetycznego oraz zbieżność w sensie zbieżności miar.

Otrzymane wyniki teoretyczne pozwolą na badanie własności innych algorytmów genetycznych bazujących na strukturach hierarchicznych, dla których do tej pory nie wykazano własności asymptotycznych ani zbieżności. Przykładem takich algorytmów jest algorytm genetyczny z hierarchicznym chromosomem.

Drugim celem pracy będzie analiza algorytmu genetycznego z hierarchicznym chromosomem [10], używanego w projektowaniu wspomaganym komputerowo, na podstawie rozszerzonego modelu markowowskiego.

Dodatkowo, w celu dokładniejszego zbadania algorytmu genetycznego z hierarchicznym chromosomem, zostanie zastosowane również drugie podejście - mikroskopowa teoria schematów Poli'ego. Podejście to umożliwi obliczenie oczekiwanej liczby osobników pasujących do schematu H w następnej generacji oraz efektywnego przystosowania dla algorytmu genetycznego z na hierarchicznym chromosomem.

1.2 Ogólny zarys pracy

Prezentowana praca obejmuje dwie dziedziny informatyki: projektowanie wspomagane komputerowo oraz algorytmy ewolucyjne. Utworzenie systemu wspomagającego projektowanie w oparciu o algorytm genetyczny wymagało znalezienia odpowiedniej reprezentacji projektowanych obiektów. Reprezentacja ta powinna spełniać zarówno kryteria wymagane przez system projektowania jak i umożliwić efektywne działanie algorytmu genetycznego. Sposób reprezentowania obiektów ma wpływ na wybór odpowiedniego dla danej reprezentacji algorytmu ewolucyjnego.

Praca niniejsza obejmuje dziewięć rozdziałów.

Rozdział drugi poświęcony jest wymaganiom, jakie powinna spełniać reprezentacja bryły oraz istniejącym modelom reprezentacji. Spośród tych modeli została wybrana najbardziej odpowiednia w naszym zadaniu projektowym reprezentacja – za pomocą prostopadłościanów z płaszczyzną przecinającą (ang. Clipped Stretched Cubes).

W rozdziale trzecim omówiono schemat działania algorytmu ewolucyjnego oraz popularną notację związaną ze schematami reprodukcji i sukcesji stosowanymi w algorytmach ewolucyjnych. Następnie przedstawiono charakterystykę algorytmów genetycznych, programowania genetycznego, programowania ewolucyjnego oraz strategii ewolucyjnych. Struktury nieliniowe bardzo często używane są jako reprezentacja obiektów w wielu algorytmach ewolucyjnych – zwłaszcza tych, które są wykorzystywane do projektowania wspomaganego komputerowo. W rozdziale tym omówione zostaną szczegółowo dwa podejścia: reprezentacja z chromosomem hierarchicznym oraz reprezentacja grafowa.

Rozdział czwarty przedstawia istniejące teorie algorytmów genetycznych. Dokładniej zostały omówione dwa podejścia, które w tej pracy zostały wykorzystane: mikroskopowe modele systemów dynamicznych, czyli modele markowowskie oraz teoria schematów.

Rozdział piąty opisuje markowowski model algorytmu genetycznego rozszerzonego o dodatkowe operatory: przesunięcie w prawo i w lewo oraz permutację. Wymieniony algorytm genetyczny zostanie zamodelowany za pomocą jednorodnego łańcucha Markowa o skończonej liczbie stanów. Model ten zostanie wykorzystany do obliczenia macierzy przejścia algorytmu genetycznego oraz do wykazania, że wszystkie stany należą do jednej klasy stanów

istotnych. Udowodniona zostanie również ergodyczność łańcucha Markowa opisującego algorytm oraz zbieżność w sensie zbieżności miar. Powyższy model może zostać wykorzystany do modelowania algorytmów genetycznych bardziej złożonych niż prosty algorytm genetyczny.

W rozdziale szóstym przedstawiony został model algorytmu genetycznego z hierarchicznym chromosomem o zmiennej długości. Przedstawione wyniki teoretyczne: obliczona macierz przejścia, ergodyczność łańcucha Markowa oraz zbieżność w sensie zbieżności miar zostały otrzymane na podstawie rozszerzonego modelu markowowskiego opisanego w rozdziale piątym.

Rozdział siódmy opisuje wyniki teoretyczne otrzymane na podstawie ogólnej teorii schematów dla programowania genetycznego z krzyżowaniem polegającym na przepinaniu poddrzew. Teoria bazować będzie na notacji kartezjańskiego systemu referencyjnego i hiperschematach wprowadzonych przez Polie'go w 2001 roku [73]. Po odpowiednim zakodowaniu genotypów w kartezjańskim systemie referencyjnym, krzyżowaniu hierarchicznemu będzie odpowiadało krzyżowanie uwarunkowane typem, opisane przez D. J. Montana [59]. Na podstawie prezentowanej teorii otrzymane zostaną dokładne definicje prawdopodobieństwa, że stworzone osobniki pasują do schematu oraz efektywnego przystosowania.

Rozdział ósmy opisuje zaimplementowaną aplikację bazującą na algorytmie genetycznym z hierarchicznym chromosomem. Ponieważ aplikacja została zaprojektowana jako otwarta, może być stosowana do rozwiązywania wielu różnorodnych problemów optymalizacji. W rozdziale przedstawione zostało rozwiązanie przykładowego problemu -problemu optymalizacji kształtu platformy.

Rozdział dziewiąty zawiera posumowanie osiągniętych wyników.

2 Reprezentacja obiektów trójwymiarowych

2.1 Wprowadzenie

Reprezentacja obiektów (w tej pracy fenotypów) jest jednym z najważniejszych elementów systemu projektowania. Zdefiniowanie sposobu reprezentowania projektów określa zarazem przestrzeń dostępnych rozwiązań generowanych przez system. Reprezentacja fenotypów odgrywa również istotną rolę w określeniu rozmiaru i złożoności genotypów ze względu na to, że fenotypy muszą mieć odpowiadający sobie genotyp, aby umożliwić algorytmowi genetycznemu działanie. W tej pracy fenotypy będą projektami trójwymiarowymi.

Typowy system optymalizacji projektów optymalizuje jedynie wybrane parametry. Nasze podejście będzie bardziej ogólne, ponieważ będziemy generować projekty ze szkiców. Tworząc nowe projekty system powinien być w stanie zmodyfikować każdą część projektu. Reprezentacja bryły powinna być tak dobrana, aby umożliwić efektywne działanie algorytmu genetycznego – przejście z każdego projektu do każdego innego projektu w ciągu kroków.

2.2 Wymagania reprezentacji

Zastosowanie algorytmu genetycznego do projektowania narzuca szereg ograniczeń na reprezentację fenotypów [10]. Reprezentacja, która opisuje projekt za pomocą dużej liczby wierzchołków będzie wymagała dużej liczby parametrów do zdefiniowania nawet najprostszych kształtów. Im więcej będzie parametrów w fenotypie, tym więcej będzie genów w genotypie, a więc problem poszukiwania rozwiązania będzie trudniejszy. Reprezentacja powinna dokładnie opisać kształt używając niewielkiej liczby parametrów.

Reprezentacja powinna również umożliwiać prostą (dla algorytmu genetycznego) modyfikację kształtów. Użycie reprezentacji, która umieszcza podobne projekty blisko siebie w przestrzeni poszukiwań, umożliwi algorytmowi genetycznemu znalezienie ścieżki ewolucyjnej od gorszych do lepszych projektów.

Projekt może być zdefiniowany na kilka sposobów, ale reprezentacja powinna definiować tylko jeden projekt.

2.3 Istniejące reprezentacje

Istnieją dwa podstawowe modele reprezentacji obiektów trójwymiarowych: modele definiujące kształt za pomocą powierzchni (przy założeniu, że przestrzeń wewnątrz powierzchni jest bryłą) oraz modele definiujące kształt bryły bardziej bezpośrednio: modele dekompozycyjne i konstruktywne. Zaletą reprezentacji brzegowej jest możliwość dokonywania zmian lokalnych. Przy reprezentacji za pomocą konstrukcyjnej geometrii brył można dokonywać tylko modyfikacji globalnych [32].

2.3.1. Modele brzegowe

W reprezentacji brzegowej [32] bryła jest traktowana w sposób hierarchiczny - obiekt jest opisany za pomocą powierzchni brzegowych (zbiór ścian); ściany są reprezentowane za pomocą ich jednowymiarowych krawędzi; z kolei krawędzie są reprezentowane za pomocą wierzchołków. W celu reprezentowania kierunku, w którym jest skierowana każda ściana numerujemy wierzchołki w porządku zgodnym z ruchem wskazówek zegara, patrząc z zewnątrz bryły. W celu uniknięcia powielania współrzędnych wspólnych dla ścian, każdy wierzchołek ściany jest reprezentowany za pomocą indeksu do listy współrzędnych.

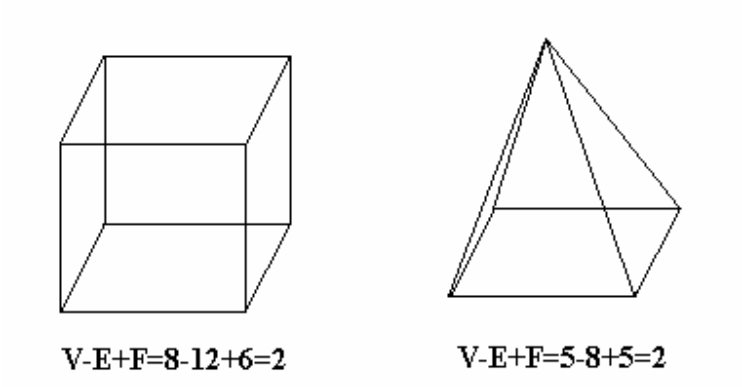
Reprezentacje brzegowe mogą być łączone za pomocą regularyzowanych operatorów boolowskich, aby utworzyć nowe reprezentacje brzegowe.

Reprezentacja brzegowa wielościanu prostego (wielościanu, który może być przekształcony w kulę – nie ma dziur) spełnia regułę Eulera, która wyraża niezmienniczą zależność między liczbą wierzchołków ścian i krawędzi wielościanu i wyraża się w następujący sposób:

$$V + F - E = 2 \tag{2.1}$$

gdzie V-jest liczbą wierzchołków, E jest liczbą krawędzi a F jest liczbą ścian.

Reguła Eulera dla przykładowych wielościanów przedstawiona jest na Rys. 2.1.

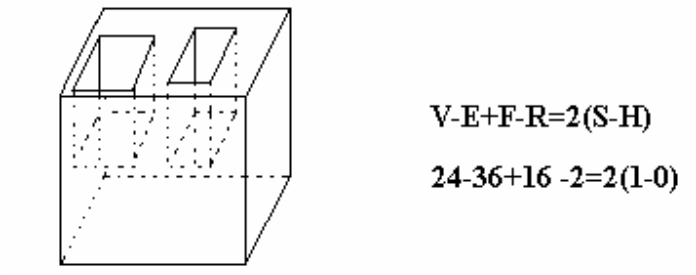


Rys. 2.1 Reguła Eulera dla przykładowych wielościanów.

Uogólnioną regułę Eulera przedstawioną wzorem:

$$V + F - E - R = 2(S - H) \tag{2.2}$$

gdzie S jest liczbą składowych bryły, H oznacza liczbę dziur a R jest liczbą pierścieni.
stosuje się do ścian z dziurami. Reguła Eulera dla przykładowego wielościanu z dwoma dziurami w górnej ścianie przedstawiona jest na Rys. 2.2.



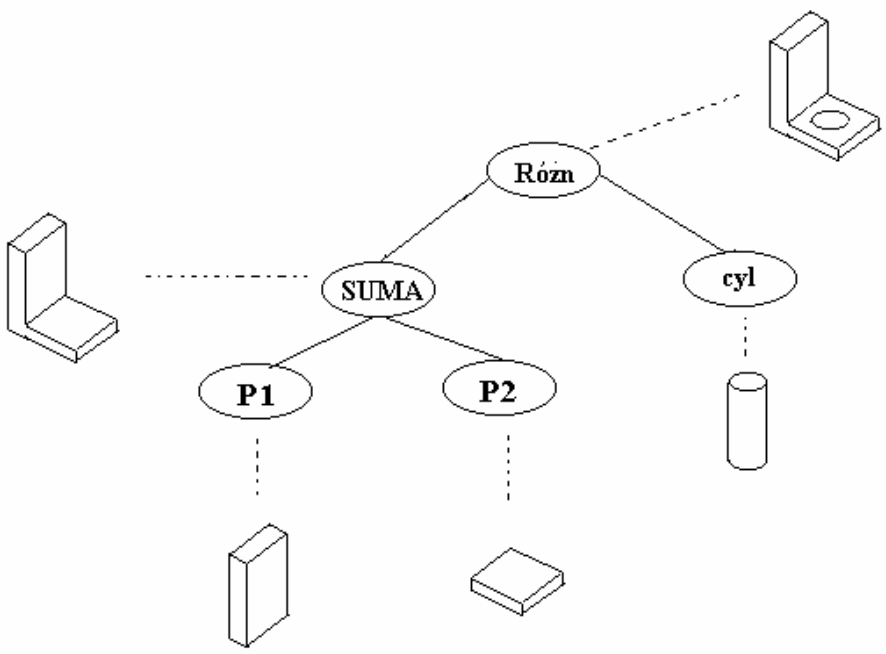
Rys. 2.2 Reguła Eulera dla przykładowego wielościanu z dwoma dziurami w górnej ścianie.

Baumgart wprowadził pojęcie zbioru operatorów Eulera, które działają na obiektach spełniających regułę Eulera w celu przekształcania ich w nowe obiekty, dla których ta reguła obowiązuje. Do operatorów tych należą operatory dodawania i usuwania wierzchołków,

krawędzi i ścian. Istnieje twierdzenie wykazujące, że operatory Eulera nie mogą tworzyć topologicznie nieważnych brzegowych struktur danych.
Przy reprezentacji brzegowej liczba parametrów potrzebna do określenia nawet najprostszego kształtu jest bardzo duża. Pozostałe reprezentacje: dekompozycyjna i konstruktywna są bardziej odpowiednie dla ewolucyjnego systemu projektowania ze względu na to, że umożliwiają zdefiniowanie bryły z dużą dokładnością przy niewielkiej liczbie parametrów.

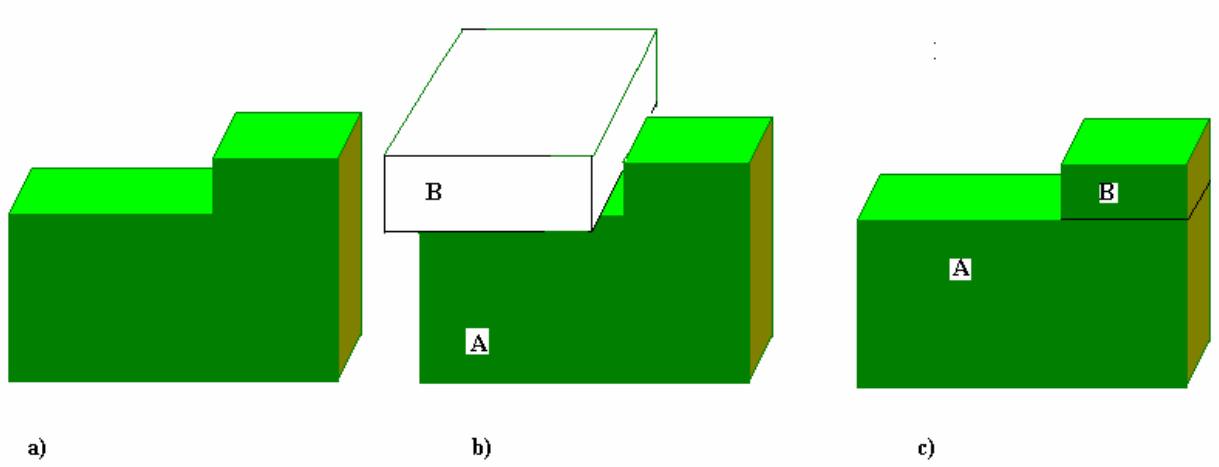
2.3.2 Modele konstruktywne

W konstruktywnej geometrii brył (ang. Constructive Solid Geometry - CSG) łączy się proste prymitywy za pomocą regularyzowanych operatorów boolowskich, które są bezpośrednio włączone do reprezentacji [32]. Obiekt jest pamiętany jako drzewo z operatorami w węzłach wewnętrznych i prostymi prymitywami w liściach. Niektóre węzły reprezentują operatory boolowskie, inne zaś wykonują przesunięcia, obroty i skalowanie. Ponieważ operatory boolowskie na ogół nie są przemienne, krawędzie drzewa są uporządkowane. Przykładowy obiekt i reprezentujące go drzewo CSG zostało przedstawione na Rys. 2.3.



Rys. 2.3 CSG drzewo.

W reprezentacji CSG używane są regularyzowane operatory boolowskie zamiast zwykłych operatorów mnogościowych, aby mieć pewność, że wykonanie operacji na bryłach zawsze w wyniku da bryłę. Dowolny regularyzowany operator boolowski można zdefiniować w zależności od odpowiedniego zwykłego operatora boolowskiego w następujący sposób:
$A \ *op\ B = domknięcie(wnętrze(A\ op\ B))$, gdzie *op* jest jedną z operacji $\cup, \cap, -$.



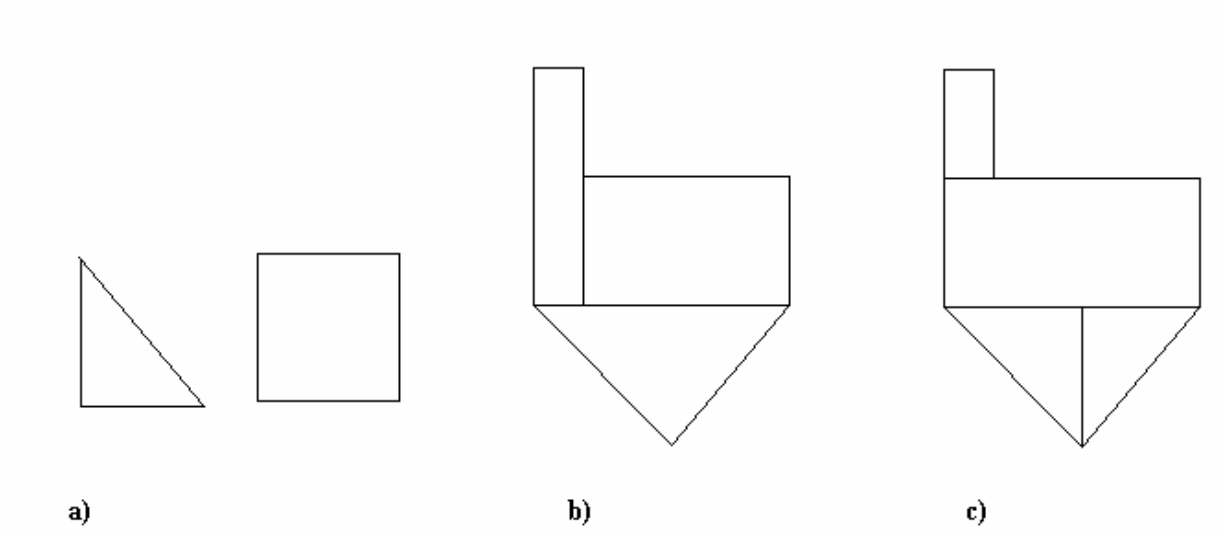
Rys. 2.4 Obiekt pokazany na rysunku a) może być zdefiniowany za pomocą różnych operacji CSG, jak na rysunkach b) i c).

Reprezentacja CSG nie zapewnia unikatowości reprezentacji. Może się więc zdarzyć, że stosując tę samą operację do dwóch początkowo takich samych obiektów, otrzymuje się dwa różne wyniki. Niemniej, dzięki możliwości dokonywania edycji przez usuwanie, dodawanie, zastępowanie i modyfikowanie poddrzew, w połączeniu ze względnie zwartym sposobem pamiętania modeli, reprezentacja CSG jest dominującą reprezentacją modelowania brył. Jednak z punktu widzenia algorytmu genetycznego nie jest ona najlepsza, ponieważ umieszcza blisko siebie w przestrzeni poszukiwań obiekty bardzo różne.

2.3.3 Modele dekompozycyjne (reprezentacje z podziałem przestrzennym)

W reprezentacjach dekompozycyjnych bryła jest dekomponowana na zbiór sąsiadujących, nie przecinających się brył, które są prostszymi prymitywami od oryginalnej bryły [32]. Prymitywy mogą być różnego typu, wielkości, mogą mieć różne położenie i orientację. Zaletą tej reprezentacji jest łatwość wprowadzania zmian oraz możliwość definiowania dowolnego kształtu trójwymiarowego (czasami jedynie poprzez aproksymację). Nie jest to reprezentacja unikalna, aczkolwiek nie jest to wadą – cecha ta może być pomocna w procesie poszukiwania rozwiązań. Dzięki temu, że bryła może być reprezentowana przez różne konfiguracje prymitywów, pojawi się w kilku miejscach w przestrzeni poszukiwań, więc potencjalnie będzie łatwiejsza do znalezienia. Metoda ta nie jest najbardziej efektywna, ze względu na liczbę parametrów.

Jedną z najogólniejszych reprezentacji z podziałem przestrzennym jest dekompozycja na komórki (podział komórkowy). Bryła jest reprezentowana jako zbiór komórek, które mogą być połączone powierzchniami bocznymi. Komórka jest obiektem topologicznie równoważnym ze sferą, zawierającym często powierzchnie zakrzywione. Reprezentacja typu dekompozycja na komórki nie jest jedyna – nawet jeden typ komórki wystarczy, aby spowodować dwuznaczności. Komórki pokazane na Rys. 2.5a mogą być przekształcane w celu skonstruowania obiektu w różny sposób tak jak na Rys.2.5b i 2.5c.



Rys. 2.5 Komórki i obiekty z nich skonstruowane.

Dekompozycja na komórki jest ważną reprezentacją przy wykonywaniu analizy metodą elementów skończonych.

Szczególnym przypadkiem dekompozycji na komórki jest reprezentacja wokselowa (wyliczanie wyczerpujące), w której bryła jest dekomponowana na identyczne komórki uporządkowane według stałej, regularnej siatki. Komórki te nazywane są wokselami. Najczęściej spotykanym rodzajem komórki jest sześcian. Przy reprezentacji obiektu metodą wokselową musimy zadecydować, które komórki są zajęte, a które nie. Obiekt może być kodowany za pomocą jednoznacznej listy zajętych komórek.

Przy takiej reprezentacji wiele brył może być tylko aproksymowanych, np. gdy komórki są sześcianami, to jedynymi obiektami, które mogą być reprezentowane dokładnie są obiekty o ścianach równoległych do ścian sześcianu i których wierzchołki pokrywają się z punktami siatki. Aby zwiększyć dokładność reprezentacji można zmniejszać komórki, ale poważnym ograniczeniem jest tutaj złożoność pamięciowa.

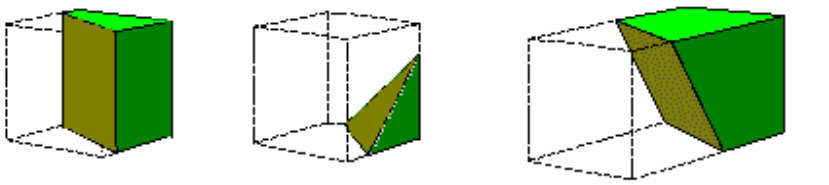
Przy zastosowaniach, w których jest duże zapotrzebowanie na pamięć używa się hierarchicznego wariantu metody wokselowej – drzew ósemkowych. Idea tej reprezentacji polega na stworzeniu sześcianu zawierającego bryłę, którą chcemy reprezentować a następnie podzieleniu tego sześcianu na osiem mniejszych sześcianów. Każdy z sześcianów będzie pełny, pusty lub częściowo zapełniony. Sześcian częściowo zapełniony jest dzielony rekursywnie na sześciany. Podział kontynuujemy tak długo, dopóki wszystkie sześciany nie są jednorodne (pełne lub puste). Następnie definiujemy drzewo, które reprezentuje charakterystykę obiektu względem sześcianu. Na drzewach wykonuje się operacje takie jak suma, różnica.

2.4 Reprezentacja za pomocą prostopadłościanów z płaszczyznami przecinającymi

W pracy, ze względu na dążenie do minimalizacji liczby parametrów, zastosujemy reprezentację za pomocą prostopadłościanów przecinanych płaszczyznami (ang. Clipped Stretched Cubes - CSC) [10].

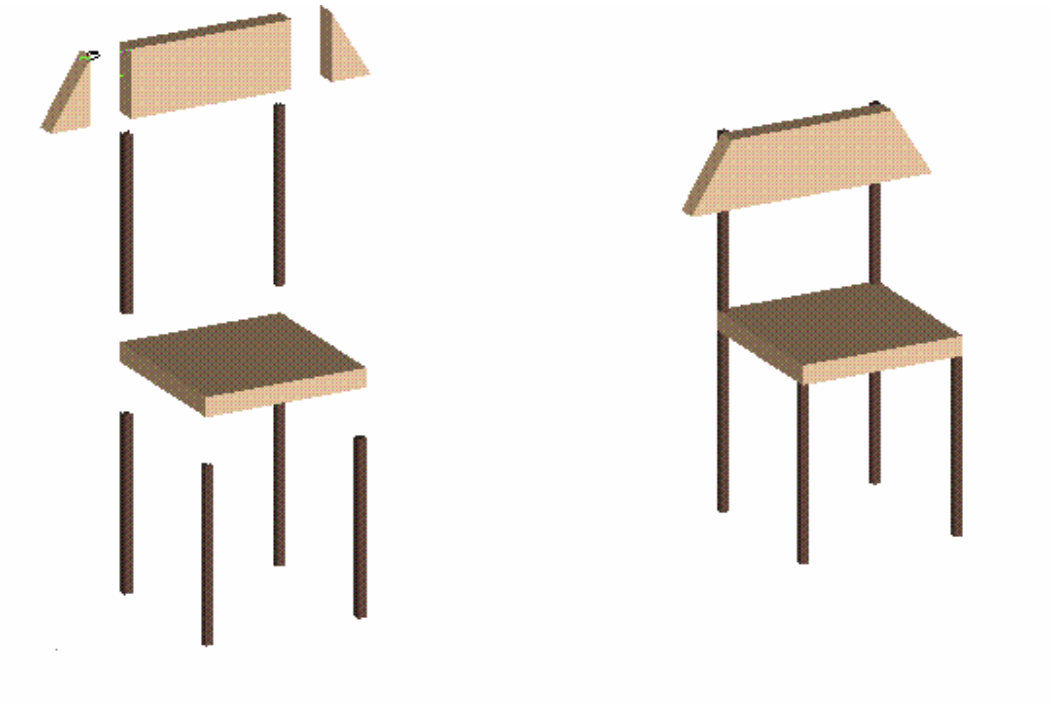
Dowolna bryła będzie zdefiniowana poprzez podzielenie jej na pewną liczbę mniejszych, nie przecinających się prymitywów. Wszędzie tam, gdzie jest to możliwe, prymitywy będą prostopadłościanami. Natomiast dla aproksymowanej powierzchni, która nie jest równoległa do

żadnej z płaszczyzn układu współrzędnych *XYZ* prymitywy będą utworzone z prostopadłościanów przeciętych płaszczyznami w odpowiednim kierunku. Na Rys.2.6 przedstawione zostały przykładowe prymitywy.



Rys.2.6 Przykładowe prymitywy.

Rys. 2.7 przedstawia przykładowy obiekt trójwymiarowy (krzesło) oraz prymitywy, z których został on otrzymany.



Rys.2.7 Przykładowe prymitywy oraz krzesło z nich otrzymane.

Każdy taki prymityw będzie opisany poprzez dziewięć następujących parametrów:

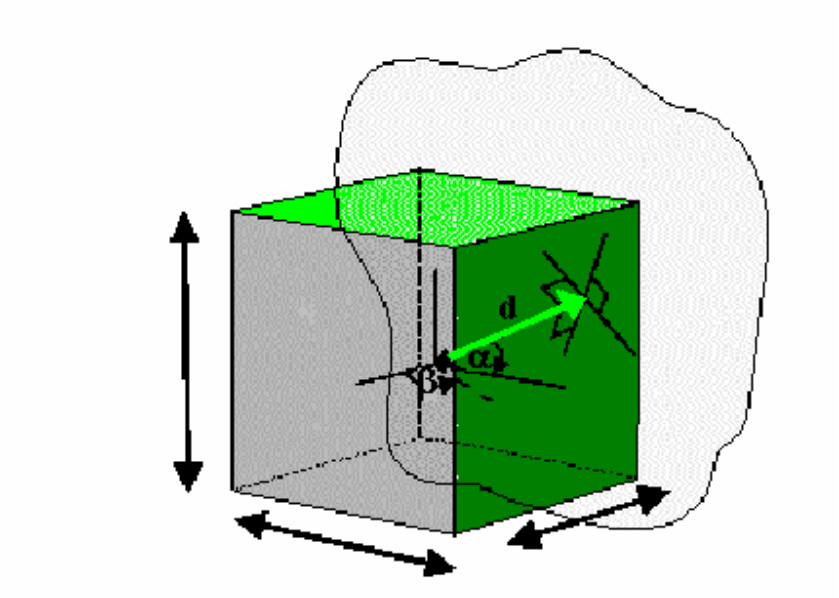
1-3. współrzędne środka prostopadłościanu (x, y, z) ,

4. wysokość *w*,
5. szerokość *s*,
6. głębokość g,

7-9. parametry definiujące płaszczyznę przecinającą – kąty α, β, oraz odległość płaszczyzny przecinającej od środka prymitywu *d* (trzy parametry).

Na Rys. 2.8. przedstawiony został prymityw z zaznaczonymi parametrami

(x, y, z, w, s, g, α, β, d)



Rys.2.8 Prymityw z zaznaczonymi parametrami *(x, y, z, w, s, g, α, β, d).*

Dla parametrów α, β, d wyznaczających płaszczyznę jej równanie wygląda następująco:

$$Ax + By + Cz + D = 0 \tag{2.3}$$

gdzie

$$A = d \cos\beta \sin\alpha$$
$$B = d \sin\beta$$
$$C = d \cos\beta \cos\alpha$$
$$D = -(A^2 + B^2 + C^2)$$

Gdy prymityw nie ma płaszczyzny przecinającej, współrzędne jego wierzchołków wyrażone są następującymi wzorami:

$$\begin{aligned}
&(x + s/2, y + g/2, z + w/2) \\
&(x + s/2, y - g/2, z + w/2) \\
&(x + s/2, y + g/2, z - w/2) \\
&(x + s/2, y - g/2, z - w/2) \\
&(x - s/2, y + g/2, z + w/2) \\
&(x - s/2, y - g/2, z + w/2) \\
&(x - s/2, y + g/2, z - w/2) \\
&(x - s/2, y - g/2, z - w/2)
\end{aligned} \tag{2.4}$$

Natomiast, w sytuacji, gdy prymityw ma płaszczyznę przecinającą, trzeba obliczyć wierzchołki tego prymitywu powstałe przez przecięcie prostopadłościanu z tą płaszczyzną.

Dla dwóch wierzchołków wyznaczających krawędź prymitywu $V_1(x_1,y_1,z_1)$ i $V_2(x_2,y_2,z_2)$ oraz dla płaszczyzny zdefiniowanej równaniem $Ax+By+Cz+D=0$ można wyliczyć współrzędne punktu przecięcia $P(x,y,z)$.

Warunki istnienia punktu przecięcia $P(x,y,z)$ pomiędzy wierzchołkami V_1 i V_2 są wyrażone za pomocą następujących wzorów:

$$x_2 \geq x \geq x_1 \tag{2.5}$$
$$y_2 \geq y \geq y_1$$
$$z_2 \geq z \geq z_1$$

gdzie

$$x = \frac{Bx_1(y_2\text{-}y_1) + Cx_1(z_2\text{-}z_1)\text{-}(By_1 + Cz_1 + D)(x_2\text{-}x_1)}{A(x_2 - x_1) + B(y_2 - y_1) + C(z_2 - z_1)} \tag{2.6}$$

$$y = \frac{Ay_1(x_2 - x_1) + C_{y_1}(z_2\text{-}z_1)\text{-}(A_{x_1} + C_{z_1} + D)(y_2 - y_1)}{A(x_2 - x_1) + B(y_2 - y_1) + C(z_2 - z_1)}$$

$$z = \frac{Az_1(x_2 - x_1) + Bz_1(y_2 - y_1)\text{-}(A_{x_1} + By_1 + D)(z_2 - z_1)}{A(x_2 - x_1) + B(y_2 - y_1) + C(z_2 - z_1)}$$

Gdy już mamy obliczone wierzchołki powstałe z przecięcia krawędzi prymitywu z płaszczyzną przecinającą, trzeba usunąć wierzchołki które już teraz nie należą do naszego prymitywu. Można to zrobić sprawdzając odległość między każdym wierzchołkiem a płaszczyzną. Oznaczając ją przez *odl* wystarczy dla wierzchołka *V(r,s,t)* i płaszczyzny danej wzorem *Ax+By+Cz+D* obliczyć odległość z równania 2.7:

$$odl = \frac{Ar + Bs + Dt}{\sqrt{A^2 + B^2 + C^2}} \tag{2.7}$$

Ponieważ interesuje nas tylko znak obliczanego wyrażenia wystarczy ograniczyć się do rozwiązania równania:

$$odl = Ar + Bs + Dt \tag{2.8}$$

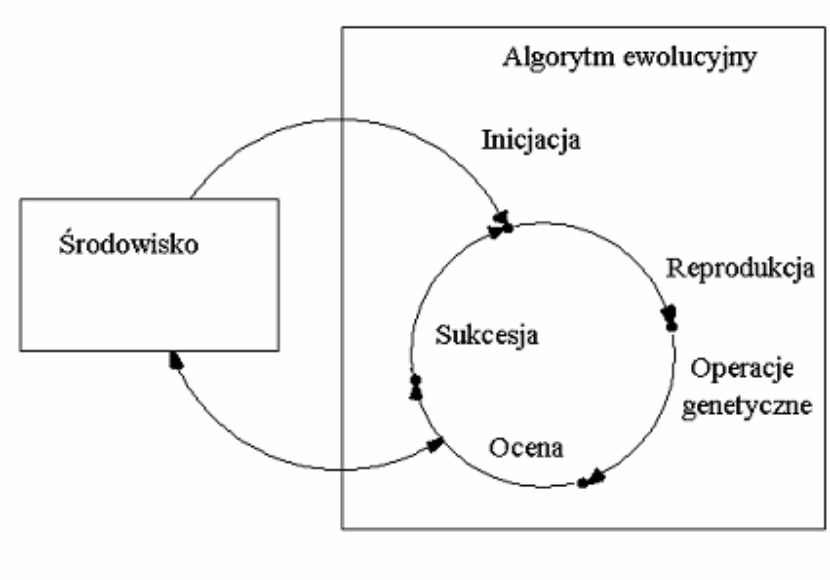
Jeżeli dla *d>0* odległość między wierzchołkiem a płaszczyzną jest ujemna, albo dla *d<0* dodatnia, to taki wierzchołek można usunąć z prymitywu.

3 Algorytmy ewolucyjne

3.1 Wstęp

Wiele problemów w informatyce definiowanych jest jako przeszukiwanie pewnej przestrzeni. Przestrzeń ta zawiera wszystkie możliwe rozwiązania danego problemu, a każdy jej punkt odpowiada jednemu rozwiązaniu [47]. **Algorytmami ewolucyjnymi** nazywamy algorytmy przeszukiwania przestrzeni rozwiązań z wykorzystaniem zasad sztucznej ewolucji [58]. Algorytmy te oparte są na zasadach ewolucji naturalnej [40, 41].

Algorytm ewolucyjny przetwarza pewien podzbiór przestrzeni poszukiwań zwany **populacją. Osobniki,** czyli elementy populacji ewoluują zgodnie z **regułami selekcji** (wyboru) oraz **operatorami genetycznymi**. Wśród operatorów tych można wyróżnić mutację (ang. mutation) oraz krzyżowanie (ang. crossover). Algorytm ewolucyjny działa w **środowisku,** które można zdefiniować na podstawie **problemu** rozwiązywanego przez algorytm. W środowisku każdemu osobnikowi przyporządkowana jest wartość liczbowa określająca jakość reprezentowanego przez niego rozwiązania; wartość ta nazywana jest **przystosowaniem osobnika**. Przystosowanie osobnika zależy od tego jak dobrze spełnia on kryteria rozważanego problemu. Środowisko można opisać **funkcją przystosowania**, za pomocą której osobnikowi przypisuje się przystosowanie. Działanie algorytmu ewolucyjnego przedstawione zostało na Rys. 3.1.



Rys. 3.1 Schemat działania algorytmu ewolucyjnego.

Działanie algorytmu ewolucyjnego polega na wykonywaniu w pętli reprodukcji, operacji genetycznych, oceny i sukcesji. Reprodukcję i sukcesję określa się czasami wspólną nazwą selekcja. Reprodukcja oraz operatory genetyczne modelują rozmnażanie, podczas którego materiał genetyczny rodziców jest przekazywany dzieciom. Podczas reprodukcji zostają powielane losowo wybrane osobniki z populacji bazowej. W procesie tym algorytm ewolucyjny koncentruje się na osobnikach lepiej przystosowanych, czyli takich, które w procesie oceniania dostali najwyższe wartości. Osobnikom tym algorytm pozwala mieć więcej potomków, podczas gdy osobniki gorzej przystosowane mają mniejszą szansę przekazania swojego materiału genetycznego do następnej populacji. Powstałe w wyniku reprodukcji kopie, zwane osobnikami rodzicielskimi, poddawane są operacjom genetycznym: mutacji oraz krzyżowaniu. Osobniki utworzone w wyniku działania operatorów genetycznych tworzą populację potomną. Populacja potomna poddawana jest ocenie środowiska. Następnym etapem jest sukcesja – tworzona jest nowa populacja bazowa. Sposób doboru osobników zarówno do reprodukcji jak i do sukcesji zależy od zastosowanej metody selekcji. Inicjacja pętli ewolucji polega na utworzeniu początkowej populacji bazowej poprzez wygenerowanie genotypów osobników i obliczenie ich przystosowania. Najczęściej proces inicjacji jest losowy.

Poniżej przedstawiona została popularna notacja związana ze schematami reprodukcji i sukcesji stosowanymi w algorytmach ewolucyjnych. Notacja ta przypisywana jest Schwefelowi [94], następnie została rozszerzona przez Rechenberga [80] i Rudolpha [87].

$(\mu+\lambda)$ Algorytm, który produkuje λ potomków i wybiera osobników do następnej populacji o liczności μ z sumy zbiorów potomków i rodziców, którego liczność wynosi $\mu+\lambda$.

(μ,λ) Algorytm, który produkuje λ potomków i wybiera osobników do następnej populacji o liczności μ tylko spośród potomków. Zazwyczaj zakłada się, że $\mu \geq \lambda$, aby selekcja nie degenerowała materiału genetycznego. Przypadek $\mu=\lambda=1$ wykorzystujący tylko mutację jest klasycznym błądzeniem losowym.

$(\mu \overset{+}{,} \lambda)$ Algorytm, który należy do jednej z opisanych powyżej klas (do ich sumy mnogościowej). Przy stosowaniu tego oznaczenia sprecyzowana jest jedynie liczność populacji μ i liczba potomków λ wytwarzanych w każdym kroku ewolucji.

(μ,κ,λ) Strategią typu $(\mu \overset{+}{,} \lambda)$, dla której selekcja odbywa się poprzez porównanie czasu życia osobnika algorytmów jego maksymalną wartością κ. Strategie tego typu opisano w [90].

$\left[\mu' \overset{+}{,} \lambda'\left(\mu \overset{+}{,} \lambda\right)^{\gamma}\right]^{\gamma'}$ Zagnieżdżony algorytm wielopopulacyjny, dla którego struktura próby losowej składa się z μ' populacji, każda licząca μ osobników. Są one wykorzystywane do wytwarzania λ' nowych populacji początkowych po λ osobników każda. Dla tych populacji początkowych wykonywany jest γ razy algorytm typu $\left(\mu \overset{+}{,} \lambda\right)$. Schemat może być wykonywany γ' razy.

Wśród algorytmów ewolucyjnych można wyróżnić:

- algorytmy genetyczne,
- programowanie genetyczne,
- programowanie ewolucyjne,
- strategie ewolucyjne.

Wszystkie te algorytmy opierają się na zasadach ewolucji. Istnieją jednak różnice pomiędzy poszczególnymi typami.

3.2 Algorytmy genetyczne

Algorytmy genetyczne są najbardziej znane spośród algorytmów ewolucyjnych [22, 23, 35, 41, 58]. Powstały one w wyniku prac Johna Hollanda, dotyczących modelowania ewolucji populacji osobników, z których każdy wyposażony był w binarny kod genetyczny. J. Holland wprowadził mechanizmy przekształcania słów kodowych w sposób podobny do operacji krzyżowania i mutacji oraz mechanizm selekcji. Przypisanie osobnikom wartości liczbowych zależnych od kodu genetycznego doprowadziło do spostrzeżenia, że w wyniku działania operatorów genetycznych średnia wartość przystosowania populacji rośnie. Algorytm ten został później rozpowszechniony przez Dawida Goldberga.

Cechą charakterystyczną algorytmów genetycznych jest zdefiniowanie dwóch rozłącznych przestrzeni: **przestrzeni poszukiwań** oraz **przestrzeni rozwiązań** [41]. Przestrzeń rozwiązań, nazywana też przestrzenią fenotypów, składa się z rzeczywistych rozwiązań problemu. Przestrzeń poszukiwań, nazywana też przestrzenią genotypów zawiera zakodowane rozwiązania. Każdy osobnik składa się z **genotypu** oraz odpowiadającego mu **fenotypu**. Fenotypy składają się najczęściej z pewnej liczby parametrów opisujących cechy rozwiązania, natomiast genotypy z zakodowanej wersji tych parametrów. Konkretna zakodowana cecha nazywana jest **genem**. Wartości, jakie dany gen może przyjmować nazywane są **allelami**.

Najprostszym algorytmem genetycznym jest tak zwany prosty algorytm genetyczny (Simple Genetic Algorithm – SGA).

Idea działania tego algorytmu wygląda następująco:

```
begin
Inicjalizuj_Populację(P);
while Nie_Zachodzi_Warunek_końca do
begin
  Oceń(P);
  Utwórz_Pustą_Populację(P');
  while Rozmiar(P')<Rozmiar(P) do
  begin
        Wybierz_Rodziców(p1, p2, P);      (wybiera dwóch osobników osobników populacji P)
        Krzyżuj(p1,p2,p1',p2',pc);        (krzyżuje osobniki p1i p2 z prawdopodobieństwem pc,
                                           w wyniku powstają osobniki p1' i p2' )
        Mutuj(p1',p2',p1'',p2'',pm);      (mutuje osobniki p1' i p2' z prawdopodobieństwem pm,
                                           w wyniku powstają osobniki p1'' i p2'' )
        Umieść_w_Populacji(p1'',p2'',P');
  end;
  P:=P';
end;
end;
```

Algorytm ten działa w sposób następujący. W pierwszym etapie generuje się genotyp każdego osobnika w populacji w sposób losowy. Kolejny etap jest główną pętlą algorytmu. Najpierw genotypy są przekształcane do postaci odpowiadających im fenotypów. Następnie otrzymane fenotypy poddawane są ocenie przez funkcję przystosowania i przypisywane są im wartości. Wartości te (przystosowania osobników) są wykorzystywane na etapie selekcji. Im wyższa jest wartość przystosowania danego osobnika, tym większe jest prawdopodobieństwo, że osobnik ten zostanie wybrany do reprodukcji. Z wybranych w ten sposób osobników generowane jest za pomocą operatora krzyżowania potomstwo. Tak powstałe dwa nowe osobniki poddawane są działaniu operatora mutacji. Operator ten stosowany jest z niewielkim prawdopodobieństwem, podobnie jak to się dzieje w naturze. Proces selekcji, krzyżowania i mutacji powtarzany jest do momentu wygenerowania tylu nowych osobników, ile było w populacji poprzedniej. Proces od momentu oceny fenotypów do momentu utworzenia nowej populacji nazywamy generacją. Proces ten jest powtarzany do otrzymania satysfakcjonującego rozwiązania lub do momentu osiągnięcia określonej liczby generacji.

3.3 Programowanie genetyczne

Cechą charakterystyczną programowania genetycznego jest przestrzeń poszukiwań, której elementami są programy. Programowanie genetyczne zostało zapoczątkowane przez J. R. Kozę, który zaczął stosować algorytmy genetyczne do automatycznego pisania programów w języku LISP. W programowaniu genetycznym nie ma rozróżnienia na przestrzeń genotypów oraz przestrzeń fenotypów, wszystkie operacje wykonywane są bezpośrednio na rozwiązaniach. Osobniki reprezentowane są w postaci drzew analizy składniowej danego programu (ang. parse tree). Obecnie termin „genetyczne programowanie" jest często używany jako synonim algorytmu ewolucyjnego z reprezentacją w postaci drzew. W związku z reprezentacją osobników jako drzew operator krzyżowania został zmodyfikowany. Najogólniej można zdefiniować krzyżowanie drzew jako zamianę poddrzew w strukturze przodków. Istnieje wiele różnych typów krzyżowania. Wśród nich można wyróżnić krzyżowanie jednopunktowe, krzyżowanie jednorodne, gładkie krzyżowanie jednorodne [66] oraz krzyżowanie polegające na przepinaniu poddrzew [73].

Zmodyfikowany został również sposób obliczania funkcji przystosowania. Ponieważ rozwiązaniami problemu są programy, muszą one zostać uruchomione, aby można było ocenić ich działanie. Przystosowanie programu zależy od tego, w jakim stopniu otrzymane przez niego wyniki dla ciągu wartości wejściowych zbliżone są do wyników pożądanych.

Algorytm programowania genetycznego wygląda następująco:

```
begin
Inicjalizuj_Populację(P);
while Nie_Zachodzi_Warunek_końca do
begin
  Uruchom_Programy_z_Populacji(P);
  Oceń(P);
   while Rozmiar(P')<Rozmiar(P) do
  begin
        Losowo_Wybierz_Operację_Genetyczną(pc ,pr) (wybiera losowo z prawdopodobieństwem
                                                    krzyżowania pc  i prawdopodobieństwem
                                                    reprodukcji pr, jaka operacja genetyczna
                                                    zostanie użyta)
        if Wybrane_Krzyżowanie then
```

```
        begin
            Wybierz_Rodziców(p1, p2, P);          (wybiera dwóch osobników
                                                   populacji P)
            Krzyżuj(p1,p2,p1',p2');               (krzyżuje osobniki p1 i p2, w wyniku
                                                   powstają osobniki p1' i p2' )
            Umieść_w_Populacji(p1',p2',P');
        end
        else
        begin
            Reprodukuj(p1,p1');                   (reprodukuje osobnika p1, w wyniku
                                                   powstaje osobnik p1' )
            Umieść_w_Populacji(p1',,P');
        end;
    end;
    P:=P';
end;
end;
```

Podsumowując, programowanie genetyczne polega na wykonaniu trzech następujących kroków:

1) wygenerowaniu populacji losowej,
2) dopóki nie zachodzi warunek końca, wykonywaniu iteracyjnie następujących etapów:
    a. Wykonanie każdego programu w populacji i przypisaniu mu wartości oszacowania zależnej od tego, jak dobrze rozwiązuje problem,
    b. Utworzeniu nowej populacji, stosując dwie główne operacje genetyczne. Operacje są wykonywane na osobnikach populacji wybranych losowo z prawdopodobieństwem zależnym od ich przystosowania.,
3) możliwości potraktowania najlepszego osobnika pojawiającego się w każdej populacji jako rezultatu programowania genetycznego i rozwiązania problemu.

Dodatkowo, oprócz dwóch głównych operatorów genetycznych, reprodukcji i krzyżowania, stosuje się również opcjonalnie operatory, takie jak: mutacja, permutacja, edycja, enkapsulacja.

Mutacja wprowadza losowe zmiany w strukturze populacji. Polega na losowym wybraniu węzła w drzewie, usunięcia całego jego poddrzewa i zastąpienia go losowym poddrzewem.
Permutacja polega na losowym wybraniu węzła wewnętrznego, który jest funkcją. Następnie argumenty tej funkcji zostają losowo permutowane. Jeżeli funkcja jest przemienna, to permutacja jej argumentów nie wpływa na zwracaną wartość.
Operacja edycji umożliwia edycję i uproszczenie wyrażenia w trakcie działanie programowania genetycznego. Operacja ta stosuje rekurencyjnie do każdego wyrażenia w populacji ustalony zbiór reguł niezależnych od dziedziny oraz specyficznych dla dziedziny. Uniwersalną regułą niezależną od dziedziny jest reguła następująca: jeżeli jakaś funkcja nie jest zależna kontekstowo a jej argumenty są prostymi atomami, to operacja edycji może wyliczyć jej wartość i zastąpić tą funkcję obliczoną wartością. Przykładem reguły specyficznej dla dziedziny może być reguła dla problemów numerycznych, która wstawia 0 wszędzie tam, gdzie podwyrażenie jest odejmowane od samego siebie.
Operacja enkapsulacji jest sposobem automatycznej identyfikacji użytecznych poddrzew i nadawania im nazw, do których można się później odwoływać.
Programowanie genetyczne jest używane w wielu dziedzinach, między innymi do automatycznego projektowania [51], do rozpoznawania wzorców [99] czy też do generowania obrazów [39].

3.4 Programowanie ewolucyjne

Programowanie ewolucyjne (EP) zostało wprowadzone przez L. J. Fogla [30] w 1960 roku jako strategia optymalizacji stochastycznej. Jest to metoda, którą stosuję się w przypadku, gdy inne techniki nie są możliwe do wykorzystania. Często stosuje się programowanie ewolucyjne w optymalizacji kombinatorycznej oraz optymalizacji funkcji rzeczywistych. Strategia ta jest podobna do algorytmów genetycznych, lecz główny nacisk położony jest na relacje pomiędzy przodkiem a jego potomkami, a nie – jak w przypadku algorytmów genetycznych - na naśladowanie operacji genetycznych występujących w naturze.
Prace, które dały początek programowaniu ewolucyjnemu dotyczyły badania populacji automatów skończonych, z których każdy miał za zadanie rozpoznawać pewien nieznany mu język. Metoda polegała na tym, że automaty były poddawane losowym perturbacjom i procesowi

selekcji. Po wielu cyklach perturbacji i selekcji powstawały automaty, które prawie bezbłędnie rozpoznawały nieznane wcześniej prawidłowe wyrażenia języka.

W latach 80-tych D.Fogel rozwinął programowanie ewolucyjne wprowadzając do niego nowe elementy, co doprowadziło do wzrostu zainteresowania tą techniką. Obecnie można wyróżnić trzy główne typy programowania ewolucyjnego: standardowe EP, meta-EP oraz Meta-EP. Typy te różnią się poziomem samoadaptacji zawartym w każdym z nich. Meta-Ep są dzisiaj uważane za standardowe i nazywane prostym programowaniem ewolucyjnym (ang. Simple EP) [4].

W programowaniu ewolucyjnym nie rozróżnia się przestrzeni rozwiązań i przestrzeni genotypów. Elementami populacji są wektory liczb rzeczywistych, z których każda jest parametrem rozwiązania. Parametr taki często jest nazywany **zmienną decyzyjną**. Poniżej przedstawiony został prosty algorytm programowania ewolucyjnego.

```
begin
Inicjalizuj_Populację(P);
while Nie_Zachodzi_Warunek_końca do
begin
  Oceń(P);
  Utwórz_Pustą_Populację(P');
  while Rozmiar(P')<Rozmiar(P) do
  begin
        Wybierz_Rodzica(p1,P);      (wybiera osobnika z populacji P)
        Mutuj(p1,p1');     (mutuje osobnika p1, w wyniku powstaje p1'
        Umieść_w_Populacji(p1',P');
end;
Oceń(P');
 P:=Wybierz_Połowę(P+P');
end;
end;
```

W przedstawionym algorytmie pierwszym etapem jest inicjacja populacji początkowej. Następnie wybiera się przodka – najczęściej używana jest selekcja turniejowa. W wyniku mutacji powstaje potomek, który wstawiany jest do populacji. Po podwojeniu się liczebności populacji wszystkie nowe osobniki poddawane są ocenie. Połowa osobników z najniższym przystosowaniem zostaje odrzucona. Pozostała połowa stanowi nową populację.

3.5 Strategie ewolucyjne

Strategie ewolucyjne (ang. evolutionary strategie, ES) powstały w Niemczech. Ich twórcy, I. Rechenberg oraz H. Schwefel [81] eksperymentowali nad optymalizacją urządzeń mechanicznych metodą permutowania pewnego początkowego rozwiązania. Otrzymali bardzo dobre wyniki, które były odmienne od tych otrzymywanych z zastosowaniem dotychczasowej metody projektowania. Dostrzegli w tej metodzie podobieństwo do procesu ewolucji i zachęceni wynikami, uogólnili ją na przypadek, gdy naraz przetwarzanych jest wiele rozwiązań - strategie ewolucyjne.

Podobnie jak w programowaniu genetycznym i programowaniu ewolucyjnym, nie ma rozróżnienia na przestrzeń fenotypów i genotypów. Każdy osobnik reprezentowany jest jako wektor parametrów będących liczbami zmiennoprzecinkowymi. Najprostszą formą strategii są strategie dwuelementowe –(1+1)-ES. W takich strategiach operuje się na dwóch osobnikach: rodzicu i potomku. Określony jest tylko jeden operator – mutacja. Potomek powstaje z rodzica w wyniku mutacji. Następnie poddawany jest on ocenie. Jeżeli jest lepszy od swojego rodzica to zastępuje go i poddawany jest kolejnym mutacjom. Natomiast, jeżeli potomek nie jest lepszy, to zostaje odrzucony, a kolejny osobnik powstaje w wyniku mutacji niezmienionego rodzica.

Strategia ta miała jednak wady – najważniejszą z nich jest fakt, że przeszukiwanie sąsiednich punktów naraża algorytm na osiągnięcie optimum lokalnego zamiast globalnego.
Obecnie stosuje się bardziej zaawansowane strategie ewolucyjne [6, 95, 96].

Podstawową zmianą jest zastąpienie jednego przodka populacją osobników. Istnieją dwa typy strategii ewolucyjnych z populacją: (μ,λ)-ES oraz $(\mu+\lambda)$-ES. Przypominają one częściowo algorytmy genetyczne, ponieważ operują na populacji osobników, stosują operatory rekombinacji i mutacji oraz selekcji zależnej od przystosowania osobników.
Przebieg typowej strategii ewolucyjnej wygląda następująco:

```
begin
Utwórz_Pustą_Populację(P);
Oceń(P);
while Nie_Zachodzi_Warunek_końca do
begin
  while Rozmiar(P')< λ do
```

```
  begin
        Wybierz_Rodziców(p1, p2, P);      (wybiera dwóch osobników osobników populacji P)
        Krzyżuj(p1,p2,p1',p2',pc);        (krzyżuje osobniki p1 i p2 z prawdopodobieństwem pc,
                                           w wyniku powstają osobniki p1' i p2' )
        Mutuj(p1',p2',p1'',p2'',pm);      (mutuje osobniki p1' i p2' z prawdopodobieństwem pm,
                                           w wyniku powstają osobniki p1'' i p2'' )
        Umieść_w_Populacji(p1'',p2'',P');
  end;
Oceń(P');
P:=Wybierz_Populację(P');                 (w strategiach (μ,λ) )
lub
P:=Wybierz_Populację(P+P');               (w strategiach (μ+λ))
end;
end;
```

Pierwsza populacja jest generowana w sposób losowy, albo powstaje w wyniku mutacji jednego, podanego przez użytkownika rozwiązania. Każdy osobnik populacji składa się z dwóch wektorów: wektora parametrów $X = (x_1,...,x_n)$ opisujących rozwiązywany problem oraz wektora parametrów strategii $S = (s_1,...,s_n)$. Parametry strategii określają zakres mutacji odpowiadających ich parametrów problemu. Najczęściej na etapie mutacji parametru x_i używa się rozkładu normalnego Gaussa z zakresu (0, s_i), gdzie s_i jest odchyleniem standardowym. Następnie z populacji początkowej wybiera się losowo dwóch rodziców. W wyniku krzyżowania powstaje dwóch nowych potomków, którzy następnie poddawani są mutacji. Operacja ta jest powtarzana do momentu otrzymania λ potomków. Kolejnym etapem jest ocena osobników populacji i wybranie μ najlepszych jako nowej populacji. W przypadku strategii $(\mu + \lambda)$ nowa populacja wybierana jest z populacji przodków i potomków, w strategiach (μ, λ) - tylko z nowych osobników.

3.6 Hierarchia w algorytmach genetycznych

Struktury hierarchiczne bardzo często są używane jako reprezentacja obiektów w wielu algorytmach ewolucyjnych [91]. Już w 1987 roku Wilson pisał o znaczeniu hierarchii w reprezentacji zadań i podzadań [111]. W tym samym roku A. S. Bickel oraz R.W. Bickel używali drzew w swoim systemie [12]. Również programowanie genetyczne często bazuje na drzewach [51, 54].

W projektowaniu wspomaganym komputerowo bardzo często są używane struktury hierarchiczne do reprezentowania brył. Jedną z takich reprezentacji jest reprezentacja z genotypem przedstawionym jako graf hierarchiczny [38, 98]. Genetyczne operatory dla reprezentacji grafowej są znacznie bardziej złożone niż dla reprezentacji binarnej. Reprezentacja ta ma jednak wielką zaletę - umożliwia kodowanie relacji pomiędzy komponentami artefaktu. Inną używaną reprezentacją jest reprezentacja za pomocą hierarchicznego chromosomu [10]. Jest ona znacznie łatwiejsza do zaimplementowania niż reprezentacja grafowa, nie zawiera jednak bezpośrednich informacji o relacjach pomiędzy komponentami artefaktu. Zaletą algorytmu opartego na hierarchicznym chromosomie jest możliwość zainicjowania losowej populacji początkowej.

Istnieje również grupa algorytmów, w których hierarchia nie dotyczy struktury obiektów, lecz dokładności w globalnym poszukiwaniu genetycznym. Przykładem może być kodowanie delta wprowadzające hierarchię kodów binarnych, która umożliwia intensywniejsze poszukiwania wokół najlepiej przystosowanych osobników. Strategia ta została rozszerzona do strategii wielopopulacyjnej opartej na schemacie wyspowym. Hierarchia poszukiwań jest wykorzystywana również w DPE (ang. Dynamic Parametr Encoding). Kolejnym przykładem może być HGS (ang. Hierarchical Genetic Strategy) wprowadzona przez J. Kołodziej i R. Schaefera [91]. Główną ideą hierarchicznych strategii genetycznych jest równoległe uruchomienie zbioru zależnych procesów ewolucyjnych. Zależność między procesami ma strukturę drzewa. Procesy niższego rzędu (znajdujące się bliżej korzenia struktury) reprezentują chaotyczne poszukiwania z małą dokładnością. Znajdują obiecujące regiony, w których można uruchomić dokładniejszy proces wyższego rzędu. Populacje ewaluujące w różnych procesach mogą zawierać osobniki, które reprezentują fenotypy z różną dokładnością. Precyzja może być otrzymana poprzez binarne genotypy o różnej długości (proste HGS) lub też przez różne

skalowanie fenotypów (GS-FP). Wyniki przedstawione w pracy [109] porównujące strategie genetyczne opisane przez Gordona i Whitleya [109] z HGS-FP pokazują, że jest to najlepsza strategia dla wszystkich zadanych funkcji testowych. Wydajność algorytmu związana jest z równoczesnym szukaniem w przestrzeni poszukiwań przez wiele małych populacji.

W rozdziale opisane zostały dokładniej algorytmy oparte na strukturach hierarchicznych, które są używane w projektowaniu wspomaganym komputerowo [91, 98].

3.6.1 Algorytm genetyczny z hierarchicznym chromosomem

Reprezentacja obiektów jest jednym z najważniejszych elementów systemu projektowania. W pracy, ze względu na dążenie do minimalizacji liczby parametrów, zastosowana została reprezentacja CSC, opisana w Rozd. 3. W naszej reprezentacji każdy artefakt złożony jest z pewnej liczby składowych, każda składowa jest zdefiniowana przez dziewięć parametrów. Wartości parametrów każdej składowej są zakodowane binarnie. Każdy parametr jest ciągiem binarnym o długości q, gdzie pierwszy wyraz ciągu będzie oznaczał znak (1 dla liczb ujemnych, 0 dla dodatnich), następne q_i wyrazów ciągu to zakodowana część całkowita parametru, a ostatnich q_j wyrazów to część ułamkowa [10].
Początkowo w pracach Bentley'a [10] (dotyczących optymalizacji obiektów trójwymiarowych, opisanych reprezentacją CSC, za pomocą algorytmu genetycznego) genotypy były ciągami binarnymi. Ze względu na zmienną liczbę składowych artefaktu, były to ciągi o różnej długości.
Stosując krzyżowanie dla rodziców postaci:

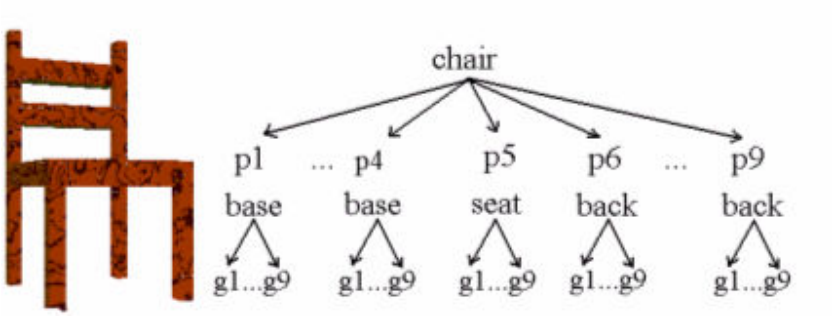
Rodzic_1 a b c d e f g h i j k l m n o p q r
Rodzic_2 A B C D E F G H I - - - - - - - - -

przy losowo wybranym punkcie krzyżowania, na przykład jako czternasty wyraz ciągu,
otrzymalibyśmy potomków postaci:

Potomek_1 a b c d e f g h i j k l m n - - - -
Potomek_2 A B C D E F G H I - - - - - o p r s

Zarówno Potomek$_1$ jak i Potomek$_2$ mają tylko częściowo określony drugi prymityw i nie mogą zostać odkodowani. Aby nie dochodziło do takich sytuacji, ewolucyjnym systemom projektowania stawia się wymagania, żeby tylko kompatybilne grupy alleli mogły się ze sobą krzyżować.

Bentle'y zastosował zamiast tradycyjnego sposobu określania, do którego genu należy dany allel, tak zwaną semantyczną hierarchię genotypu, reprezentowaną za pomocą hierarchicznego chromosomu. Na Rys.3.2 przedstawiony został przykładowy artefakt oraz odpowiadający mu hierarchiczny chromosom.



Rys. 3.2 Krzesło oraz odpowiadający mu hierarchiczny chromosom.

Kodując fenotyp jako hierarchiczny chromosom zdefiniowano odpowiedniki klasycznej mutacji i krzyżowania jednopunktowego: mutacje hierarchiczną oraz krzyżowanie hierarchiczne.

Mutacja jest używana w algorytmach genetycznych do zmiany alleli w celu odkrycia nowych rozwiązań bliskich aktualnemu rozwiązaniu w przestrzeni. Wyróżnione zostały dwa rodzaje mutacji hierarchicznej: mutacje pojedynczych alleli oraz mutacje grup alleli.

Mutacja pojedynczych alleli odpowiada klasycznej mutacji dla ciągów binarnych.

Algorytm mutacji pojedynczych alleli wygląda następująco:

- losowo wybierz geny w hierarchicznym chromosomie,
- zamień 0 na 1 lub 1 na 0.

Mutacja grupy alleli (czyli zakodowanej składowej) pozwala dodać lub usunąć z fenotypu całe składowe i w konsekwencji jest odpowiedzialna za zmianę długości genotypu.

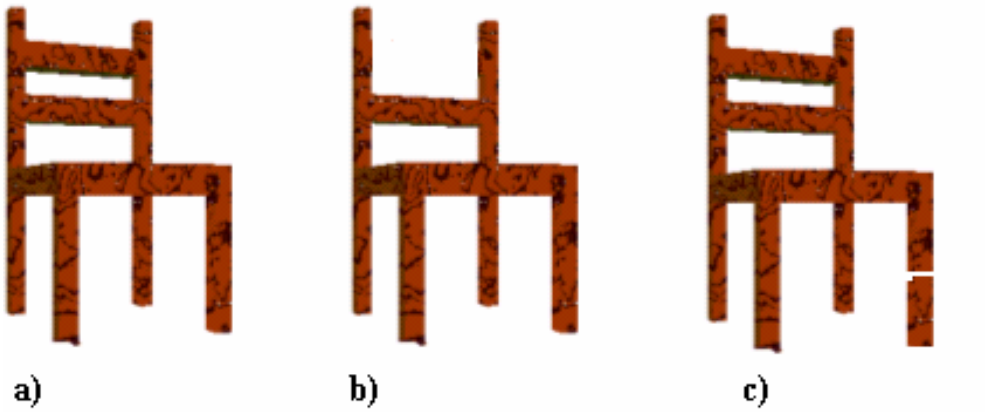
Algorytm mutacji grup alleli wygląda następująco:

- losowo wybierz grupę genów (składową) w hierarchicznym chromosomie,
- losowo zadecyduj, czy usuwamy czy dzielimy wybraną składową ,
- jeżeli usuwamy :

  usuń całą grupę genów i korespondujące z nimi allele z chromosomu,
- jeżeli dzielimy:

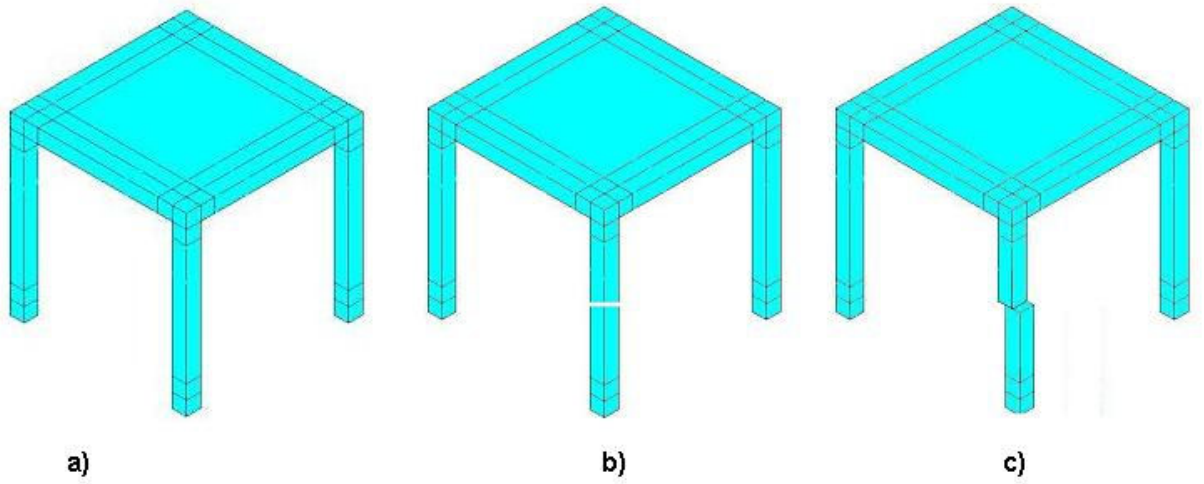
  - losowo wybierz kierunek podziału zakodowanej składowej,
  - oblicz nową wartość dla każdego allela z aktualnej składowej oraz każdego allela , z nowopowstałej składowej a następnie dodaj nową grupę alleli do chromosomu.

Rys. 3.3. a) przedstawia przykładowy fenotyp przed mutacją. Rys. 3.3. b) przedstawia fenotyp z Rys. 3.3. a) po usunięciu składowej, natomiast Rys 3.3. c) fenotyp po podzieleniu składowej.



Rys. 3.3 Przykładowe fenotypy przed mutacją i po mutacji.

Mutacja fenotypu, który we wcześniejszej epoce uległ mutacji polegającej na podzieleniu składowej, może spowodować np. przesunięcie nowopowstałej składowej.
Rys. 3.4. a) przedstawia przykładowy fenotyp przed mutacją. Rys. 3.4. b) przedstawia fenotyp z Rys. 3.4. a) po podzieleniu składowej, natomiast Rys 3.4. c) fenotyp z Rys. 3.4.b) po mutacji alleli.



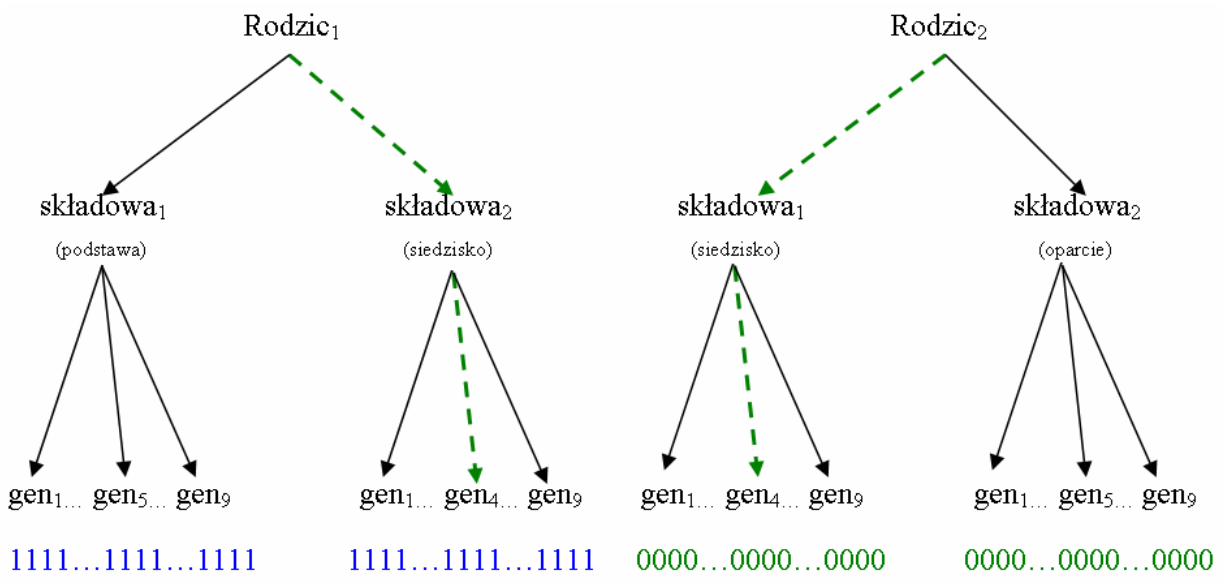
Rys. 3.4 Przykładowe fenotypy przed mutacją i po mutacji.

Drugim użytym operatorem genetycznym jest krzyżowanie hierarchiczne [7, 8]. Podobnie jak klasyczne krzyżowanie jest to proces dwuetapowy, polegający na:

1. znalezieniu odpowiedniego punktu krzyżowania u rodziców,
2. zastosowaniu krzyżowania do wygenerowania dzieci.

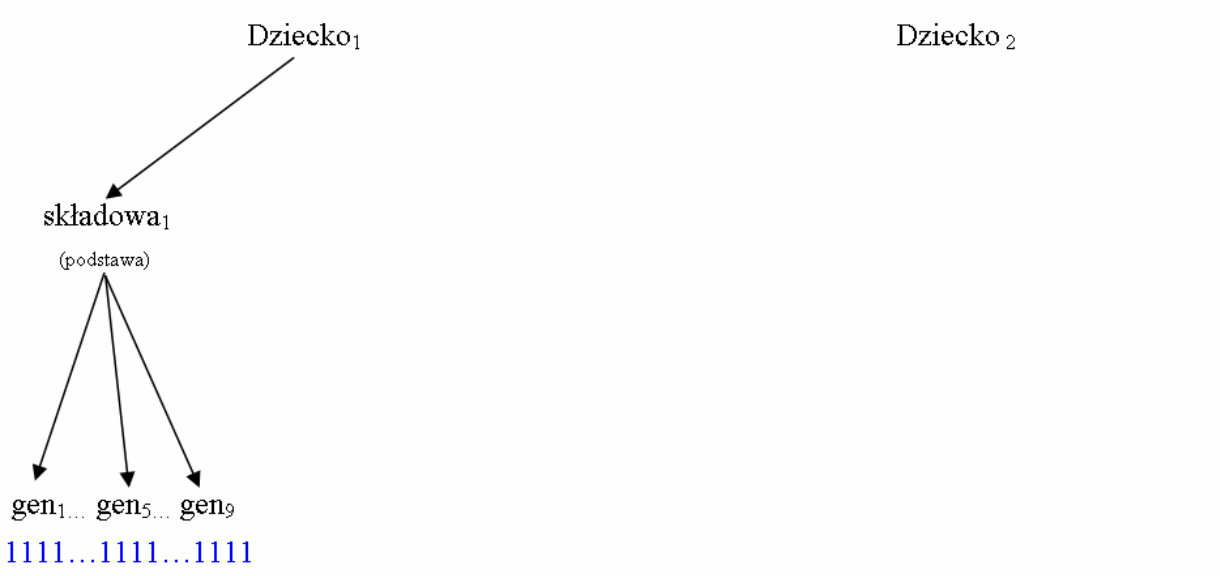
Przy tradycyjnym krzyżowaniu jednopunktowym etap pierwszy polega na losowym wybraniu numery bitu – punktu przecięcia. Dla krzyżowania hierarchicznego etap ten jest bardziej złożony. Punkt krzyżowania jest tutaj czwórką (P_1, P_2, g, b), gdzie P_1 jest numerem składowej u osobnika pierwszego, P_2 jest numerem składowej u osobnika drugiego, g jest numerem genu, b jest numerem bitu. Najpierw wybierany jest losowo wierzchołek P_1 u osobnika pierwszego na poziomie pierwszym, a następnie wyszukiwany jest u drugiego osobnika odpowiadający mu wierzchołek P_2 na tym samym poziomie. Następnie wybieramy losowo numer genu oraz numer bitu, które są wspólne dla obydwóch osobników. W ten sposób mamy już znaleziony punkt krzyżowania.

Rys. 3.5 przedstawia dwa osobniki rodzicielskie z wybranym punktem krzyżowania.



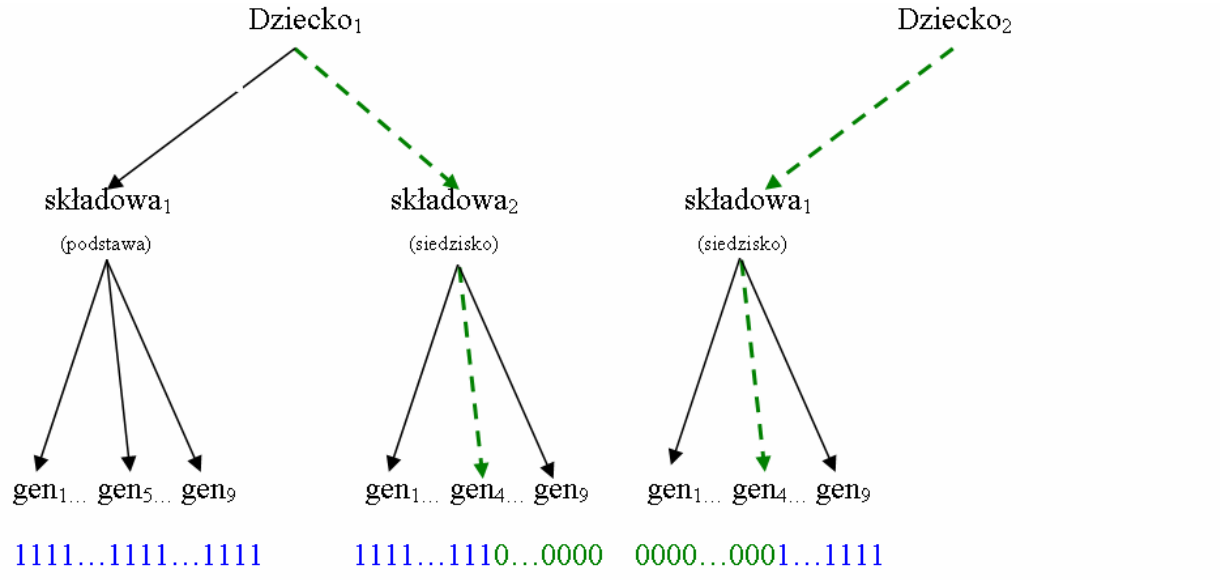
Rys.3.5 Osobniki rodzicielskie z wybranym punktem krzyżowania.

Gdy punkt krzyżowania jest już wybrany, może się rozpocząć proces krzyżowania. Algorytm startuje od korzenia każdego osobnika. W określonym porządku bierzemy wierzchołek na pierwszym poziomie u osobnika pierwszego. Następnie odpowiadający wierzchołek jest znajdowany u osobnika drugiego (jeśli istnieje). Jeżeli te wierzchołki są różne od wierzchołków P_1, P_2 znalezionych wcześniej, to są kopiowane od obydwóch rodziców do dzieci. Kopiujemy odpowiednio od rodzica pierwszego do dziecka pierwszego i od rodzica drugiego do dziecka drugiego. Należy zauważyć, że słowo wierzchołek ma tu abstrakcyjne znaczenie - obejmuje wszystko, od pojedynczego allela do grup alleli reprezentujących składowe. Jeżeli wierzchołek grupy jest kopiowany od rodzica do dziecka, to wszystkie geny w tej grupie i wszystkie allele dla tych genów są również kopiowane (kopiujemy wtedy całe poddrzewa). Proces kopiowania trwa tak długo, dopóki wszystkie wierzchołki rodzica na danym poziomie, oprócz wierzchołka znajdującego się na liście punktów podobieństwa, nie są skopiowane.



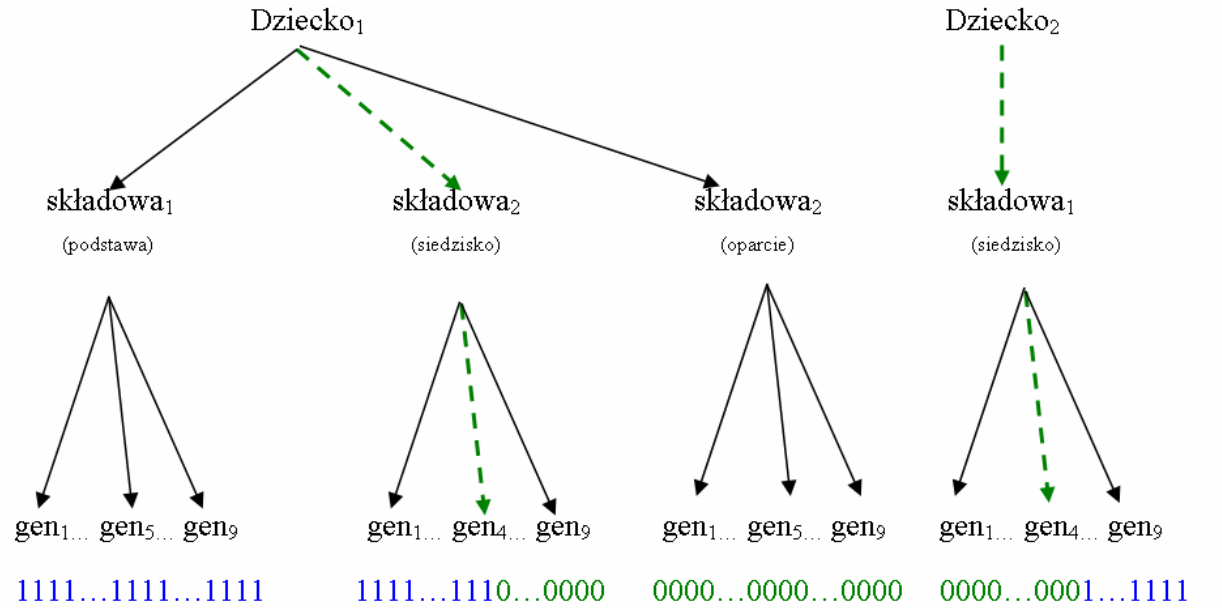
Rys.3.6 Dzieci w trakcie procesu krzyżowania.

Następnie algorytm powtarza proces kopiowania wierzchołków na niższym poziomie aż dojdzie do wierzchołka *g*. Dla wierzchołka *g* używane jest krzyżowanie jednopunktowe z punktem krzyżowania *b*.



Rys.3.7 Dzieci w trakcie procesu krzyżowania.

W końcowym kroku etapu drugiego wszystkie wierzchołki na poziomie genów występujące na prawo od wierzchołka *g*, których przodkami są wierzchołki P_1 lub P_2 są kopiowane odpowiednio od rodzica pierwszego do dziecka drugiego i od rodzica drugiego do dziecka pierwszego. Następnie wszystkie wierzchołki na poziomie składowych występujące u rodzica pierwszego na prawo od wierzchołka P_1 są kopiowane do dziecka drugiego, a wierzchołki występujące u rodzica drugiego na prawo od wierzchołka P_2 są kopiowane do dziecka pierwszego.



Rys.3.8 Dzieci powstałe w wyniku krzyżowania hierarchicznego.

Jak widać, hierarchiczne krzyżowanie rozwiązuje problem generowania nowych chromosomów z rodziców o różnej długości.

3.6.2 Algorytm genetyczny z grafem hierarchicznym

Reprezentacja grafowa jest stosowana często w projektowaniu wspomaganym komputerowo, niewiele jest jednak prac będących połączeniem takiej reprezentacji z algorytmami ewolucyjnymi. W pracy [98] przedstawiona została jedna z odmian grafów hierarchicznych opartych o grafy kompozycyjne. Zastosowanie reprezentacji grafowej jako metody kodowania rozwiązań w algorytmie ewolucyjnym wymagało nie tylko zdefiniowania grafów, które mogą

być wykorzystane w takim systemie, ale również opracowania odpowiedników standardowych operatorów genetycznych zdolnych wykonywać operacje na strukturach grafowych.

Grafy hierarchiczne (HG) są jedną z odmian grafów kompozycyjnych (CP-grafów) [36, 37]. Każda składowa projektu jest kodowana jako wierzchołek. Krawędzie między wierzchołkami kodują relacje pomiędzy składowymi reprezentowanymi przez te wierzchołki. Relacje takie mogą oznaczać przyleganie prymitywów lub odległość między nimi. Krawędzie nie są skierowane. Wierzchołki w grafie hierarchicznym mają wiązania. Wiązanie te sygnalizują potencjalne połączenia danego wierzchołka z innymi wierzchołkami.

Poniżej przestawiona najważniejsze definicje dotyczące grafów hierarchicznych.

Definicja 3.1 Wiązaniem *b* wierzchołka grafu nazywamy parę $(i, id) \in N \times N$, gdzie *i* jest identyfikatorem wierzchołka zawierającego to wiązanie, a *id* jest identyfikatorem wiązania.

□

Definicja 3.2 Wierzchołkiem hierarchicznym *v* nazywamy trójkę (i, B, C), gdzie:

- *i* jest identyfikatorem wierzchołka, $i \in N$,
- *B* jest zbiorem wiązań wierzchołka *v*, $B \subset \{i\} \times N$,

C jest zbiorem wierzchołków hierarchicznych zwanych **bezpośrednimi potomkami wierzchołka** *v*.

□

Dla danego wierzchołka $v=(i, B, C)$ przez i_v, B_v, C_v oznaczone będą odpowiednio identyfikator, zbiór wiązań oraz zbiór bezpośrednich potomków wierzchołka *v*. Przez v_i oznaczony będzie wierzchołek z identyfikatorem *i*. Przez *v(b)* oznaczone będzie wiązanie *b* wierzchołka *v*.

Definicja 3.3 Wierzchołek hierarchiczny nazywamy **wierzchołkiem prostym** (bez potomków), jeżeli zbiór jego potomków jest pusty.

□

Definicja 3.4 Krawędzią *e* pomiędzy wierzchołkami *v* i *w* nazywamy zbiór dwuelementowy $\{\{v(b_i), w(b_j)\}\}$ gdzie $v(b_i)$ jest wiązaniem wierzchołka *v* a $w(b_j)$ jest wiązaniem wierzchołka *w*.

Dla danej krawędzi *e* jej wiązania będą oznaczane przez b_e i b_e'. Krawędzie mogą łączyć zarówno wierzchołki hierarchiczne jak i wierzchołki potomne tego samego wierzchołka.

□

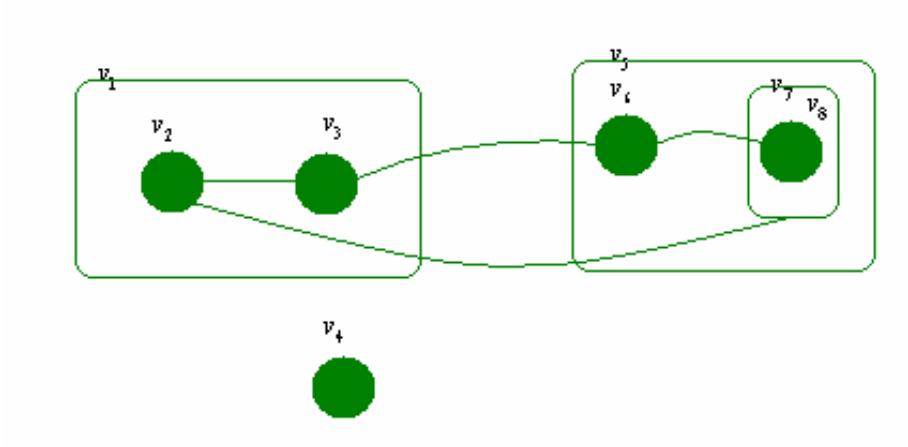
Definicja 3.5 Grafem hierarchicznym G nazywamy parę (*V*, *E*), gdzie *V* jest zbiorem wierzchołków, *E* jest zbiorem krawędzi oraz spełnione są następujące warunki:

- identyfikatory wierzchołków są unikalne,
- do każdego wiązania może dochodzić tylko jedna krawędź,
- każdy wierzchołek w grafie posiada co najwyżej jednego przodka,

nie istnieją krawędzie pomiędzy przodkiem i potomkiem.

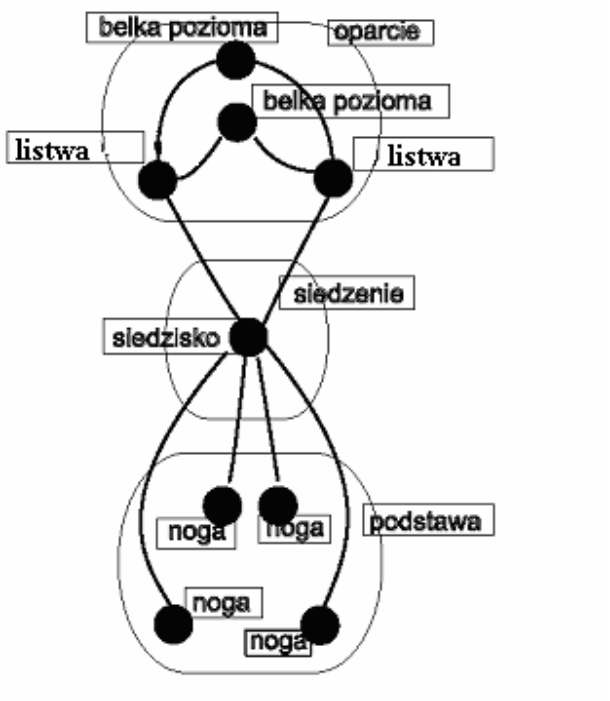
□

Na Rys. 3.9 przedstawione zostały przykładowe wierzchołki hierarchiczne oraz łączące je krawędzie. Potomkami wierzchołka v_1 są wierzchołki v_2 oraz v_3. Potomkami wierzchołka v_5 są wierzchołki v_6, v_7 oraz v_8. Potomkiem wierzchołka v_7 jest wierzchołek v_8.



Rys.3.9 Graf hierarchiczny.

Dodatkowo, zarówno wierzchołki jak i krawędzie są etykietowane. Na Rys. 3.10 przedstawiony został przykład etykietowanego grafu hierarchicznego.



Rys.3.10 Etykietowany graf hierarchiczny.

4 Teoria algorytmów genetycznych

Jednym z najważniejszych pytań, na które teoria algorytmów ewolucyjnych chce znaleźć odpowiedź, jest pytanie, jak możemy scharakteryzować, zbadać i przewidzieć takie poszukiwanie. W zależności od różnych podejść, możemy wyróżnić kilka głównych teoretycznych nurtów w algorytmach genetycznych: mikroskopowe modele systemów dynamicznych, krajobrazy przystosowania, analiza komponentów, teoria schematów czy NFS (ang. No Free Lunch Theorems)

W rozdziale tym przedstawione zostaną istniejące teorie algorytmów genetycznych. Dokładniej zostaną omówione dwa podejścia, które w tej pracy zostały wykorzystane: mikroskopowe modele systemów dynamicznych, czyli modele markowowskie oraz teoria schematów.

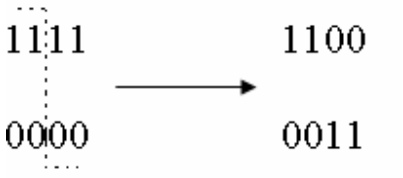
4.1 Krajobrazy przystosowania

Najprostsza wersja tego podejścia może być rozumiana jako wykres, gdzie współrzędna pozioma każdego punktu reprezentuje wszystkie geny osobnika (genotyp) odpowiadające temu punktowi. Natomiast przystosowanie osobnika jest umieszczone na osi pionowej. Jeżeli genotyp może być wizualizowany w dwóch wymiarach, wykres będzie trójwymiarowy i będzie zawierał wzgórza i doliny. Duże obszary przystosowania o małych wartościach mogą być rozważane jako depresje, podczas gdy na duże obszary o podobnych, dużych wartościach przystosowania możemy patrzeć jak na płaskowyż. Techniki poszukiwań szukają najwyższego punktu – będzie to odpowiadało genotypowi o najwyższej wartości przystosowania, który będzie optymalnym rozwiązaniem.

4.2 Analiza komponentów

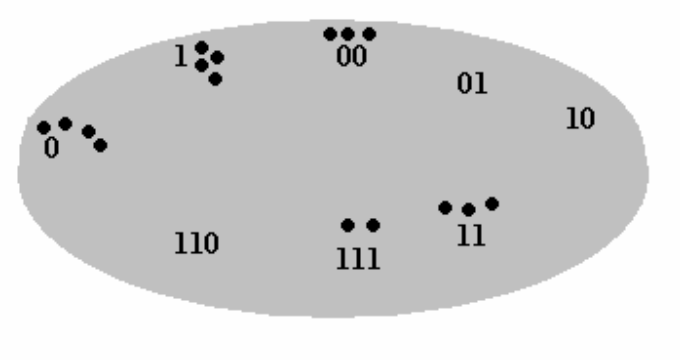
Analiza komponentów koncentruje się na tym, jak geny lub ich kombinacje rozprzestrzeniają się. Podejście to skupia się na propagacji podkomponentów osobników. W algorytmach genetycznych z reprezentacją binarną analiza komponentów rozważa, co się dzieje wewnątrz ciągów binarnych, koncentrując się na pojedynczych bitach lub ich grupach.

Rozważmy przykładową populację zawierającą tylko dwa ciągi binarne: 0000 i 1111. Na Rys. 4.1 przedstawione zostały wyniki krzyżowania osobników 0000 i 1111 w ich środkowym punkcie.



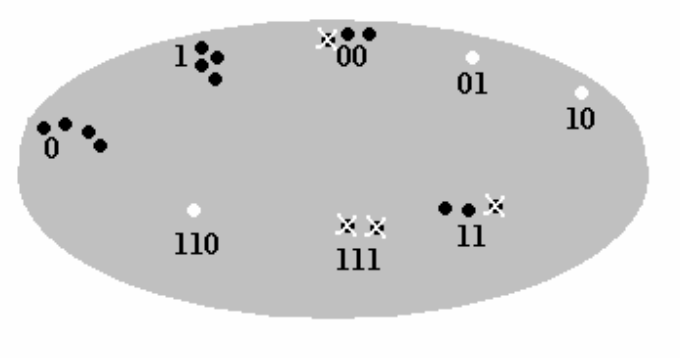
Rys.4.1 Krzyżowanie genotypów 1111 i 0000.

Analiza komponentów nie koncentruje się na wpływie operacji np. krzyżowania na osobnika, lecz na wpływie tej operacji na wzorce bitów. Na przykład przed krzyżowaniem populacja zwiera trzy instancje wzorca bitów 00, tak jak jest to pokazane na Rys. 4.2. (Ciąg bitów 0000 zawiera wzorzec bitów 00 na trzy różne sposoby, używając bitów 1 i 2, 2 i 3 oraz 3 i 4).



Rys.4.2 Wzorce bitów z zaznaczoną liczbą wystąpień w populacji złożonej z osobników 1111 i 0000.

Po krzyżowaniu populacja zwiera dwie instancje wzorca bitów 00. Usunięcie jednej instancji jest pokazane za pomocą krzyżyka na Rys. 4.3. Jednocześnie ta sama operacja krzyżowania tworzy inne komponenty. Przykładem mogą być wzorce bitów 01 i 10, które są reprezentowane przez białe kropki na Rys. 4.3.



Rys.4.3 Efekty krzyżowania ciągów binarnych 0000 i 1111 na wzorcach bitów.

W programowaniu genetycznym komponentami mogą być pojedyncze składowe, całe wyrażenia lub grupy wyrażeń. Przykładem mogą być tu rozważania, dotyczące liczy kopii konkretnego komponentu programu występującego w populacji w poszczególnych epokach.

4.3 NFL

Teoria NIL (ang. No Free Lunch Theorems) to seria rezultatów, które dowodzą, że żaden algorytm poszukiwań nie może być uznany za lepszy od innego algorytmu dla wszystkich możliwych problemów. Konsekwencją tej teorii jest fakt, że jeżeli jakiś algorytm, dla pewnej klasy problemów jest lepszy, dla innej klasy problemów będzie gorszy [112].

4.4 Markowowski model algorytmu genetycznego

Pierwszymi pracami dotyczącymi modelowania algorytmu genetycznego za pomocą łańcucha Markowa są prace M. Vose'a oraz A. Nix'a [61, 101, 102, 103] dotyczące prostego algorytmu genetycznego. Następnie model ten został rozszerzony przez G. Koehler'a i M. Vose'a na przypadek algorytmu z kodowaniem za pomocą ciągów liczb w innym systemie niż binarny [49]. Modelowanie algorytmu genetycznego za pomocą jednorodnego łańcucha markowa umożliwia efektywne obliczenie macierzy przejścia, badanie zbieżności algorytmu oraz jego własności asymptotycznych. Niestety – jak do tej pory – niewiele algorytmów ewolucyjnych udało się zamodelować w ten sposób.

W pracy [76] R. Poli, J.E. Rove oraz N. F. McPhee stworzyli model markowowski dla programowania genetycznego z homologicznym krzyżowaniem, co umożliwiło obliczenie macierzy przejścia algorytmu. Jednak żadnych innych wyników teoretycznych nie udało się w ten sposób otrzymać.

Szczegółowy opis modelu markowowskiego znajduje się pracach M. Vose'a [61, 101,102] oraz w Rozd. 5 i 6 rozprawy doktorskiej. Podstawowe definicje i twierdzenia dotyczące łańcuchów Markowa [25, 46, 50, 72, 105] przedstawione zostały w Rozd. 4.4.1.

4.4.1 Podstawy teorii łańcuchów Markowa

Definicja 4.1 Przestrzenią probabilistyczną nazywamy trójkę (Ω,Σ,P), gdzie Ω jest przestrzenią zdarzeń elementarnych, Σ jest δ-ciałem przeliczalnie addytywnym podzbiorów zbioru Ω, a P jest miarą określoną nad Σ.

□

Definicja 4.2 Zbiór S nazywamy **zbiorem skierowanym** jeśli S jest częściowo uporządkowany i spełniona jest następującą własność:

$\forall k \in N\ \forall \{s_1,...s_k\} \subset S\ \exists s \in S : \forall i = 1..k\ s_i < s$

□

Niech (Ω,Σ,P) będzie przestrzenia probabilistyczną i niech S będzie zbiorem skierowanym

Definicja 4.3 Procesem stochastycznym nazywamy dowolną funkcję $X:\Omega\times S\rightarrow R^n$ spełniającą warunek:

dla każdego $s \in S$ $X(\cdot,s)$ jest mierzalna

□

Proces oznaczamy symbolem $\{X_t \equiv X(\cdot,t), t \in S\}$.

Zbiór S nazywamy przestrzenią chwil czasowych. Na ogół S jest podzbiorem zbioru liczb rzeczywistych.

Jeśli $S = N$ to mówimy, że $\{X_t\}$ jest procesem z czasem dyskretnym.

Jeśli $S = R$ to mówimy, że $\{X_t\}$ jest procesem z czasem ciągłym.

Szczególnym przypadkiem procesu jest łańcuch Markowa.

Definicja 4.4 Proces $\{X_t:t\in N\}$ nazywamy **łańcuchem Markowa** jeśli spełnione są warunki:

- zbiór $\bigcup_{n=0}^{\infty} X_n(\Omega) = \{x_1, x_2, ...\}$ jest co najwyżej przeliczalny,
- dla każdego $n \in N, j \in N, m \in N, m<n,$

  dla każdego $j_1 < j_2 < ... < j_m < n$,

  dla każdego $i_1 ... i_m \in N$ zachodzi

  $P(X_n = x_j \mid X_{j_m} = x_{i_m}, ..., X_{j_1} = x_{i_1}) = P(X_n = x_j \mid X_{j_m} = x_{i_m})$.

□

Zbiór $\bigcup_{n=0}^{\infty} X_n(\Omega) = \{x_1, x_2, ...\}$ nazywamy **zbiorem stanów**.

Prawdopodobieństwo $P(X_n = x_j \mid X_{n-1} = x_i)$ oznaczamy przez p_{ij}^n i nazywamy prawdopodobieństwem przejścia ze stanu x_i do stanu x_j.

Łańcuch Markowa nazywamy jednorodnym, jeżeli prawdopodobieństwo p_{ij}^n przejścia ze stanu x_i do stanu x_j nie zależy od chwili czasowej *n*. Formalna definicja łańcucha jednorodnego została przedstawiona poniżej.

Definicja 4.5 Jednorodnym łańcuchem Markowa nazywamy łańcuch Markowa $\{X_t:t\in N\}$, który spełnia warunek przesunięcia:

$$P(X_m = x_j \mid X_{m-1} = x_i) = P(X_n = x_j \mid X_{n-1} = x_i) \qquad 4.1.$$

□

Prawdopodobieństwa przejść o co najwyżej przeliczalnej liczbie stanów można ułożyć w macierz $M^n = p_{ij}^n$

W przypadku, gdy ciąg $\{X_t:t\in N\}$ jest jednorodnym łańcuchem Markowa mamy:

$$p_{ij}(n) = P(X_n = x_j \mid X_0 = x_i) \qquad 4.2.$$

Definicja 4.6 Mówimy, że **stan** x_j **jest osiągalny ze stanu** x_i wtedy i tylko wtedy, gdy istnieje takie $n \in N, n \geq 1$ że $p_{ij}(n) > 0$.

□

Definicja 4.7 Mówimy, że **stan** x_j **komunikuje się ze stanem** x_i wtedy i tylko wtedy, gdy zarówno **stan** x_j jest osiągalny ze stanu x_i jak i stan x_i jest osiągalny ze stanu x_j.

□

Definicja 4.8 Klasą stanów nazywamy największy podzbiór przestrzeni stanów taki, że dowolne jego dwa stany komunikują się. Przez największy podzbiór rozumiemy taki zbiór, dla którego nie jest możliwe dodanie stanu komunikującego się ze stanami tego zbioru.

□

Uwaga

Dwie różne klasy muszą być rozłączne, ponieważ istnienie elementu wspólnego powodowałoby komunikowanie się (przez ten element) pomiędzy ich stanami, co z kolei powodowałoby połączenie się tych klas w jedną.

Stany jednorodnego łańcucha Markowa można pogrupować w klasy w zależności od tego, czy mogą one być osiągane z dowolnego danego stanu.

Definicja 4.9 Podzbiór *M* przestrzeni stanów nazywamy zamkniętym wtedy i tylko wtedy, gdy $\sum_{j\in M} p_{ij} = 1$ dla każdego $i \in M$.

□

Definicja 4. 10 Jednorodny łańcuch Markowa $\{X_t : t \in N\}$, nazywamy nieprzywiedlnym, jeśli zbiór stanów tworzy zamknięta klasę stanów komunikujących się.

□

Definicja 4. 11 Stan x_i nazywamy **chwilowym,** jeśli istnieje takie $k \in N$ oraz stan x_j taki, że $p_{ij}(k) > 0$ oraz dla każdego $m \in N \quad p_{ji}(m) = 0$.

□

Definicja 4.12 Stan x_i nazywamy **istotnym,** jeśli dla każdego x_j takiego, że istnieje $k_j \in N$ takie, że $p_{ij}(k_j) > 0$ to istnieje $m_j \in N$ takie, że $p_{ji}(m_j) > 0$.

□

Definicja 4.13 **Stan** istotny x_i nazywamy **okresowym**, jeśli istnieje $d > 1, d \in N$, takie, że $p_{ii}(n) = 0$ dla każdego $n \in N$ nie będącego wielokrotnością *d*.

□

Twierdzenie 4.1 (Ergodyczne)

Założenia:

- Niech $\{X_t, t \in N\}$ będzie jednorodnym łańcuchem Markowa o skończonej liczbie stanów $x_1, x_2 \ldots x_s$
- Niech $M_1 = (p_{ij})$ będzie jego macierzą przejścia

Teza:

Jeśli istnieje liczba kroków *r* większa od zera taka, że w macierzy *M(r)* istnieje $1 \le \mathbf{s_1} \le s$ kolumn niezerowych (istnieje minimum $\min_{1 \le i \le s_1} p_{ij}(r) > \delta > 0$)

to spełnione są następujące warunki:

1. $\exists \lim_{n \to \infty} p_{ij}(n) = p_j,\ j = 1,\ldots,s$ (granica nie zależy od *i*)

   gdzie $p_j \ge \delta > 0$ dla $j = q_1, q_2, \ldots q_{s_1}$,

2. $| p_{ij}(n) - p_j | \le (1 - s_1\delta)^{\left\lfloor \frac{n}{r} - 1 \right\rfloor}$.

p_j nazywamy prawdopodobieństwem ergodycznym.

Wniosek 4.1

Przestrzeń stanów stanowi klasę stanów istotnych.

4.4.2 Markowowska teoria algorytmów genetycznych

Niech *U* oznacza uniwersum genetyczne stanowiące zbiór genotypów. Przestrzenią stanów ε algorytmu genetycznego nazywać będziemy zbiór, do którego należą wszystkie populacje (lub ich jednoznaczne reprezentacje), jakie może wytworzyć ten algorytm.

Algorytm genetyczny możemy modelować jako system transformujący populacje z przestrzeni stanów w przestrzeń stanów w kolejnych epokach genetycznych *t=0,1,....* Generowany w ten sposób ciąg populacji P_0, P_1, ... tworzy proces stochastyczny z dyskretnym czasem. Przy założeniu, że operatory genetyczne działające na osobniki populacji P_t nie zależą od poprzednich populacji, ciąg zmiennych losowych $\{P_t\}$ spełnia warunek Markowa. W sytuacji, gdy parametry operatorów genetycznych są stałe, ciąg zmiennych losowych spełnia również warunek przesunięcia. Algorytm genetyczny może być modelowany za pomocą jednorodnego łańcucha Markowa. Model ten umożliwia znalezienie macierzy przejścia dla algorytmu, badanie ergodyczności łańcucha Markowa oraz badanie klas stanów. Z ergodyczności łańcucha Markowa wynika prawdziwość następującego twierdzenia:

Twierdzenie 4.2

Prawdopodobieństwo osiągnięcia stanu *j* po skończonej liczbie kroków ze stanu początkowego j_0 jest równe jeden.

4.5 Teoria schematów

Pierwszym podejściem formalnym, które miało wyjaśnić zachowanie działania algorytmów genetycznych była teoria schematów zapoczątkowana przez Hollanda [41]. Podejście to miało na celu ocenę zmiany liczby osobników pewnego typu w populacji.

Podstawowym pojęciem w teorii schematów jest wzorzec podobieństwa zdefiniowany jako łańcuch symboli należących do zbioru $\{0,1,\#\}$. Do schematu S należą chromosomy o wartościach genów identycznych jak zapisane we wzorcu podobieństwa na tych pozycjach, na których nie ma symbolu #.

Definicja 4.14 Binarne uniwersum genetyczne jest zbiorem postaci:

$$\Omega = \{(a_0, a_1, ..., a_{l-1}), a_i \in \{0,1\}, i = 1, ..., l-1\} \qquad 4.3.$$

□

Definicja 4.15 Przestrzeń schematów *S* jest zbiorem $S = \{0,1,\#\}^l$

□

Definicja 4.16 Schematem H o wzorcu podobieństwa $(h_0, h_1, ..., h_{l-1}) \in S = \{0,1,\#\}^l$ nazywać będziemy podzbiór uniwersum genetycznego $H \subset \Omega$

$$H = \{(a_0, a_1, ..., a_{l-1}), a_i = \begin{cases} h_i \text{ gdy } h_i \in \{0,1\} \\ 0 \text{ lub } 1 \text{ gdy } h_i = \# \end{cases}, i = 1, ..., l-1\} \quad 4.4.$$

□

Definicja 4.17 Długością definiującą $\Delta(H)$ schematu $H \subset \Omega$ jest maksymalna odległość pomiędzy ustalonymi pozycjami jego kodu:

$$\Delta(H) = \max_{i,j=0,1,...,l-1} \{|i-j| : h_i, h_j \in \{0,1\}\} \quad 4.5.$$

□

Definicja 4.18 Rząd $O(H)$ schematu $H \subset \Omega$ jest liczbą symboli różnych od symbolu „#" (czyli liczbą pozycji ustalonych).

□

Przez *m(H,t)* oznaczymy liczbę osobników w populacji w generacji *t* takich, że ich kody należą do *H*, a przez $f(H,t)$ średnie przystosowanie osobników typu *H* w znajdujących się w populacji w chwili *t*. Oznaczając poprzez $\bar{f}(t)$ średnie przystosowanie osobników populacji w chwili *t* zgodnie z pierwszą hipotezą teorii schematów otrzymamy następujące równanie, określające oczekiwaną liczbę osobników typu H zmieniającą się na skutek selekcji :

$$E(m(H, P_{t+1})) = m(H, P_t) \frac{f(H, P_t)}{\bar{f}(P_t)} \quad 4.6.$$

Oznaczmy przez M liczbę osobników w populacji, przez p_c prawdopodobieństwo wyboru osobnika z populacji P do krzyżowania oraz poprzez p_m intensywność mutacji.

Druga hipoteza teorii schematów mówi, że jeżeli krzyżowanie i mutacja są wykonywane niezależnie, to wartość oczekiwana liczebności schematu H w kolejnym pokoleniu może być oszacowana od dołu w następujący sposób:

$$E(m(H,t+1)) \geq m(H,t)\underbrace{\frac{f(H,t)}{\bar{f}(t)}}_{selekcja}\underbrace{(1-p_m)^{O(H)}}_{mutacja}\underbrace{\left(1-p_c\frac{\Delta(H)}{l-1}\right)}_{krzyzowanie} \qquad 4.7.$$

Szczegółowe wyprowadzenie oszacowania na dolną granicę wartości oczekiwanej liczebności schematu H po pojedynczej epoce ewolucyjnej dla prostego algorytmu genetycznego zostało opisane w pracy [92]. Oszacowanie wyprowadzone zostało na podstawie modelu matematycznego stworzonego przez Vose'a [101] dla SGA. Prezentowane w pracy oszacowanie jest silniejsze od oszacowania wprowadzonego przez Hollanda i przeformułowanego przez Goldberga [35] Michalewicza [58] oraz Reevensa i Rovy'ego [82]. Dane jest ono następującym wzorem:

$$E\big(m(H,t+1)\big) \geq m(H,t)\frac{f(H,t)}{\bar{f}(t)}(1-p_m)^{O(H)}\left(1-p_c\frac{\Delta(H)}{l-1}\left(1-\frac{f(H,t)m(H,t)}{\bar{f}(t)N}\right)\right) \qquad 4.8.$$

Przedstawiona teoria schematów Hollanda jest często krytykowana. Do podstawowych jej wad zaliczamy:

- wyprowadzenie tylko dolnej granicy wartości oczekiwanej liczebności schematu w kolejnym pokoleniu,
- trudności w przewidywaniu globalnego zachowania ,
- przyjęte założenie, że wiedza na temat zachowania schematów pomogą zrozumieć jak i dlaczego działają algorytmy genetyczne,
- brak możliwości badania własności asymptotycznych algorytmu,
- trudności w stworzeniu specyficznych modeli,
- większość algorytmów ewolucyjnych modelowanych za pomocą teorii schematów to proste algorytmy, które są rzadko używane w praktyce.

Przedstawione hipotezy zostały postawione dla prostego algorytmu genetycznego. Istnieje jednak szereg prac dotyczących teorii schematów dla bardziej złożonych algorytmów ewolucyjnych.

4.5.1 Teoria schematów Koza dla programowania genetycznego

W 1992 roku J. R. Koza pokazał, że teoria schematów może być z powodzeniem zastosowane również do zrozumienia działania programowania genetycznego [51]. Zdefiniował schemat jako podprzestrzeń wszystkich drzew zawierających predefiniowany zbiór poddrzew. Zgodnie z definicją J. R. Koza schemat *H* jest reprezentowany jako zbiór S-wyrażeń. Na przykład schemat *H*={(* x y),(- x 1)} reprezentuje wszystkie programy zawierające co najmniej jedno wystąpienie wyrażenia (* x y) i co najmniej jedno wystąpienie wyrażenia (- x 1). Taka definicja uwzględnia jedynie definicje komponentów schematu a nie ich pozycje. Tak więc schematy mogły zostać dopasowane na wiele sposobów, wiele razy w tym samym programie.

4.5.2 Teoria schematów O'Reilly dla programowania genetycznego

Prace Koza nad schematami zostały sformalizowane i przedefiniowane przez O'Reilly [62]. Wyprowadzona została teoria schematów dla programowania genetycznego z selekcją proporcjonalną do przystosowania i krzyżowaniem. Teoria opierała się zdefiniowaniu schematu jako multizbioru poddrzew i fragmentów drzew. Fragmenty drzew były drzewami z co najmniej jednym liściem będącym symbolem „#". Na przykład schemat *H*= {(- x 1), (- x 1), (+ # x)} reprezentował wszystkie schematy z co najmniej jednym wystąpieniem fragmentu drzewa (+ # x) oraz co najmniej dwoma wystąpieniami (- x 1). Podobnie jak w definicji J. R. Koza, definicja schematu obejmuje jedynie definicje komponentów schematu a nie ich pozycję. Definicje schematu U. O'Reilly umożliwiły zdefiniowanie porządku oraz długości schematu. Porządek schematu był liczbą węzłów różnych od symbolu „#" w wyrażeniu albo fragmencie schematu. Długość schematu była liczbą wiązań zawartych w wyrażeniu lub fragmencie drzewa w schemacie razem z wiązaniami łączącymi je razem. Definiowana długość schematu nie była stała, gdyż zależała od sposobu tworzenia instancji schematu wewnątrz programu. Prawdopodobieństwo przerwania $P_d(H,h,t)$ schematu *H*

zawartego w programie *h* w chwili *t* przy krzyżowaniu jest ilorazem długości definiującej *H* w *h* oraz liczby punktów krzyżowania w *h*. Oznacza to, że prawdopodobieństwo $P_d(H,h,t)$ zależy od kształtu i rozmiaru drzewa *h* należącego do schematu. Teoria schematów wyprowadzona przez U. O'Reilly

$$E(i(H,t+1)) \geq i(H,t)\underbrace{\frac{f(H,t)}{\bar{f}(t)}}_{selekcja}\underbrace{\left(1-p_c\overbrace{\max_{h\in P_t} P_d(H,h,t)}^{P_d(H,t)}\right)}_{krzyzowanie} \tag{4.9}$$

przezwycięża ten problem poprzez rozważanie maksymalnego takiego prawdopodobieństwa $P_d(H,t)=\max_{h\in P_t} P_d(H,h,t)$. Istotny jest fakt, że $i(H,t)$ jest liczbą instancji schematu *H* w generacji *t*, a $f(H,t)$ jest średnim przystosowaniem instancji schematu *H*. Wartość $f(H,t)$ została obliczona jako ważona suma przystosowań programów należących do schematu, gdzie wagami były ilorazy liczby instancji schematu *H* zawarte w programie oraz liczba instancji H w całej populacji. Teoria opisuje sposób, w jaki komponenty reprezentacji schematu są propagowane z jednej generacji do następnej. Przedstawiona teoria schematów nie uwzględniała mutacji.

4.5.3 Teoria schematów Whighama dla programowania genetycznego

P. A. Whigham stworzył koncepcję schematu dla programowania genetycznego z użyciem gramatyk bezkontekstowych (CFG-GP) oraz związaną z nim teorię schematów [106, 107, 108]. W CFG-GP programy są wynikiem zastosowania zbioru reguł przepisujących, wziętych z predefiniowanej gramatyki, do symbolu startowego *S*. Proces tworzenia programu może być reprezentowany jako drzewo wywodu, którego wewnętrzne węzły są regułami przepisującymi, a liście są funkcjami i terminalami używanymi w programie. W CFG-GP osobniki populacji są drzewami wywodu. Proces poszukiwania przy pomocy specjalizowanych operatorów mutacji i krzyżowania przebiega tak, aby powstawały tylko prawidłowe drzewa wywodu.

P. A. Whigham definiuje schemat jako częściowe drzewo wywodu, ukorzenione w pewnym nieterminalnym węźle, czyli jako kolekcję reguł przepisujących zorganizowanych w pojedyncze drzewo wywodu. Przy założeniu, że terminale schematu mogą być zarówno symbolami terminalnymi jak i nieterminalnymi gramatyki oraz, że korzeń schematu może być symbolem różnym od symbolu startowego *S*, schemat reprezentuje wszystkie programy, które mogą zostać otrzymane przez uzupełnienie schematu (dodanie innych reguł do liści, dopóki nie zostaną same symbole terminalne) oraz wszystkie programy reprezentowane przez schematy, które zawierają go jako komponent. Gdy korzeniem schematu jest symbol różny od symbolu startowego, schemat może pojawić się wiele razy w drzewie wywodu tego samego programu. Jest to rezultatem nie uwzględniania informacji o pozycji w schemacie.

Definicja schematów P. A. Whighama prowadzi do prostych równań określających przerwanie schematów przez krzyżowanie $P_{d_c}(H,h,t)$ i mutację $P_{d_m}(H,h,t)$. Podobnie jak to było w przypadku O'Reilly, prawdopodobieństwa te zmieniają się wraz z rozmiarem drzewa *h* należącego do schematu. W swojej teorii schematów Whigham użył średnich prawdopodobieństw przerwań instancji schematu przy krzyżowaniu i mutacji $\overline{P}_{d_c}(H,t)$ i $\overline{P}_{d_m}(H,t)$ oraz średniego przystosowania tych instancji $f(H,t)$. Twierdzenie brzmi następująco:

$$E(i(H,t+1)) \geq i(H,t)\frac{f(H,t)}{\bar{f}(t)}\left\{\left[1-p_m\overline{P}_{d_m}(H,t)\right]\left[1-p_c\overline{P}_{d_c}(H,t)\right]\right\} \tag{4.10}$$

Twierdzenie to może zostać zastosowane zarówno do algorytmów genetycznych z ciągami binarnym o stałej długości jak i do standardowego programowania genetycznego poprzez zmiany w gramatyce używanej przez CFG-GP.

4.5.4 Teoria schematów Rosca dla programowania genetycznego

Rosca zaproponował nową definicję schematów, w której schemat był ukorzenionym fragmentem drzewa. Na przykład schemat *H*=(+ # x) reprezentuje wszystkie programy, których korzeń jest symbolem „+”, a drugi argument jest symbolem „x”. Taka definicja schematów wnosi informacje o pozycji, których brakowało w poprzednich definicjach schematów. Konsekwencją jest fakt, że schemat może mieć tylko jedną instancję w obrębie jednego programu. Tak więc badanie propagacji komponentów schematu w populacji jest równoważne z analizowaniem sposobu w jaki zmienia się w czasie liczba programów należących do schematu. Rosca wyprowadził następujące twierdzenie:

$$E(m(H,t+1)) \geq m(H,t)\frac{f(H,t)}{\bar{f}(t)}\left[1-(p_m+p_c)\underbrace{\sum_{h\in H\cap Pop(t)}\overbrace{\frac{O(H)}{N(h)}}^{P_d(H,h,t)}\frac{f(h)}{\sum_{h\in H\cap Pop(t)} f(h)}}_{P_d(H,t)}\right] \tag{4.11}$$

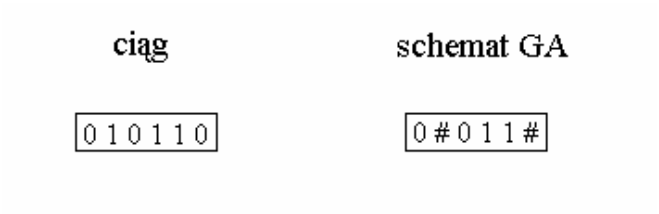
gdzie *N(h)* jest rozmiarem programu *h* należącego do schematu *H*, *f(h)* jest jego przystosowaniem, *O(H)* jest liczbą symboli definiujących, które są w nim zawarte.

Wszystkie trzy wymienione powyżej definicje schematów determinowały podział przestrzeni programów na podprzestrzenie zawierające programy różnych rozmiarów i kształtów. W następnym podrozdziale zostanie przedstawiona definicja schematów wyznaczająca podział przestrzeni programów na podprzestrzenie programów o określonym rozmiarze i kształcie.

4.5.5 Definicja schematu Poli'ego dla programowania genetycznego

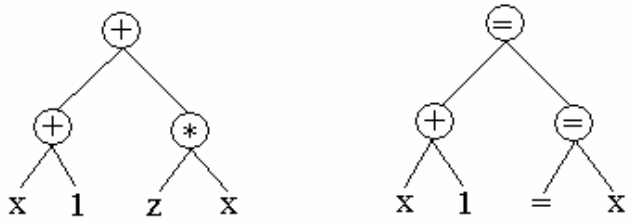
Patrząc na schematy jako na podprzestrzenie przestrzeni możliwych rozwiązań można traktować schematy jako narzędzie matematyczne do opisu, które obszary przestrzeni poszukiwań są próbkowane przez populację. Aby schematy mogły być użyte w programowaniu genetycznym, ich definicja musi umożliwić łatwe obliczenie skutków mutacji, krzyżowania i selekcji. W wymienionych powyżej definicjach schematów dla GP skutki działania na schematach operatorów genetycznych używanych w genetycznym programowaniu były trudne do obliczenia.

Definicja schematów R. Poli'ego i W. Langdona została zainspirowana przez definicje schematów dla liniowych algorytmów genetycznych. W standardowych liniowych algorytmach genetycznych możliwe jest otrzymanie wszystkich schematów zawierających dany ciąg poprzez zastąpienie, na wszystkie możliwe sposoby, jego symboli symbolami „#", które zastępują pojedynczy znak. Przykład ciągu oraz zawartego w nim schematu został przedstawiony na Rys.4.4.



Rys. 4.4 Przykładowy ciąg i zawarty w nim schemat.

Podobne podejście zostało zastosowane przez R. Poli'ego i W. Langdona do otrzymania schematów GP próbkowanych przez program. Schemat powstawał poprzez zastąpienie pojedynczych węzłów symbolem „=". Symbol „=" został tutaj użyty w celu podkreślenia, że zastępujemy tylko pojedynczy węzeł, a nie – jak to było w innych definicjach dla GP – całe poddrzewo. Przykładowy program oraz zawarty w nim schemat przedstawiony został na Rys.4.5 [74, 75, 78].



Rys. 4.5 Przykładowy program oraz zawarty w nim schemat przedstawiony.

Niech F będzie zbiorem funkcji i niech T będzie zbiorem terminali.

Definicja 4.19 GP schemat H jest ukorzenionym drzewem złożonym z węzłów zbioru $F \cup T \cup \{=\}$.

□

Definicja 4.20 Porządkiem $O(H)$ **GP schematu** H nazywamy liczbę węzłów różnych od „=”,

□

Definicja 4.21 Długością $N(H)$ **GP schematu** H nazywamy liczbę wszystkich węzłów schematu.

□

Definicja 4.22 Długością definiującą $\Delta(H)$ **GP schematu** H nazywamy liczbę wiązań w minimalnym fragmencie drzewa zawierającym wszystkie symbole w schemacie różne od „=”.

□

Definicja 4.23 Hiperprzestrzenią nazywamy schemat G nie zawierający żadnego węzła definiującego **(O(G)=0).** Schemat H jest **hiperpłaszczyzną**, jeżeli zawiera co najmniej jeden węzeł definiujący. Schemat $G(H)$ powstały ze schematu przez zastąpienie wszystkich węzłów definiujących symbolem = nazywamy **hiperprzestrzenną związaną ze schematem** H.

□

Tak jak w tradycyjnym binarnym algorytmie genetycznym zdefiniowany porządek, długość oraz długość definiująca są niezależne od kształtu i rozmiaru programów w aktualnej populacji.

Dla selekcji proporcjonalnej do przystosowania, krzyżowania jednopunktowego oraz mutacji punktowej [54] R. Poli i W. Langdon otrzymali następujące twierdzenie:

$$E(m(H,t+1)) \geq m(H,t)\frac{f(H,t)}{\bar{f}(t)}(1-p_m)^{O(H)} \tag{4.12}$$
$$\left\{1-p_c\left[p_{diff}(t)\left(1-\frac{m(G(H),t)f(G(H),t)}{M\bar{f}(t)}\right)+\frac{\Delta(H)}{(N(H)-1)}\frac{m(G(H),t)f(G(H),t)-m(H,t)f(H,t)}{M\bar{f}(t)}\right]\right\}$$

Innymi słowy, wyprowadzona została dolna granicy wartości oczekiwanej liczebności schematu po pojedynczym kroku programowania genetycznego.

W rozprawie korzystać będziemy z wyników teoretycznych otrzymanych przez R. Polie'go w oparciu o dokładną teorię mikroskopową schematów dla programowania genetycznego

z krzyżowaniem polegającym na przepinaniu poddrzew.

Dokładna teoria mikroskopowa schematów R. Polie'go zostanie szerzej omówiona w Rozd. 7.

5 Rozszerzenie modelu markowowskiego dla algorytmu genetycznego z operatorami przesunięcia i permutacji

W bieżącym rozdziale przedstawimy markowowski model prostego algorytmu genetycznego rozszerzonego o dodatkowe operatory – operator przesunięcia bitów w prawo lub lewo oraz operator permutacji bitów. W Rozd. 6 model ten zostanie zastosowany do modelowania algorytmu genetycznego z hierarchicznym chromosomem (wraz z operatorami mutacji alleli, mutacji grup alleli oraz krzyżowaniem hierarchicznym).

Oznaczmy przez U_m zbiór wszystkich możliwych ciągów binarnych o długości l ($U_m = (Z_2)^l$), przez Ω_r zbiór wszystkich możliwych elementowych r-elementowych populacji, a przez Ω dowolną przestrzeń probabilistyczną.

Definicja 5.1 Selekcja jest operatorem losowym

$$S : \Omega_r \times \Omega \to U_m \tag{5.1}$$

takim, że

$$S(j, \omega) = S_j \tag{5.2}$$

przy ustalonym *m* jest zmienną losową taką, że

$$F_k(j) = P(S_j = k) = 0 \ \forall k : r_k^j = 0 \tag{5.3}$$

□

Definicja 5.2

Niech

$$\Lambda = \left\{ p = (p_o, ..., p_{m-1}) \in \Re^m : \sum_{i=0}^{m-1} p_i = 1; p_i \geq 0 \forall i = 0,1,...,m-1 \right\} \tag{5.4}$$

i niech przekształcenie $\varphi : \Omega_r \to \Lambda$ będzie dane wzorem:

$$\varphi(j) = (\varphi_0(j), \varphi_1(j), ..., \varphi_{m-1}(j)) \text{ dla każdego } j = 0,...,\binom{m+r-1}{m-1} \tag{5.5}$$

gdzie $\varphi_i(j) = \frac{r_i^j}{r}$

Niech

$$f : U_m \to \Re^+ - \{0\} \tag{5.6}$$

Oznaczmy $f_i = f(i)$ - wartość funkcji przystosowania osobnika *i*.

Selekcja proporcjonalna jest operatorem losowym postaci:

$$F(j) = \left(F_0(j), F_1(j), \ldots F_{m-1}(j)\right) = \frac{(\varphi_0(j) f_0, \varphi_1(j) f_1, \ldots, \varphi_{m-1}(j) f_{m-1})}{\sum_{k=0}^{m-1} \varphi_k(j) f_k} \tag{5.7}$$

□

Definicja 5.3

Niech $\eta : \Omega \to U_m$ będzie wektorem losowym i niech $mut : U_m \times U_m \to U_m$ będzie przekształceniem danym wzorem:

$$mut(x, y) = x \oplus y \tag{5.8}$$

Mutacją nazywamy operator losowy

$$mut(i, \eta) = i \oplus \eta \tag{5.9}$$

□

W dalszej części pracy będziemy rozważać jedynie rozkłady η dane wzorem:

$$P(\eta = i) = \mu_i = a^{\mathbf{1}^T i}(1-a)^{l - \mathbf{1}^T i} \tag{5.10}$$

gdzie

$\mathbf{1}^T i$ - liczba jedynek w zapisie dwójkowym liczby *i*

l-$\mathbf{1}^T i$ - liczba zer w zapisie dwójkowym liczby *i*

$a \in \left(0, \frac{1}{2}\right)$ nazywamy rzędem mutacji

W Tab.5.1 przedstawione zostały wartości operatora mutacji obliczone dla przykładowych argumentów *i* oraz η

i	η	$mut(i, \eta) = i \oplus \eta$
0011	1001	1010
0000	1100	1100

Tab. 5.1 Wartości operatora mutacji obliczone dla przykładowych argumentów *i* oraz η.

Definicja 5.4

Niech S będzie selekcją i niech

$$\Lambda = \left\{ p = (p_o,..., p_{m-1}) \in \Re^m : \sum_{i=0}^{m-1} p_i = 1; p_i \geq 0 \forall i = 0,1,..., m-1 \right\} \tag{5.11}$$

Heurystycznym poszukiwaniem definiowanym przez mutację nazywamy poszukiwanie o macierzy przejścia zdefiniowanej następująco:

$$Q_{j,i} = r! \prod_{k=0}^{r-1} \frac{(G_k(j))^{r_k^i}}{r_k^i!} \tag{5.12}$$

gdzie $G : \Omega_r \to \Lambda$ dane jest wzorem:

$$G_k(j) = P\left(S_j \oplus \eta = k\right) \tag{5.13}$$

□

Definicja 5.5

Niech $\zeta : \Omega \to U_m$ oraz $\kappa : \Omega \to \{0,1\}$ będą zmiennymi losowymi, a przekształcenie $K : U_m \times U_m \times U_m \times \{0,1\} \to U_m$ będzie dane wzorem:

$$K(x, y, z, u) = x \times z^u \oplus z^{1-u} \otimes y \tag{5.14}$$

gdzie $z^u = z$ jeśli $u = 0$

$z^u = \bar{z}$ jeśli $u = 1$

Krzyżowaniem nazywamy operator losowy $K(x, y, \zeta, \kappa)$

□

Definicja 5.6 Krzyżowaniem jednopunktowym nazywamy krzyżowanie z rozkładem:

$$\chi_i = P(\zeta = i) = \begin{cases} \dfrac{b}{l-1} & \text{gdy } \exists k \in (0,l] : i = 2^k - 1 \\ 1 - b - \dfrac{b}{l-1}\text{d} & \text{dla } i = 0 \\ 0 & \text{dla pozostalych } i \end{cases} \tag{5.15}$$

$b \in [0,1]$ nazywane jest rzędem krzyżowania

□

W Tab. 5.2 przedstawione zostały wartości operatora krzyżowania obliczone dla przykładowych argumentów x,y,z oraz *u*

x	y	z	u	$K(x,y,z,u) = x \times z^u \oplus z^{1-u} \otimes y$
1111	0000	0011	0	0011
1111	0000	0011	1	1100

Tab. 5.2 Wartości operatora mutacji alleli obliczone dla przykładowych argumentów *i* oraz η.

Definicja 5.7

Niech S i $\overline{S}$ będą selekcjami o tych samych rozkładach.

Heurystycznym poszukiwaniem definiowanym przez krzyżowanie nazywamy łańcuch Markowa $\{\xi_i, i \in N\}$ o macierzy przejścia zdefiniowanej następująco:

$$Q_{j,i} = P(\xi_n = i \mid \xi_{n-1} = j) = r! \prod_{k=0}^{r-1} \frac{(G_k(j))^{r_k^i}}{r_k^i!} \tag{5.16}$$

gdzie $G : \Omega_r \to \Lambda$ dane jest wzorem:

$$G_k(j) = P(K(S_j, \overline{S}_j, \zeta, \kappa) = k) = P(S_j \otimes \zeta^{\kappa} \oplus \zeta^{1-\kappa} \otimes \overline{S}_j = k) \tag{5.17}$$

□

Ponieważ $S_j, \overline{S}_j, \zeta, \kappa$ są niezależne, więc:

$$G_k(j) = P\left(\bigcup_{[x \otimes i^p \oplus i^{1-p} \otimes y = k]} S_j = x \wedge \overline{S}_j = y \wedge \zeta = i \wedge \kappa = p \right) =$$

$$\sum_{x,y,i,p} P(S_j = x) P(\overline{S}_j = y) P(\zeta = i) P(\kappa = p) \left[x \otimes i^p \oplus i^{1-p} \otimes y = k \right] =$$

$$\sum_{x,y,i,p} F_x(j) F_y(j) \chi_i P(\kappa = p) \left[x \otimes i^p \oplus i^{1-p} \otimes y = k \right] =$$

$$\sum_{x,y,i} F_x(j) F_y(j) \chi_i P(\kappa = 0) \left[x \otimes i \oplus \bar{i} \otimes y = k \right] + \sum_{x,y,i} F_x(j) F_y(j) \chi_i P(\kappa = 1) \left[x \otimes \bar{i} \oplus i \otimes y = k \right]$$

$$\frac{1}{2}\sum_{x,y,i} F_x(j)F_y(j)\chi_i P(\kappa=0)\left[x\otimes i\oplus \bar{i}\otimes y=k\right]+$$

$$\frac{1}{2}\sum_{x,y,i} F_x(j)F_y(j)\chi_i P(\kappa=1)\left[x\otimes \bar{i}\oplus i\otimes y=k\right]+$$

$$\frac{1}{2}\sum_{x,y,i} F_x(j)F_y(j)\chi_{\bar{i}} P(\kappa=0)\left[x\otimes \bar{i}\oplus i\otimes y=k\right]+$$

$$\frac{1}{2}\sum_{x,y,i} F_x(j)F_y(j)\chi_{\bar{i}} P(\kappa=1)\left[x\otimes i\oplus \bar{i}\otimes y=k\right]=$$

$$\frac{1}{2}\sum_{x,y,i} F_x(j)F_y(j)\chi_i \left[x\otimes i\oplus \bar{i}\otimes y=k\right]\left(P(\kappa=0)+P(\kappa=1)\right)+$$

$$\frac{1}{2}\sum_{x,y,i} F_x(j)F_y(j)\chi_{\bar{i}} \left[x\otimes \bar{i}\oplus i\otimes y=k\right]\left(P(\kappa=0)+P(\kappa=1)\right)=$$

$$\frac{1}{4}\sum_{x,y,i} F_x(j)F_y(j)\times$$
$$\left(\chi_i\left[x\otimes i\oplus \bar{i}\otimes y=k\right]+\chi_{\bar{i}}\left[x\otimes \bar{i}\oplus i\otimes y=k\right]+\chi_i\left[y\otimes i\oplus \bar{i}\otimes x=k\right]+\chi_{\bar{i}}\left[y\otimes \bar{i}\oplus i\otimes x=k\right]\right)=$$

$$\frac{1}{4}\sum_{x,y,i} F_x(j)F_y(j)\left(\left(\chi_i+\chi_{\bar{i}}\right)\left[x\otimes i\oplus \bar{i}\otimes y=k\right]+\left(\chi_i+\chi_{\bar{i}}\right)\left[x\otimes \bar{i}\oplus i\otimes y=k\right]\right)=$$

$$\frac{1}{2}\sum_{x,y,i} F_x(j)F_y(j)\left(\chi_i+\chi_{\bar{i}}\right)\left[x\otimes i\oplus \bar{i}\otimes y=k\right]$$

W celu rozszerzenia modelu Vose'a dla prostego algorytmu genetycznego o dodatkowe operatory: przesunięcie w lewo, przesunięcie w prawo i permutację, musimy zdefiniować nowe operatory genetyczne i ich rozkłady prawdopodobieństw.

Aby wyjaśnić ideę operacji przesunięcia w lewo i prawo, rozważmy nieskończony w lewo i prawo ciąg bitów. Przyjmijmy, że znaczące będą tylko bity indeksowane wartościami 0,…,*l-1*.

W Tab. 5.3 przedstawiony został przykład nieskończonego ciągu binarnego.

numer bitu	...	-1	0	...	c-1	c	...
wartość bitu	...	0	1	...	1	0	...

Tab 5.3 Przykład nieskończonego ciągu binarnego.

Przesunięcie w prawo albo w lewo o *m* bitów odpowiada przesunięciu całego ciągu o *m* w prawo lub w lewo odpowiednio i zastąpieniu pierwszych lub ostatnich *m* bitów przez 0. Ponieważ znaczące są tylko bity o indeksach 0,…,*l-1*, pozostałe bity zostaną odrzucone.

W Tab. 5.4 przedstawiony został ciąg binarny będący wynikiem przesunięcia w prawo o *m=5* ciągu binarnego o długości *l=7* złożonego z samych 1.

numer bitu	...	-1	0	1	2	3	4	5	6	7	...
wartość bitu	...	0	0	0	0	0	0	1	1	0	...

Tab. 5.4 Ciągu binarny złożony z samych 1 po operacji przesunięcia o 5 bitów w prawo.

Formalne definicje operatorów przesunięcia w lewo i w prawo oraz ich rozkładów prawdopodobieństw zostały przedstawione jak następuje.

Niech $\gamma : \Omega \to U_m$ będzie wektorem losowym i niech $left(u) : U_m \to \{0,1,...,l\}$ będzie przekształceniem zwracającym indeks pierwszego od lewej bitu równego jeden, które dane jest wzorem:

$$left(u) = \begin{cases} 0 & \text{dla } u = \mathbf{0} \\ \min\{c \in \{0,1,...,1-1\} : u_c = 1\} + 1 & \text{w przeciwnym przypadku} \end{cases} \tag{5.18}$$

gdzie $\mathbf{0} = \underbrace{(0,...,0)}_{l \text{ razy}}$

Niech $pwp : U_m \times U_m \to U_m$ oraz $pwl : U_m \times U_m \to U_m$ będą przekształceniami danymi wzorami:

$$\forall 0 \le w < l \;\; pwp_w(x,u) = \begin{cases} 0 & \forall w < left(u) \\ x_{w-left(u)} & \forall w \ge left(u) \end{cases} \tag{5.19}$$

$$\forall 0 \le w < l \;\; pwl_w(x,u) = \begin{cases} 0 & \forall w \ge l - left(u) \\ x_{w+left(u)} & \forall w < l - left(u) \end{cases} \tag{5.20}$$

Definicja 5.8 Niech γ będzie zmienną losową.

Przesunięciem w prawo nazywamy operator losowy $pwp(x,\gamma)$.

□

Definicja 5.9 Niech γ będzie zmienną losową.

Przesunięciem w lewo nazywamy operator losowy $pwl(x,\gamma)$.

□

W przykładzie 5.1 przedstawione zostały wartości operatora przesunięcia w prawo obliczone dla przykładowych argumentów x_1, u_1 oraz x_2, u_2

x_1 =111111 $\quad$ x_2 =101101
u_1 =010000 (przesunięcie o 2 bity w prawo) $\quad$ u_2 =001000 (przesunięcie o 3 bity w prawo)
$z_1 = pwp(x_1,u_1)$=001111 $\quad$ $z_2 = pwp(x_2,u_2)$=000101

Przykład 5.1 Wartości operatora przesunięcia w prawo obliczone dla przykładowych argumentów x_1, u_1 oraz x_2, u_2.

Rozkład prawdopodobieństwa wylosowania maski przesunięcia dany jest wzorem:

$$P(\gamma = u) = \gamma_u = \begin{cases} \frac{1}{l} & \text{jesli } (\mathbf{1},u) = 1 \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{5.21}$$

Niech S będzie selekcją.

Definicja 5.10 Heurystycznym poszukiwaniem definiowanym przez przesunięcie w prawo nazywamy poszukiwanie o macierzy przejścia zdefiniowanej następująco:

$$Q_{j,i} = r! \prod_{k=0}^{r-1} \frac{(G_k(j))^{r_k^i}}{r_k^i!} \tag{5.22}$$

gdzie $G : \Omega_r \to \Lambda$ dane jest wzorem:

$$G_k(j) = P\left(pwp(S_j,\gamma) = k\right) \tag{5.23}$$

□

Definicja 5.11 Heurystycznym poszukiwaniem definiowanym przez przesunięcie w lewo nazywamy poszukiwanie o macierzy przejścia zdefiniowanej następująco:

$$Q_{j,i} = r! \prod_{k=0}^{r-1} \frac{(G(j))_k^{r_k^i}}{r_k^i!}$$

gdzie $G : \Omega_r \to \Lambda$ dane jest wzorem:

$$G_k(j) = P\left(pwl(S_j, \gamma) = k\right) \tag{5.24}$$

□

Kolejnym wprowadzonym operatorem jest operator permutacji. Zdefiniujemy tylko operator transpozycji, ponieważ każdą permutację da się przedstawić jako transpozycję elementów sąsiednich.

Definicja 5.12

Niech $\alpha : \Omega \to U_m$ będzie wektorem losowym i niech $t : U_m \times U_m \to U_m$ będzie przekształceniem danym wzorem:

$$\forall 0 \le w < l \quad t_w(x, v) = \begin{cases} x_w & \text{dla } v = \mathbf{0} \\ x_{\min\{a \in \{0,\ldots,l-1\} : v_a = 1\}} & \text{dla } w = \max\{a \in \{0,\ldots,l-1\} : v_a = 1\} \\ x_{\max\{a \in \{0,\ldots,l-1\} : v_a = 1\}} & \text{dla } w = \min\{a \in \{0,\ldots,l-1\} : v_a = 1\} \end{cases} \tag{5.25}$$

Transpozycją nazywamy operator losowy $t(x, \alpha)$

□

W przykładzie 5.2 przedstawione zostały wartości operatora transpozycji obliczone dla przykładowych argumentów x_1, v_1 oraz x_2, v_2

x_1 =111111 x_2 =101101

v_1 =010010 (transpozycja bitów 1 i 4) v_2 =001010 (transpozycja bitów 2 i 4)

$z_1 = t(x_1, v_1)$=111111 $z_2 = t(x_2, v_2)$=100111

□

Przykład 5.2 Wartości operatora transpozycji obliczone dla przykładowych argumentów x_1, v_1 oraz x_2, v_2.

Rozkład prawdopodobieństwa wylosowania maski transpozycji dany jest wzorem:

$$P(\alpha = v) = \alpha_v = \begin{Bmatrix} \frac{1}{\binom{1}{2}+1} & \forall v : (v,\mathbf{1}) \in \{0,2\} \\ 0 & \text{w przeciwnym przypadku} \end{Bmatrix} \tag{5.26}$$

Niech S będą selekcją.

Definicja 5.13 Heurystycznym poszukiwaniem definiowanym przez transpozycję nazywamy poszukiwanie o macierzy przejścia zdefiniowanej następująco:

$$Q_{j,i} = r! \prod_{k=0}^{r-1} \frac{(G_k(j))^{r_k^i}}{r_k^i!}$$

gdzie $G : \Omega_r \to \Lambda$ dane jest wzorem:

$$G_k(j) = P\left(t(S_j, \alpha) = k\right) \tag{5.27}$$

□

Definicja 5.14

Niech $\zeta : \Omega \to U_m$, $\kappa : \Omega \to \{0,1\}$, $\eta : \Omega \to U_m$, $\gamma : \Omega \to U_m$ oraz $\alpha : \Omega \to U_m$ będą zmiennymi losowymi, a przekształcenie $M : {U_m}^7 \times \{0,1\} \to U_m$ będzie dane wzorem:

$$M(x, y, v_1, z, o, s, u) = mut\left(pwp\left(t\left(K(x, y, z, u), o\right), s\right)v_1\right)$$

Mieszaniem nazywamy operator losowy

$$M(x, y, \eta, \zeta, \alpha, \gamma, \kappa)$$

□

Oznaczmy $m_{xyz} = P\left(M(x, y, \eta_1, \zeta, \alpha, \gamma, \kappa) = z\right)$

Definicja 5.15 Heurystycznym poszukiwaniem definiowanym przez mieszanie nazywamy poszukiwanie, dla którego $G:\Omega_r \to \Lambda$ dane jest wzorem:

$$G_z(j) = P\left(M(S_j, \overline{S}_j, \eta_1, \zeta, \alpha, \gamma, \kappa) = z\right) \tag{5.28}$$

Mamy

$$G_z(j) = P\left(M(S_j, \overline{S}_j, \eta_1, \zeta, \alpha, \gamma, \kappa) = z\right) = \sum_{x,y \in U_m} F_x(j) F_y(j) P\left(M(x, y, \eta_1, \zeta, \alpha, \gamma, \kappa) = z\right) = \tag{5.29}$$

$$\sum_{x,y \in U_m} F_x(j) F_y(j) m_{xyz}$$

□

Lemat 1 Dla każdego $z \in U_m$, dla każdego $j \in \Omega_r$ zachodzi nierówność

$$G_z(j) > 0 \tag{5.30}$$

Dowód:

Wystarczy pokazać, że dla każdego $\mathrm{x, y, z} \in \mathrm{U_m}$ $m_{xyz} > 0$.

Ponieważ $\eta, \zeta, \kappa, \gamma, \alpha$ zmienne losowe niezależne, więc mamy:

$$m_{xyz} = P\left(M(x, y, \eta, \zeta, \alpha, \gamma, \kappa) = z\right) =$$

$$\sum_{i,p,v,s,o} P(\zeta = i) P(\kappa = p) P(\gamma = s) P(\alpha = o) P(\eta = v) \left[mut\left(t\left(pwp\left(K(x, y, i, p), s\right), o\right), v\right) = z\right]$$

Ponieważ mamy:

$$P(\zeta = i) = \chi_i$$
$$P(\eta = v) = \mu_v$$
$$P(\gamma = s) = \gamma_s$$
$$P(\alpha = o) = \alpha_0$$
$$p \in \{0,1\}$$

więc otrzymamy:

$$\sum_{i,p,v,s,o} P(\zeta = i) P(\kappa = p) P(\gamma = s) P(\alpha = o) P(\eta = v) \left[mut\left(t\left(pwp\left(K(x, y, i, p), s\right), o\right), v\right) = z\right] =$$

$$\sum_{i,v,s,o} \frac{\left(\chi_i + \chi_{\bar{i}}\right)}{2} \gamma_s \alpha_o \mu_v \left[mut\left(t\left(pwp\left(x \otimes i \oplus \bar{i} \otimes y, s\right), o\right), v\right) = z\right]$$

Wykażemy, że da się dobrać maski krzyżowania, przesunięcia, transpozycji oraz mutacji w taki sposób, że jeden ze składników sumy jest większy od zera.

niech i_1: $\chi_{i_1} > 0$; $s_1 : \gamma_{s_1} > 0$; $o_1 : \alpha_{o_1} > 0$;

niech $v_1 := t(pwp(x \otimes i_1 \oplus \bar{i}_1 \otimes y, s_1), o_1) \oplus z$

ponieważ $\mu_v > 0$ dla każdego $v \in U_m$, więc w szczególności $\mu_{v_1} > 0$

Dla i_1, s_1, o_1, v_1 zachodzi:

$$mut(t(pwp(x \otimes i_1 \oplus \bar{i}_1 \otimes y, s_1), o_1), v_1) =$$

$$t(pwp(x \otimes i_1 \oplus \bar{i}_1 \otimes y, s_1), o_1) \oplus v_1 =$$

$$t(pwp(x \otimes i_1 \oplus \bar{i}_1 \otimes y, s_1), o_1) \oplus t(pwp(x \otimes i_1 \oplus \bar{i}_1 \otimes y, s_1), o_1) \oplus z = z$$

czyli:

$$[mut(t(pwp(x \otimes i_1 \oplus \bar{i}_1 \otimes y, s_1), o_1), v_1) = z] = 1$$

gdzie [ ] – oznacza binarną ewaluację wyrażenia logicznego, czyli

$$[X] = \begin{cases} 1 \text{ gdy wyrazenie X jest prawdziwe} \\ 0 \quad \text{gdy wyrazenie X jest fałałszy} \end{cases}$$

tak więc:

$$\frac{(\chi_{i_1} + \chi_{\bar{i}_1})}{2} \gamma_{s_1} \alpha_{o_1} \mu_{v_1} [mut(t(pwp(x \otimes i_1 \oplus \bar{i}_1 \otimes y, s_1), o_1), v_1) = z] > 0$$

Skoro jeden ze składników sumy jest silnie większy od zero, to mamy:

$$\sum_{i,v,s,o} \frac{(\chi_i + \chi_{\bar{i}})}{2} \gamma_s \alpha_o \mu_v [mut(t(pwp(x \otimes i \oplus \bar{i} \otimes y, s), o), v) = z] =$$

$$\sum_{i \neq i_1, v \neq v_1, s \neq s_1, o \neq o_1} \frac{(\chi_i + \chi_{\bar{i}})}{2} \gamma_s \alpha_o \mu_v [mut(t(pwp(x \otimes i \oplus \bar{i} \otimes y, s), o), v) = z] +$$

$$\frac{(\chi_{i_1} + \chi_{\bar{i}_1})}{2} \gamma_{s_1} \alpha_{o_1} \mu_{v_1} [mut(t(pwp(x \otimes i_1 \oplus \bar{i}_1 \otimes y, s_1), o_1), v_1) = z] \geq$$

$$\frac{(\chi_{i_1} + \chi_{\bar{i}_1})}{2} \gamma_{s_1} \alpha_{o_1} \mu_{v_1} [mut(t(pwp(x \otimes i_1 \oplus \bar{i}_1 \otimes y, s_1), o_1), v_1) = z] > 0$$

więc dla każdego $x, y, z \in U_m$ $m_{xyz} > 0$

Ponieważ dla każdego $x, y, z \in U_m$ $m_{xyz} > 0$ więc dla każdego $j \in \Omega_r$, dla każdego $z \in U_m$

$G_z(j) > 0$, co kończy dowód lematu.

Ze wzoru (5.22) możemy wysnuć następujący wniosek:

Wniosek 5.1

Macierz przejścia $Q_{j,i}$ ma wszystkie wyrazy dodatnie

Wniosek 5.2

Łańcuch Markowa opisujący dynamikę algorytmu genetycznego jest ergodyczny.
Z twierdzenia ergodycznego wynika, że:

- $\exists \lim_{n\to\infty} Q_{ij}(n) = Q_j,\ j = 1,...,s$ (granica nie zależy od *i*)
- gdzie $Q_j \geq \delta > 0$ dla $j = 1,...,s$

$$| Q_{ij}(n) - Q_j | \leq (1 - s\delta)^{\left\lfloor \frac{n}{r} - 1 \right\rfloor}$$

Ważną konsekwencją ergodyczności jest fakt, że algorytm genetyczny odwiedzi wszystkie stany, czyli w konsekwencji przeszuka całą przestrzeń artefaktów zakodowaną w U_m, bez względu na wybór populacji początkowej.

Wniosek 5.3

Wszystkie stany są istotne.

Wniosek 5.4 Prawdopodobieństwo, że po skończonej liczbie kroków osiągniemy stan *j* będąc w stanie j_0 jest równe 1.

6 Model markowowski dla algorytmu genetycznego z hierarchicznym chromosomem

6.1 Definicja przestrzeni artefaktów oraz przestrzeni genotypów

Podstawowym zadaniem w projektowaniu jest zdefiniowanie przestrzeni rozwiązań potencjalnych. Rozwiązania potencjalne będą reprezentowane za pomocą modeli graficznych artefaktów. Dla uproszczenia przestrzeń tę będziemy nazywać przestrzenią artefaktów. Poniżej przedstawione zostały definicje potrzebne do zdefiniowania przestrzeni artefaktów.

W wielu zastosowaniach projektowych modele graficzne artefaktów tworzone są z transformowanych kopii podstawowych kształtów geometrycznych zwanych prymitywami.

Niech *F* będzie zbiorem transformacji z R^n w R^n, przekształcających ograniczone podzbiory na ograniczone podzbiory. Dowolny element $f \in F$ jest nazywany **dopuszczalną transformacją**.

Definicja 6.1 Niech *F* będzie zbiorem dopuszczalnych transformacji z R^n w R^n. Niech *S* będzie zbiorem ograniczonych podzbiorów w R^n.

□

Jeśli $f(p) = q$ dla $f \in F$, $p, q \in S$ implikuje $p = q$, to elementy *S* są nazywane **prymitywami** nad *F*.

Artefakt jest sumą transformowanych prymitywów zwanych **składowymi**. Każda składowa jest reprezentowana jako para *(p,f)*, gdzie *p* oznacza prymityw a *f* jest odpowiednią transformacją.

Definicja 6.2 Niech *S* będzie zbiorem prymitywów nad zbiorem F dopuszczalnych transformacji z R^n w R^n.

Niech $Q \subset S \times F$ będzie strukturą artefaktu.

Artefakt *A* dla Q zdefiniowany jest następująco:

$$A = \bigcup_{(p,f)\in Q} f(p) \tag{6.1}$$

□

Definicja 6.3 Niech *S* będzie zbiorem prymitywów nad zbiorem F dopuszczalnych transformacji z R^n w R^n.

Przestrzeń artefaktów *R* zdefiniowana jest następująco:

$$R = \left\{ A : A = \bigcup_{(p,f)\in Q} f(p) \text{ gdzie } Q \in P(S \times F) \right\} \tag{6.2}$$

□

Definicja 6.4 Niech S będzie zbiorem prymitywów nad zbiorem F dopuszczalnych transformacji z R^n w R^n. Niech *R* będzie rodziną artefaktów.

Zbiór klas składowych dla przestrzeni artefaktów R zdefiniowany jest następująco:

$$K = \bigcup_{j=1}^{d} K_j \text{ gdzie dla każdego } j \in \{1,...,d\} \tag{6.3}$$

$$K_j \subset \{s : \exists A \in R : s \in A\}$$

□

Ze względów technicznych rozważać będziemy artefakty z przestrzeni *R* spełniające następujące własności:

1) liczba składowych dowolnego artefaktu z $B \in P_A$ nie przekracza M_A,
2) klasy $K_1^B, K_2^B, ..., K_d^B$ składowych dowolnego artefaktu $B \in P_A$ spełniają następujące założenie:

   $\#K_1^B \le N_1, \#K_2^B \le N_2 ..., \#K_d^B \le N_d$ dla ustalonych liczb naturalnych $N_1, N_2, ..., N_d$

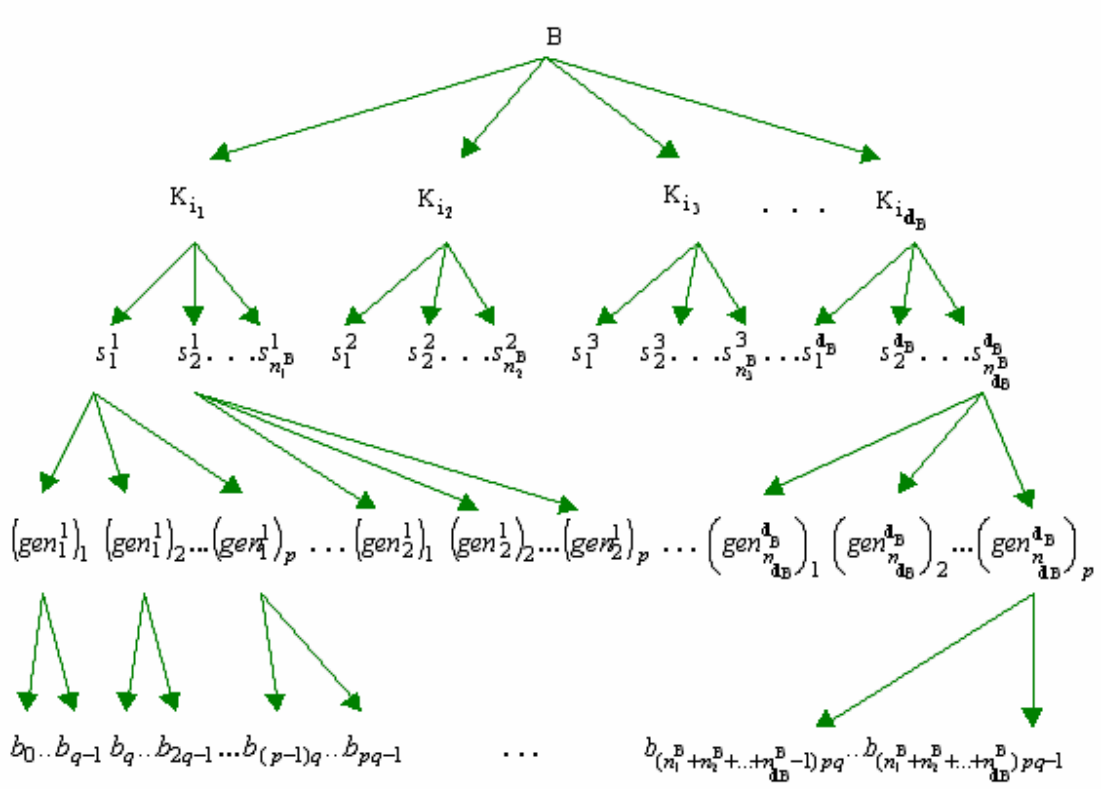
   takich, że $\sum_{i=1}^{d_A} N_i \le M_A$,
3) Każda składowa opisana jest przez *p* parametrów (genów),

4) Każdy gen zakodowany jest za pomocą ciągu binarnego o długości q.

Artefakty spełniające wymienione własności wyznaczają przestrzeń artefaktów sparametryzowanych. Przestrzeń tą oznaczamy przez P_A

Niech $B \in P_A$ będzie artefaktem, n_i^B liczbą składowych w K_i dla $i \le d_B$, gdzie d_B jest liczbą klas składowych artefaktu B. Z punktu 2) definicji P_A mamy $\sum_{i=1}^{d_A} n_i^B \le M_A$.

Artefakt B możemy reprezentować następującym drzewem:



Rys.6.1 Przykład drzewa hierarchii.

Na Rys. 6.1 Dla artefaktu B mamy:

$K_{i_1}, K_{i_2}, ..., K_{i_{d_B}}$ są klasami

s_j^i oznacza *j*-tą składową klasy *i*-tej dla każdego $i = 0,..., d_B$ oraz dla każdego $j = 0,..., n_i^B$

$\left(gen_j^i\right)_k$ oznacza *k*-ty gen *j*-tej składowej klasy *i*-tej dla każdego $i = 0,..., d_B$, dla każdego

$j = 0,..., n_i^B$ oraz dla każdego $k = 1,..., p$

b_k ,jest *k*-tym bitem dla każdego $k = 0,..., (n_1^B + n_2^B + ... + n_{d_B}^B)\, pq$

Definicja 6.5 Składową zerową będziemy nazywać składową, której wszystkie geny są ciągami złożonymi z samych zer.

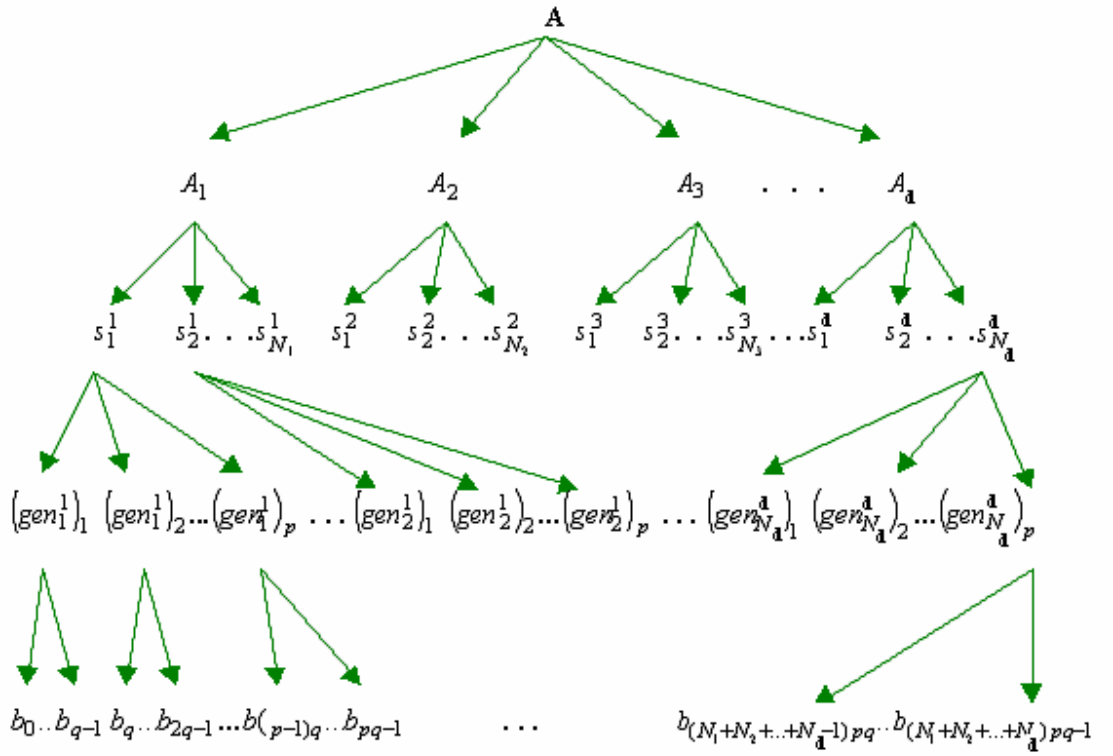
□

Definicja 6.6 Uniwersalne drzewo hierarchii to drzewo, które powstaje z drzewa hierarchii poprzez dodanie $N_i - n_i^B$ zerowych składowych**,** dla każdej klasy i takiej, że $N_i - n_i^B > 0$.

□

Drzewa, które mają takie same wszystkie składowe reprezentują ten sam artefakt, niezależnie od położenia składowych i są ze sobą utożsamiane.

Drzewo uniwersalne dla artefaktu z P_A ma postać:



Rys.6.2 Uniwersalne drzewo hierarchii.

Na Rys. 6.2 dla artefaktu A mamy:

$K_1, K_2, ..., K_d$ są klasami

s_j^i oznacza *j*-tą składową klasy *i*-tej dla każdego $i = 0,...,d$ oraz dla każdego $j = 0,...,N_i$

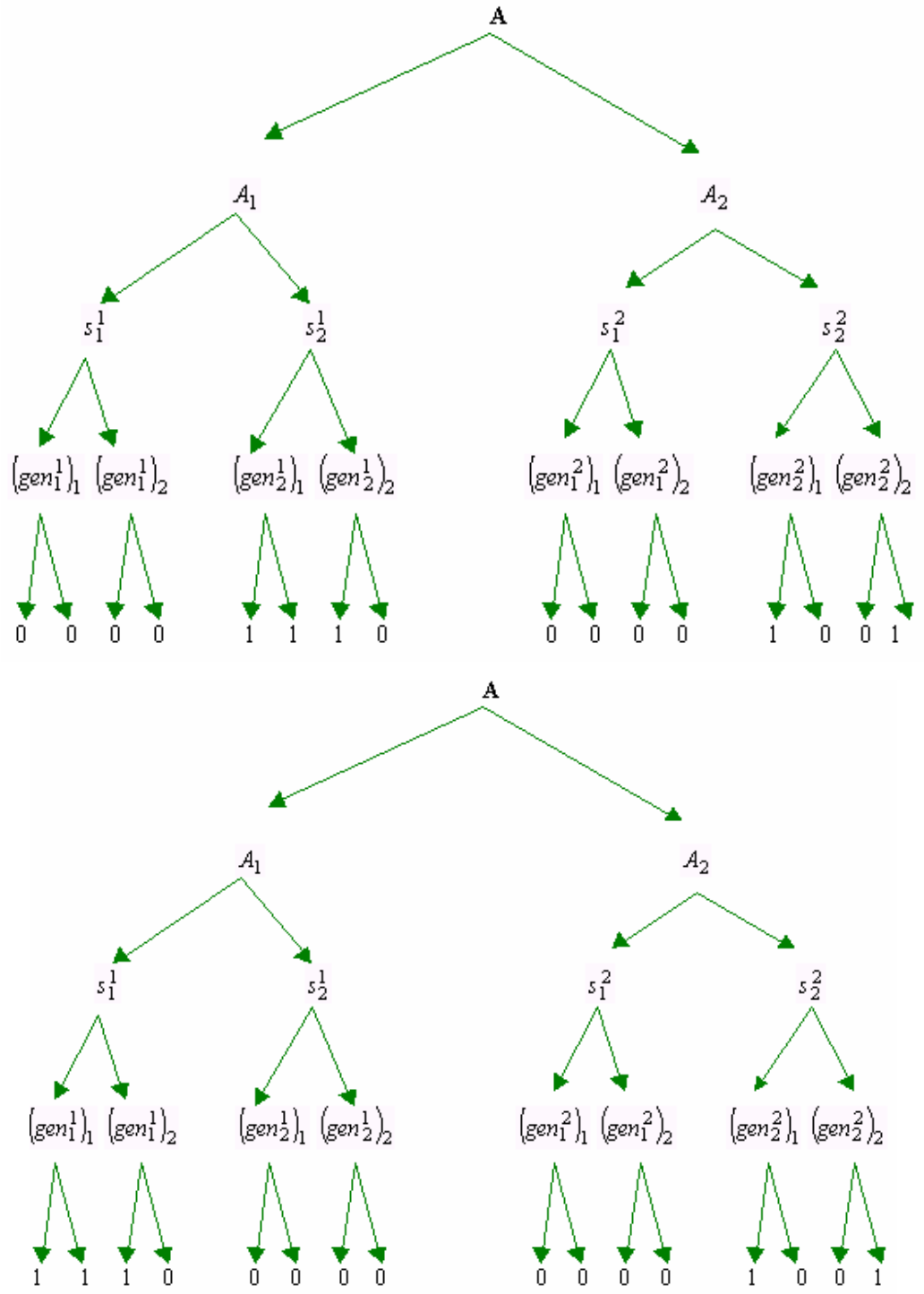
$\left(gen_j^i\right)_k$ oznacza *k*-ty gen *j*-tej składowej klasy *i*-tej dla każdego $i = 0,...,d$, dla każdego $j = 0,...,N_i$ oraz dla każdego $k = 1,...,p$

b_k jest *k*-tym bitem dla każdego $k = 0,...,M_A pq$, gdzie $M_A = N_1 + N_2 + ... + N_d$

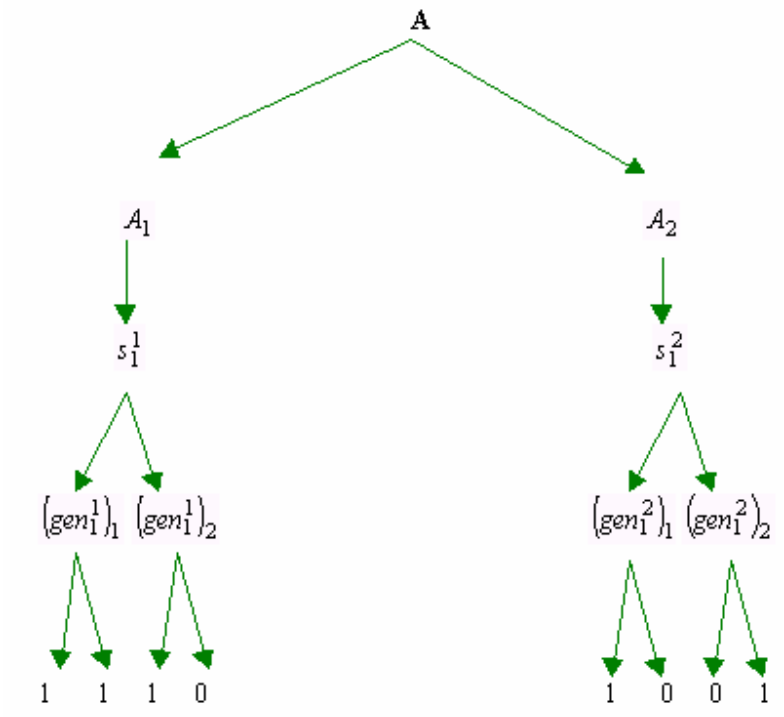
Przykład 6.1

Niech M_A=4, r=2, p=2, q=2.

Uniwersalne drzewa hierarchii przedstawione na Rys. 6.3. powstały z tego samego drzewa hierarchii przedstawionego na Rys.6.4.



Rys. 6.3 Przykłady uniwersalnych drzew hierarchii.



Rys. 6.4 Przykład drzewa hierarchii

Definicja 6.7 Przestrzeń ciągów binarnych U_m dla $m = 2^l$ gdzie $l = M_A pq$ zdefiniowana jest następująco:

$U_m = (Z_2)^l$

$\#U_m = 2^l$

□

Definicja 6.8 Kodem artefaktu z przestrzeni P_A będziemy nazywać ciąg binarny $x = (b_0, b_1, ..., b_{l-1}) \in U_m$ będący ciągiem bitów z najniższego poziomu uniwersalnego drzewa hierarchii dla tego artefaktu.

□

6.2 Model markowowski algorytmu

Algorytm genetyczny z hierarchicznym chromosomem używa operatorów genetycznych takich jak mutacja alleli, mutacja grup alleli oraz krzyżowanie hierarchiczne. Działanie tych operatorów zostało przedstawione w Rozd. 4.

Przedstawiony poniżej model został stworzony dla algorytmu genetycznego z operatorami mutacji oraz krzyżowania hierarchicznego. Krzyżowanie hierarchiczne zostanie zamodelowane za pomocą krzyżowania jednopunktowego z punktem podobieństwa znajdującym się pomiędzy składowymi. Aby umożliwić krzyżowanie z punktem krzyżowania (P_1, P_2,g,b), gdzie P_1 jest różne od P_2 a składowe o numerach P_1 i P_2 należą do tej samej klasy, wykorzystana zostanie operacja transpozycji składowych, będąca rozszerzeniem operacji transpozycji z Rozd. 5. Mutacja alleli oraz grup alleli odpowiada klasycznej mutacji.

Oznaczmy przez Ω_r zbiór wszystkich możliwych r-elementowych populacji, przez Ω dowolną przestrzeń probabilistyczną a przez r_k^j liczbę wystąpień k-tego elementu w populacji j-tej.

Niech $mut : U_m \times U_m \to U_m$ będzie zdefiniowane zgodnie z definicją 5.3:

$$mut(x, y) = x \oplus y$$

Mutację grup alleli (usuwanie całych składowych) możemy interpretować jako zamianę wszystkich bitów niezerowych wchodzących w skład składowej na bity zerowe.

Tak więc mutacji alleli oraz mutacji grup alleli będzie odpowiadać klasyczna mutacja - operator losowy $mut(i, \eta) = i \oplus \eta$, gdzie oraz $\eta : \Omega \to U_m$ jest wektorem losowym o rozkładzie danym następującym wzorem:

$$P(\eta = i) = \mu_i = a^{\mathbf{1}^T i}(1-a)^{l-\mathbf{1}^T i}$$

Niech $\zeta : \Omega \to U_m$, $\kappa : \Omega \to \{0,1\}$ oraz niech $K : U_m \times U_m \times U_m \times \{0,1\} \to U_m$ będzie zdefiniowane zgodnie ze definicją 5.5 w następujący sposób:

$$K(x, y, z, u) = x \times z^u \oplus z^{1-u} \otimes y$$

gdzie $z^u = z$ jeśli $u = 0$

$z^u = \bar{z}$ jeśli $u = 1$

Krzyżowanie hierarchiczne zostało zamodelowane za pomocą krzyżowania jednopunktowego z rozkładem zdefiniowanym w taki sposób, aby punkt krzyżowania przebiegał pomiędzy składowymi:

$$\chi_i = P(\zeta = i) = \begin{cases} \dfrac{b}{M_A - 1} & \text{gdy } \left(\exists k \in (0, l] : i = 2^k - 1\right) \wedge \left(\exists u \in (0, M_A] : \mathbf{1}^T i = upq\right) \\ 1 - b - \dfrac{b}{M_A - 1} & \text{dla } i = \mathbf{0} \\ 0 & \text{dla pozostalych } i \end{cases} \tag{6.4}$$

Z konstrukcji ciągów binarnych odpowiadających artefaktom wynika, że krzyżowanie o powyższym rozkładzie jest krzyżowaniem hierarchicznym.

Przykład 6.2

Niech P_A będzie przestrzenią wszystkich artefaktów takich, że:

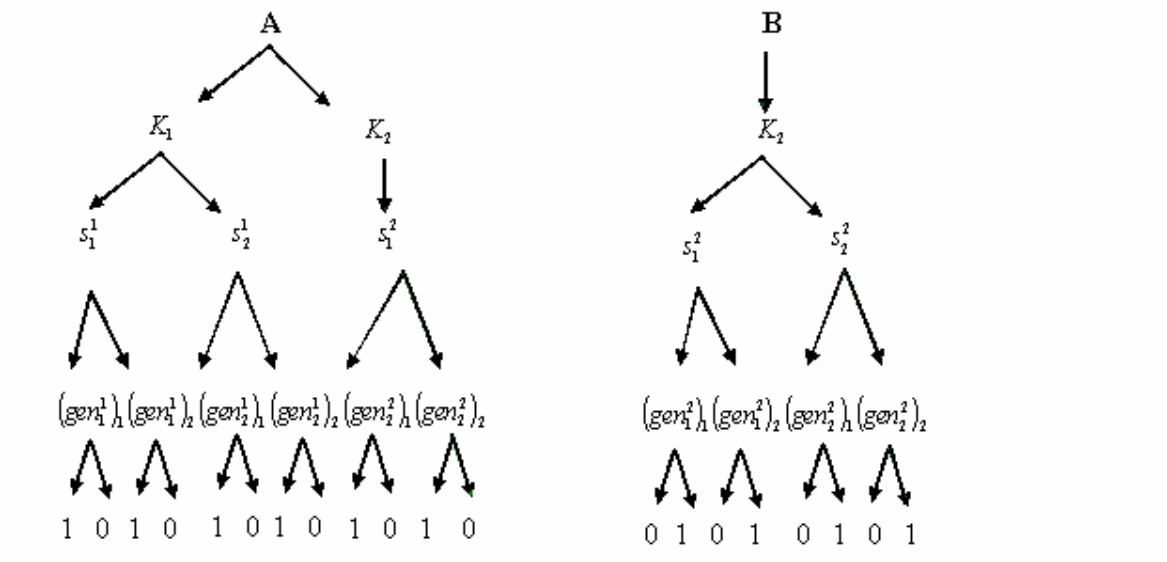
a) liczba składowych dowolnego artefaktu z $B \in P_A$ nie przekracza M_A=4,

b) klasy K_1^B, K_2^B składowych dowolnego artefaktu $B \in P_A$ spełniają następujące warunki: $\#K_1^B \le 2, \#K_2^B \le 2$,

c) każda składowa opisana jest przez *p=2* parametrów (genów),

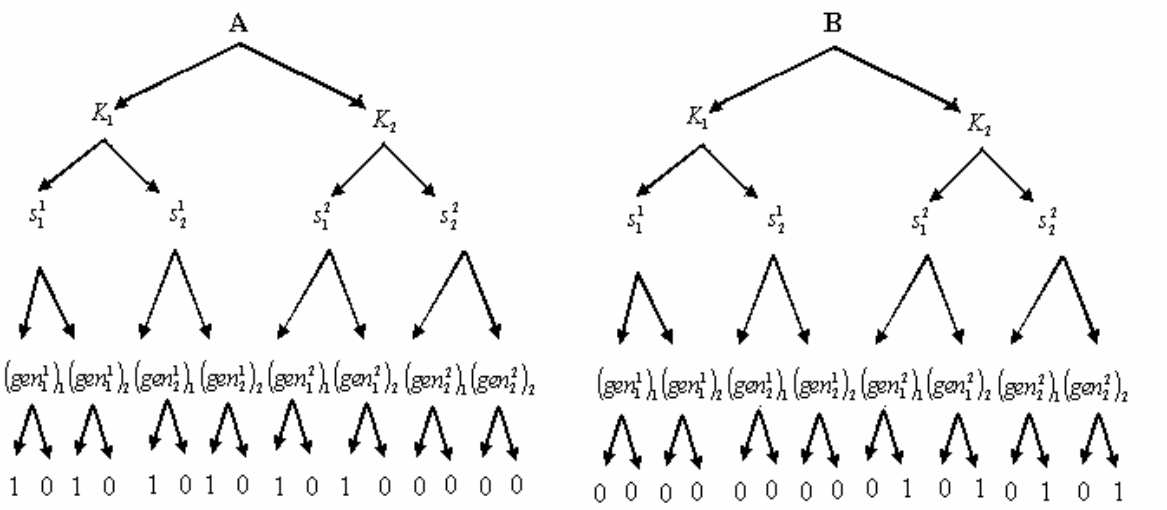
d) każdy gen zakodowany jest za pomocą ciągu binarnego o długości *q=2.*

Niech drzewa hierarchii artefaktów – osobników rodzicielskich *A* oraz *B* - wyglądają jak na Rys. 6.5.



Rys.6.5 Drzewa hierarchii przykładowych artefaktów – osobników rodzicielskich.

Uniwersalne drzewa hierarchii dla drzew hierarchii przedstawionych na Rys. 6.5 zostały przedstawione na Rys. 6.6.



Rys.6.6 Uniwersalne drzewa hierarchii osobników rodzicielskich.

Kodem artefaktu *A* będzie ciąg binarny postaci: 1010101010100000, a kodem artefaktu *B* będzie ciąg 0000000001010101. Maska krzyżowania jest wybierana z rozkładem zgodnym ze wzorem 6.4.

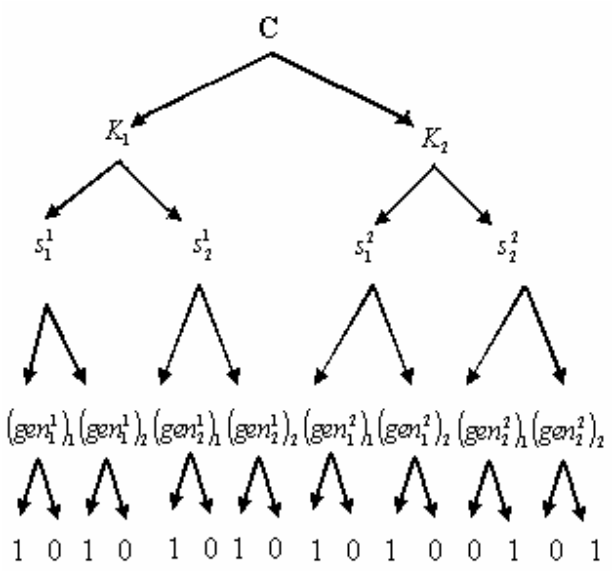
Niech maska krzyżowania z będzie postaci 0000000000001111 i niech *u* będzie równe 1.
Wtedy wynik krzyżowania *t* będzie ciągiem binarnym postaci:

$$t = K(x, y, z, u) = x \otimes z^u \oplus z^{1-u} \otimes y =$$

$$1010101010100000 \otimes 1111111111110000 \oplus 0000000000001111 \otimes 0000000001010101 =$$

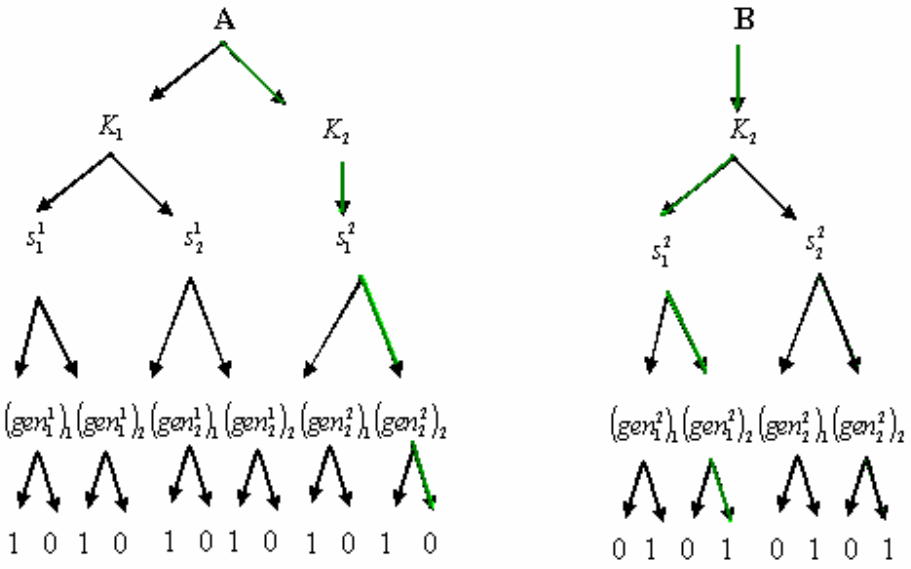
$$1010101010100000 \oplus 0000000000000101 = 1010101010100101$$

Ciąg binarny *t* odpowiada artefaktowi o uniwersalnym drzewie hierarchicznym przedstawionym na Rys. 6.7.



Rys.6.7 Uogólnione drzewa hierarchii osobnika powstałego w wyniku krzyżowania.

Powstały osobnik *C* ma cztery składowe: pierwsze trzy składowe pochodzą od osobnika *A,* natomiast ostatnia składowa pochodzi od osobnika *B.*
Ten sam wynik otrzymalibyśmy stosując krzyżowanie hierarchiczne z punktem krzyżowania (3,1,2,2) dla takich samych osobników rodzicielskich, gdzie punkt krzyżowania jest czwórką (P_1, P_2, *g*, *b*), P_1 jest numerem składowej u osobnika pierwszego, P_2 jest numerem składowej u osobnika drugiego, *g* jest numerem genu, *b* jest numerem bitu.
Na Rys. 6.8 przedstawione zostały drzewa hierarchii artefaktów *A* oraz *B* z zaznaczonym punktem krzyżowania (3,1,2,2).



Rys.6.8 Drzewa hierarchii osobników rodzicielskich z zaznaczonym punktem krzyżowania.

Zastosowanie krzyżowania hierarchicznego dla osobników rodzicielskich przedstawionych na Rys. 6.8 z punktem krzyżowania (3,1,2,2) jest równoważne krzyżowaniu jednopunktowemu z rozkładem danym wzorem 6.3 dla osobników będących kodami artefaktów z Rys. 6.8, maski krzyżowania *z* postaci 0000000000001111 oraz *u* równego 1.

Aby umożliwić powstawanie uogólnionych drzew hierarchicznych z dowolnie rozmieszczonymi składowymi w obrębie klas (na przykład, jak na Rys. 6.6.), należy przed krzyżowaniem dokonać permutacji ciągów bitów odpowiadających poszczególnym składowym. W celu zamodelowania takiej operacji należy wykonać operację transpozycji składowych będącą rozszerzeniem operacji transpozycji zdefiniowanej w Rozd. 5
(definicja 5.12.).

Definicja 6.9

Niech $v \in U_m$ i niech $k_1, k_2 \in \{0,...,l-1\}$ będą dane następującymi wzorami:

$$k_1 = \min\{a \in \{0,...,l-1\} : v_a = 1\} \tag{6.5}$$
$$k_2 = \max\{a \in \{0,...,l-1\} : v_a = 1\}$$

Niech $\alpha\_s : \Omega \to U_m$ będzie wektorem losowym i niech $t\_s : U_m \times U_m \to U_m$ będzie przekształceniem danym wzorem:

$$\forall 0 \le w < l\ ;\ \ t\_s_w(x,v) = \begin{cases} x_{k_1+i} & \text{dla } w = k_2 + i; \forall 0 \le i < pq; k_1 + i < l; k_2 + i < l \\ x_{k_2+i} & \text{dla } w = k_1 + i; \forall 0 \le i < pq; k_1 + i < l; k_2 + i < l \\ x_w & \text{w przeciwnym przypadku} \end{cases} \tag{6.6}$$

Transpozycją składowych nazywamy operator losowy $t\_s(x, \alpha\_s)$.

□

Aby zapewnić, że transpozycji ulegną jedynie składowe tej samej klasy, rozkład prawdopodobieństwa wylosowania maski transpozycji składowych dany jest wzorem:

$$P(\alpha\_s = v) = \alpha\_s_v = \begin{cases} \dfrac{1}{d\binom{\mathrm{M_A}/d}{2}} & (\forall v : (v,\mathbf{1}) = 2) \wedge \left(\left\lfloor \dfrac{k_1}{l/d} \right\rfloor = \left\lfloor \dfrac{k_2}{l/d} \right\rfloor\right) \wedge (k_1 \bmod pq = k_2 \bmod pq) \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{6.7}$$

Definicja 6.10

Niech $\zeta : \Omega \to U_m$, $\kappa : \Omega \to \{0,1\}$, $\eta : \Omega \to U_m$, $\alpha\_s_1 : \Omega \to U_m$ oraz $\alpha\_s_2 : \Omega \to U_m$ będą zmiennymi losowymi, a przekształcenie $M : {U_m}^6 \times \{0,1\} \to U_m$ będzie dane wzorem:

$$M(x, y, v_1, z, o_1, o_2, u) = mut\big(K(t\_s(x, o_1), t\_s(y, o_2), z, u)v_1\big) \tag{6.8}$$

Mieszaniem nazywamy operator losowy

$$M(x, y, \eta, \zeta, \alpha\_s_1, \alpha\_s_2, \kappa) \tag{6.9}$$

□

Definicja 6.11 Heurystycznym poszukiwaniem definiowanym przez mieszanie nazywamy łańcuch Markowa $\{\xi_i, i \in N\}$ o macierzy przejścia zdefiniowanej następująco:

$$Q_{j,i} = P(\xi_n = i \mid \xi_{n-1} = j) = r! \prod_{k=0}^{m-1} \frac{(G(j))_k^{r_k^i}}{r_k^i!} \tag{6.10}$$

gdzie $G : \Omega_r \to \Lambda$ dane jest wzorem:

$$G_z(j) = M(x, y, \eta, \zeta, \alpha\_s_1, \alpha\_s_2, \kappa) \tag{6.11}$$

□

Ponieważ operatory losowe mieszania zdefiniowane w rozszerzonym modelu markowowskim oraz dla algorytmu genetycznego z hierarchicznym chromosomem różnią się tylko rozkładami operatorów genetycznych krzyżowania oraz transpozycji, można sformułować następujący lemat:

Lemat.1 Dla każdego $z \in U_m$, dla każdego $j \in \Lambda$ zachodzi nierówność

$$G_z(j) > 0 \forall \tag{6.12}$$

Dowód:

Ponieważ $\eta, \zeta, \kappa, \alpha\_s_1, \alpha\_s_2$,zmienne losowe niezależne, więc mamy:

$$M(x, y, \eta, \zeta, \alpha\_s_1, \alpha\_s_2, \kappa) =$$

$$\sum_{i,p,v,o_1,o_2} P(\zeta = i) P(\kappa = p) P(\eta = v) P(\alpha\_s_1 = v) P(\alpha\_s_2 = v)$$

$$\left[\left(t\_s(x, o_1) \otimes i^p \oplus i^{1-p} \otimes t\_s(y, o_2)\right) \oplus v = z\right] =$$

$$\sum_{i,v} \frac{(\chi_i + \chi_{\bar{i}})}{2} \eta_v \alpha\_s_{1o_1} \alpha\_s_{2o2} \left[\left(t\_s(x, o_1) \otimes i^p \oplus i^{1-p} \otimes t\_s(y, o_2)\right) \oplus v = z\right]$$

Niech *i*: $\chi_i > 0$; $v := (t\_s(x, o_1) \otimes i) \oplus (\bar{i} \otimes t\_s(y, o_2)) \oplus z$

Ponieważ $\eta_v > 0$ dla każdego $v \in U_m$ oraz

$(t\_s(x, o_1) \otimes i) \oplus (\bar{i} \otimes t\_s(y, o_2)) \oplus v = z$

więc dla każdego $x, y, z \in U_m$ $m_{xyz} > 0$

Ponieważ dla każdego $x, y \in U_m$ $m_{xyz} > 0$ więc $\forall j \in \Lambda, \forall z \in U_m \; G_z(j) > 0$

Z lematu 1 wynikają analogiczne wnioski do wniosków z rozdziału piątego.

Ze wzoru (6.10) wynika następujący wniosek

Wniosek 6.1

Macierz przejścia $Q_{j,i}$ ma wszystkie wyrazy dodatnie

Wniosek 6.2

Łańcuch Markowa opisujący dynamikę algorytmu genetycznego jest ergodyczny.

Z twierdzenia ergodycznego wynika, że:

- $\exists \lim_{n\to\infty} Q_{ij}(n) = Q_j,\ j = 1,...,s$ (granica nie zależy od *i*)

  gdzie $Q_j \geq \delta > 0$ dla $j = q_1, q_2, ...q_{s_1}$

- $| Q_{ij}(n) - Q_j | \leq (1 - s_1\delta)^{\left\lfloor \frac{n}{r} - 1 \right\rfloor}$

Wniosek 6.3

Wszystkie stany są istotne.

Wniosek 6.4

Prawdopodobieństwo, że będąc w stanie początkowym $j_{0,}$ po skończonej liczbie kroków osiągniemy stan *j* jest równe 1.

7 Teoria schematów dla algorytmu genetycznego z hierarchicznym chromosomem

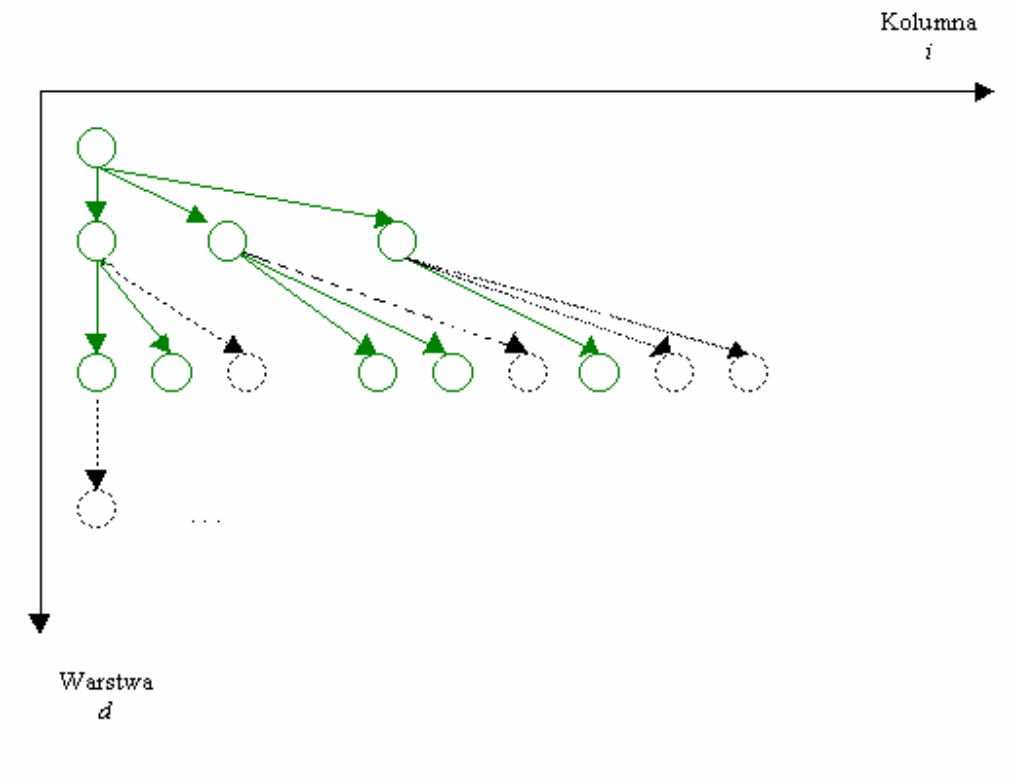
W niniejszym rozdziale przedstawione zostały wyniki teoretyczne otrzymane dla algorytmu genetycznego z hierarchicznym chromosomem na podstawie teorii mikroskopowej schematów R. Poli'ego dla programowania genetycznego z krzyżowaniem polegającym na przepinaniu poddrzew.

Zgodnie z definicją schematów Poli'ego [73], schematy dzielą przestrzeń programów na podoprzestrzenie programów o stałym, określonym rozmiarze i kształcie. Krzyżowanie hierarchiczne będzie modelowane za pomocą krzyżowania silnie uzależnionego od typu (ang. strongly typed crossover) opisanego przez D. J. Montana w [59].

Zastosowanie mikroskopowej teorii schematów Polie'go umożliwia obliczenie prawdopodobieństwa, że w epoce t nowo stworzony osobnik pasuje do schematu H oraz efektywnego przystosowania dla schematu H.

W dalszej części rozdziału przedstawione zostały otrzymane wyniki teoretyczne oraz prosty przykład obrazujący zastosowanie teorii Polie'go dla algorytmu genetycznego z hierarchicznym chromosomem.

Definicja 7.1 Drzewowo niezależny kartezjański system referencyjny [73] otrzymany jest poprzez rozważenie idealnego nieskończonego drzewa składającego się z węzłów o maksymalnej liczbie potomków a_{max}. Przykład takiego drzewa pokazany jest na Rys. 7.1.



Rys. 7.1 System referencyjny.

Takie maksymalne drzewo zawiera jeden węzeł na głębokości 0, a_{max} węzłów na głębokości 1, $(a_{max})^2$ węzłów na głębokości 2, w ogólności $(a_{max})^d$ węzłów na głębokości d.
Węzły są opisywane przez parę (d,i) określającą odpowiednio glebokosc d oraz numer kolumny i. Systemu tego można uzywac również do opisywania drzew, które nie sa maksymalne.

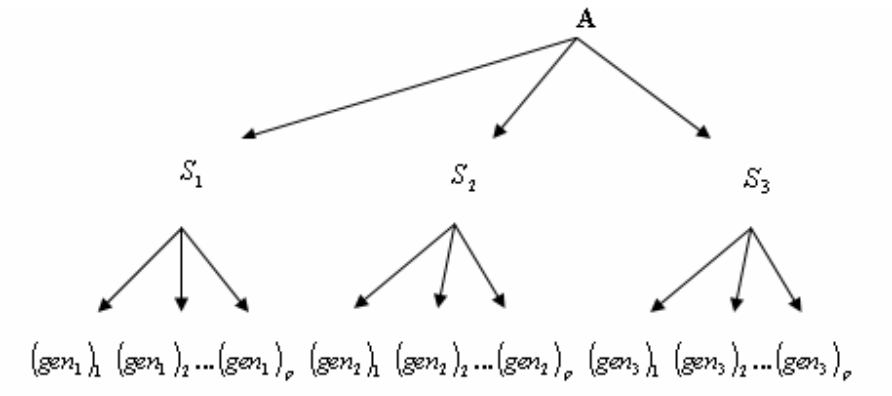
Krzyżowanie silnie uzależnione od typu, którego będziemy chcieli użyć w celu sformalizowania krzyżowania hierarchicznego, zostało przez Montanę w [59] zdefiniowane w sposób następujący:
Węzeł p_1 będący punktem krzyżowania u rodzica pierwszego wybierany jest spośród **wszystkich węzłów drzewa**. Punkt krzyżowania p_2 u rodzica drugiego wybierany jest spośród **węzłów tego samego typu**. Gdy nie ma takiego węzła losowanie punktów krzyżowania wykonywane jest ponownie. Krzyżowanie polega na kopiowaniu wszystkich węzłów, które nie należą do poddrzewa o korzeniu p_1 od rodzica pierwszego do dziecka pierwszego, wszystkich węzłów, które nie należą do poddrzewa o korzeniu p_2 od rodzica drugiego do dziecka drugiego oraz

kopiowaniu odpowiednio poddrzewa o korzeniu p_1 do dziecka drugiego, a poddrzewa o korzeniu p_2 do dziecka pierwszego.

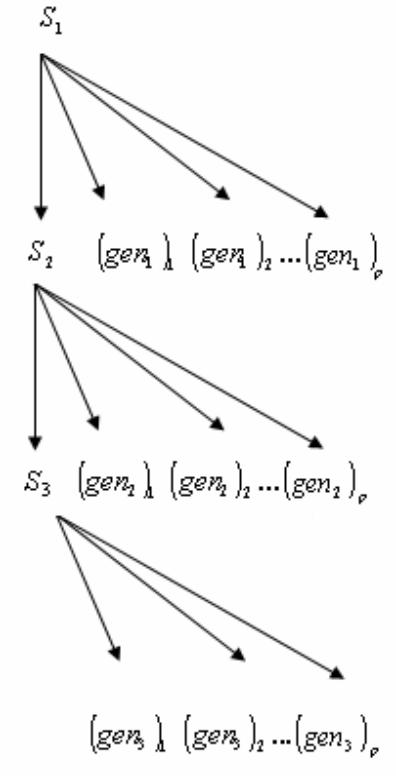
Pomiędzy krzyżowaniem hierarchicznym a krzyżowaniem silnie uzależnionym od typu możemy znaleźć następujące różnice:

- w pierwszym przypadku punkt krzyżowania wybierany jest spośród węzłów tej samej klasy, natomiast w drugim – spośród węzłów tego samego typu,
- w pierwszym przypadku jedynie węzły znajdujące się na lewo od punktu krzyżowania są kopiowane wraz z poddrzewami, odpowiednio od rodzica pierwszego do dziecka pierwszego i od rodzica drugiego do dziecka drugiego, natomiast w drugim przypadku wszystkie węzły oprócz węzłów należących do poddrzewa o korzeniu będącym punktem krzyżowania są kopiowane wraz z poddrzewami odpowiednio od rodzica pierwszego do dziecka pierwszego i od rodzica drugiego do dziecka drugiego.

Aby modelować krzyżowanie hierarchiczne wykorzystując krzyżowanie silnie uzależnione od typu musimy odpowiednio przeorganizować chierarchiczny chromosom. Tworzymy drzewo dla chromosomu w taki sposób, aby korzeniem drzewa była jego pierwsza składowa. Bezpośrednimi potomkami korzenia będą geny opisujące go oraz druga składowa. Potomkami drugiej składowej będą geny, które ją opisują oraz trzecia składowa. Ogólnie, potomkami *i*-tej składowej będą geny, które ją opisują oraz składowa *i+1*. Potomkami ostatniej składowej będą jedynie geny opisujące ją. Przykładowy hierarchiczny chromosom został przedstawiony na Rys. 7.2, a odpowiadające mu drzewo na Rys. 7.3.



Rys.7.2 Przykład hierarchicznego chromosomu.

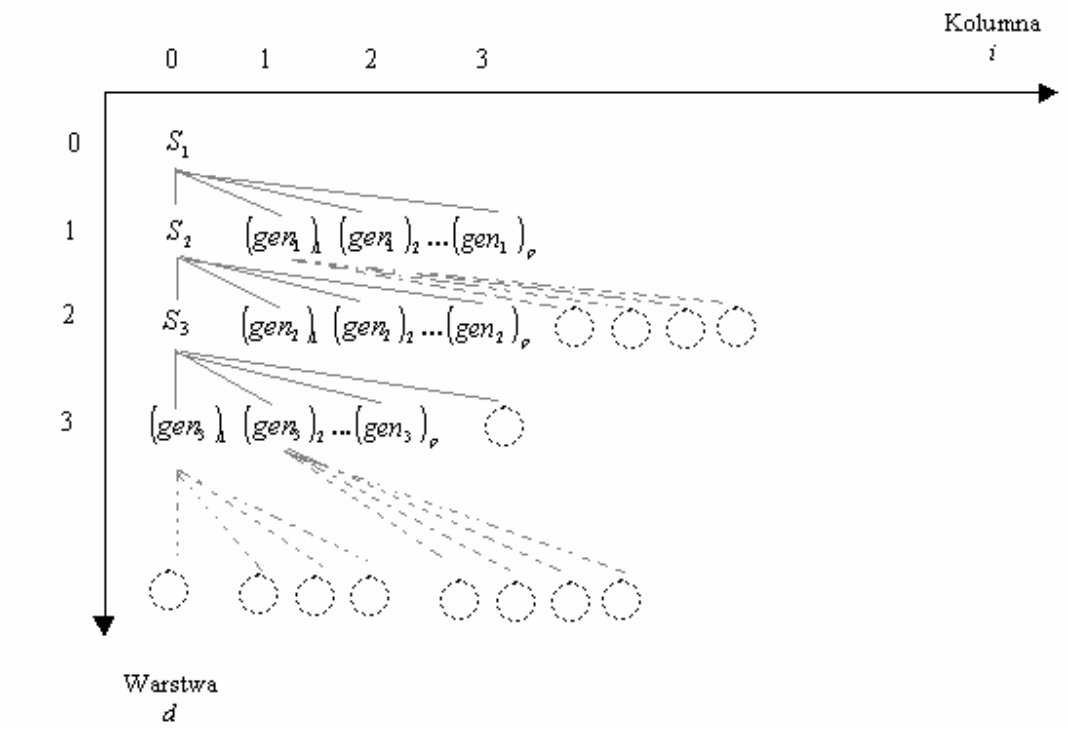


Rys.7.3. Drzewo odpowiadające hierarchicznemu chromosomowi z rys.7.2.

Dla zwiększenia przejrzystości rysunków (Rys. 7.2. oraz Rys. 7.3) nie umieszczono na nich ciągów binarnych kodujących geny.

Po przeorganizowaniu chromosomu, kodujemy węzły drzewa w kartezjańskim systemie referencyjnym.

Przyjmując, że każdy gen opisany jest przez jeden bit, chromosom z Rys. 7.2. zakodowany zostanie tak, jak to jest przedstawione na Rys. 7.4.



Rys.7.4. Zakodowany w kartezjańskim systemie referencyjnym chromosom z rysunku .7.3.

Mając już zdefiniowany sposób kodowania hierarchicznych chromosomów w kartezjańskim układzie współrzędnych (za pomocą system referencyjnego), zdefiniujemy funkcje, które umożliwią nam obliczenie prawdopodobieństwa, że w epoce *t* nowo stworzony osobnik pasuje do schematu *H* oraz efektywnego przystosowania dla schematu *H*.

Definicja 7.2 Funkcja $h(d,i,h_1)$ zwraca węzeł o współrzędnych (d,i) w drzewie h_1, jeśli (d,i)jest poprawną parą współrzędnych. Jeżeli (d,i) nie należy do drzewa h_1, zwracana jest wartość $\emptyset$.

□

Definicja 7.3 Funkcja $IB(d,i)$ przyjmująca wartość 1, gdy węzeł o współrzędnych (d,i) jest bitem oraz 0 w przeciwnym wypadku zdefiniowana jest następująco:

$$IB : N^3 \to \{0,1\} \tag{7.1}$$

$$IB(d,i,h_1) = \begin{cases} 1 & \text{jesli } h(d,i,h_1) \in \{0,1\} \\ 0 & \text{w przeciwnym przypadkup} \end{cases}$$

gdzie *p* jest liczbą genów opisujących składową, a *q* jest liczbą bitów kodujących jeden gen.

□

Definicja 7.4 Funkcja ***CM*** przyjmująca 1, gdy węzły należą do tej samej klasy, 0 w przeciwnym wypadku zdefiniowana jest następująco:

$$CM : N^6 \to \{0,1\} \tag{7.2}$$

$$CM(d_1,i_1,d_2,i_2,h_1,h_2) = \begin{cases} 1 & \text{jesli } \left(IB(d_1,i_1,h_1) = IB(d_2,i_2,h_2) = 0\right) \wedge \left(h(d_1,i_1,h_1) = h(d_2,i_2,h_2)\right) \\ 0 & \text{w przeciwnym przypadku} \end{cases}$$

□

Definicja 7.5 Niech *max_depth(h)* będzie maksymalną głębokością drzewa

Prawdopodobieństwo wybrania węzła o współrzędnych (d_1,i_1) **u rodzica** h_1 **oraz węzła o współrzędnych** (d_2,i_2) **u rodzica** h_2 **dla krzyżowania hierarchicznego** dane jest wzorem:

$$p_{hc}(d_1,i_1,d_2,i_2 \mid h_1,h_2) = \frac{CM(d_1,i_1,d_2,i_2,h_1,h_2)}{\sum_{D_1=0}^{\max\_depth(h_1)} \sum_{D_2=0}^{\max\_depth(h_2)} \sum_{I_1=0}^{pq} \sum_{I_2=0}^{pq} CM(D_1,I_1,D_2,I_2,h_1,h_2)} \tag{7.3}$$

□

Definicja 7.6 Hiperschemat jest drzewem złożonym z węzłów wewnętrznych nad zbiorem $F \cup \{=,\#\}$ i liści ze zbioru $T \cup \{=,\#\}$, gdzie *F* i *T* są odpowiednio zborem funkcji i terminali. Operator = to symbol „don't care", który zastępuje dokładnie jeden węzeł, terminalny operator # to symbol „don't care", który zastępuje dowolne poddrzewo, natomiast operator funkcyjny # to symbol „don't, care" który zastępuje dowolny węzeł funkcyjny o liczbie potomków nie mniejszej niż liczba poddrzew do niej podpiętych.

□

Dla krzyżowania hierarchicznego możemy przyjąć:

F={1,...,d}

T={0,1}

W dalszej części rozdziału przyjęliśmy upraszczające zapis założenie, że węzły będą reprezentowane poprzez pojedyncze indeksy zamiast par w kartezjańskim układzie współrzędnych. Możemy takie założenie zrobić, jako ze istnieje odwzorowanie jeden do jeden pomiędzy liczbami naturalnymi a parami współrzędnych. Oznaczmy to odwzorowanie przez *map*.

Niech *M* będzie liczbą osobników w populacji,

niech $E(m(H,t+1)$ będzie oczekiwaną liczbą osobników pasujących do schematu H w epoce *t+1,*

niech *p(H,t)* będzie prawdopodobieństwem wybrania osobnika pasującego do schematu *H,*

niech *f(H,t)* będzie średnim przystosowaniem osobników pasujących do schematu *H,*

niech $\bar{f}(t)$ będzie średnim przystosowaniem wszystkich osobników w populacji.

Zachodzi następująca równość:

$$p(H,t) = \frac{m(H,t)f(H,t)}{M\bar{f}(t)}$$

Niech p_{x_0} będzie prawdopodobieństwem krzyżowania, oraz niech *p(H,t)* będzie prawdopodobieństwem wybrania osobnika pasującego do schematu *H.*

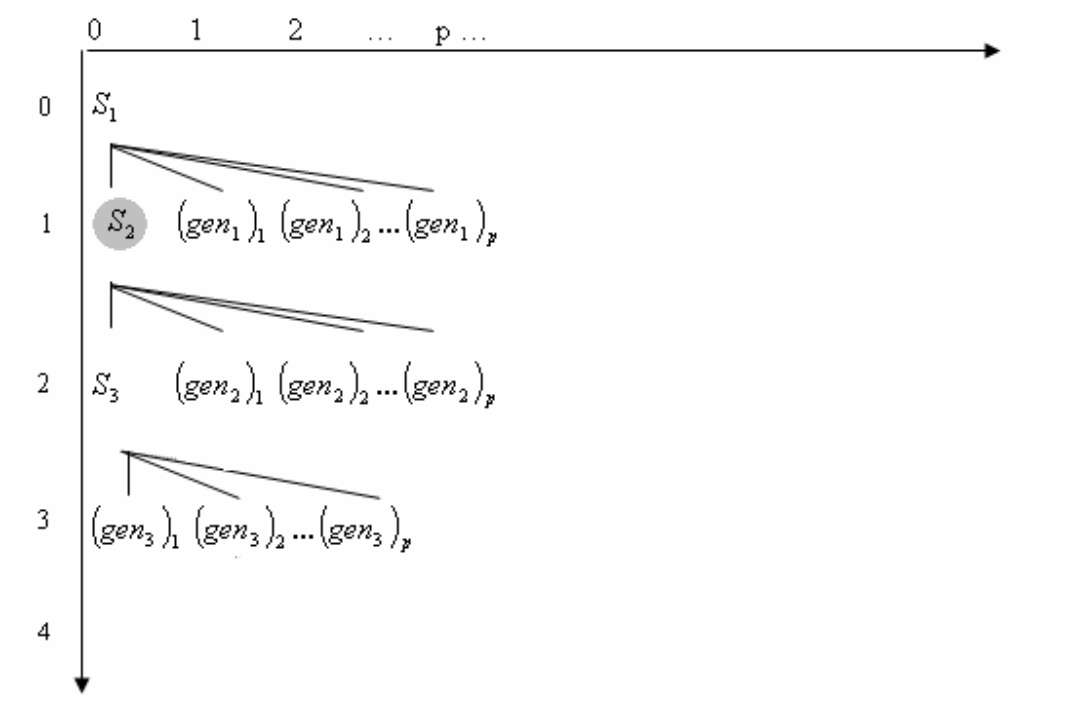
Niech $p(h_1,t),p(h_2,t)$ będzie prawdopodobieństwami wybrania rodziców h_1, h_2 odpowiednio .Niech $p(i,j \mid h_1,h_2) = p_{hc}(map(i),map(j) \mid h_1,h_2)$.

Niech $L(H,i,j)$ będzie hiperschematem powstałym poprzez wstawienie do pustego systemu referencyjnego podschematu *H* w taki sposób, aby korzeń *i* miał współrzędną *j*, a następnie zastąpienie wszystkich węzłów na ścieżce pomiędzy korzeniem a węzłem *i* przez węzeł = oraz wszystkich poddrzew tych węzłów przez węzeł #.

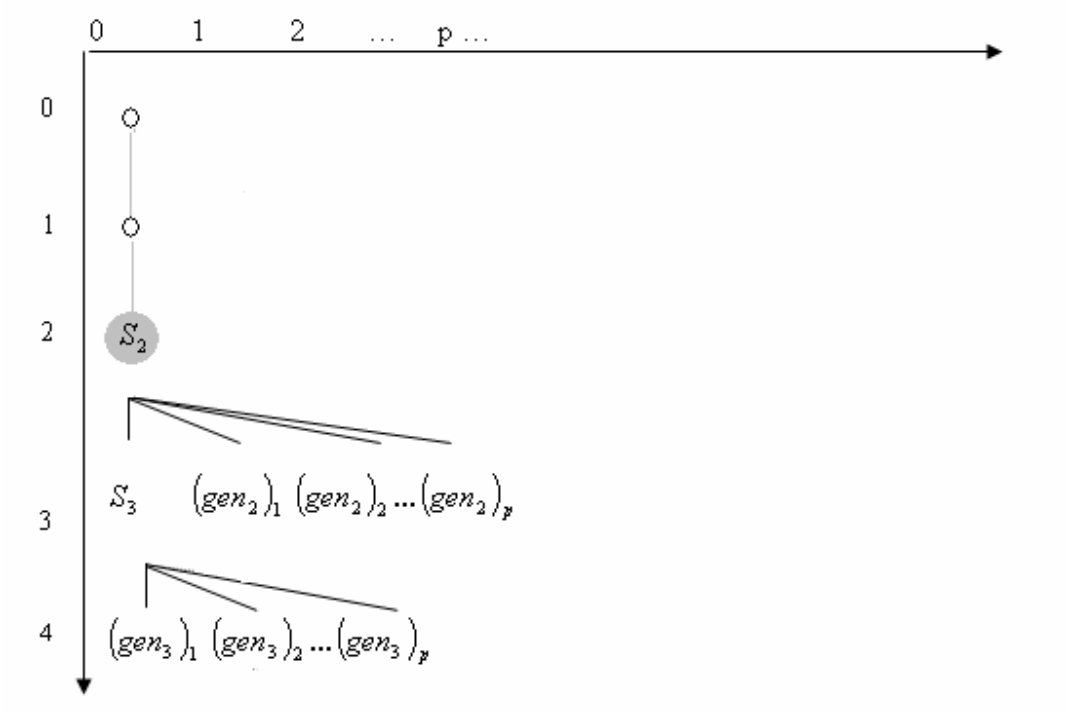
Niech $U(H,i)$ będzie hiperschematem powstałym poprzez zastąpienie poddrzewa poniżej punktu krzyżowania *i* przez #.

Hiperschematy $L(H,i,j)$ oraz $U(H,i)$ mają istotne znaczenie, ponieważ krzyżując w punkcie i osobnika z $U(H,i)$ z dowolnym osobnikiem z $L(H,i,j)$ w punkcie *j* zawsze otrzymamy osobnika należącego do schematu *H.*

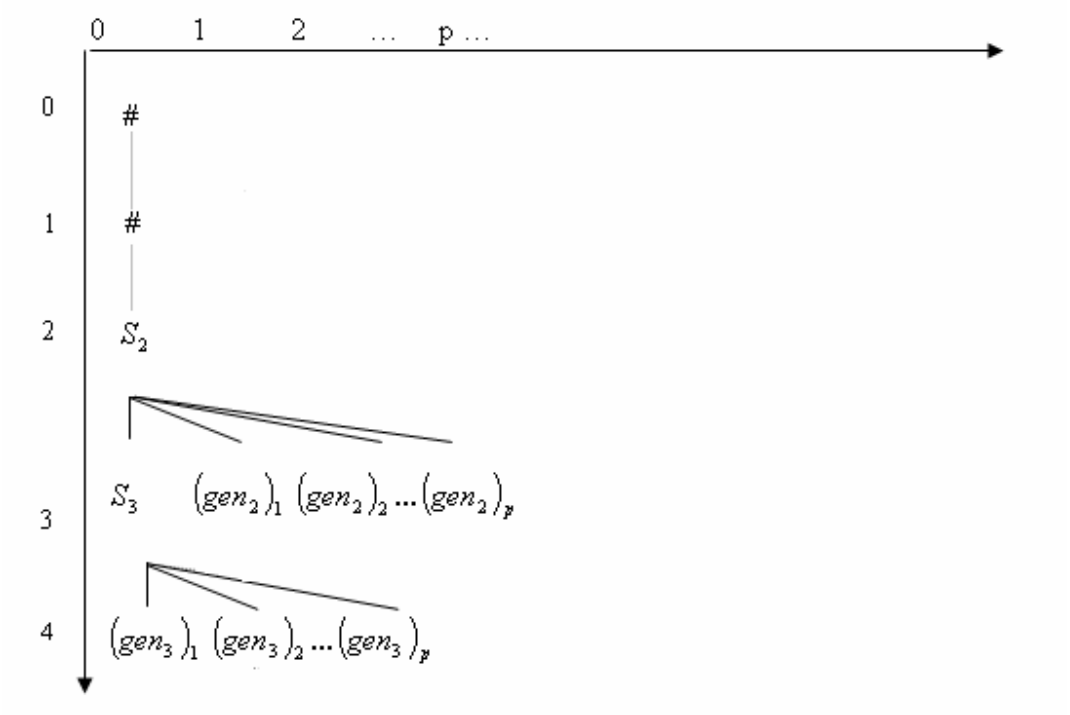
Sposób konstrukcji przykładowego hiperschematu $L(H,i,j)$ dla schematu *H* z Rys. 7.5. został przedstawiony na Rys. 7.6-7.8



Rys. 7.6 Etap pierwszy tworzenia hiperschematu L(H,(0,1),(0,2)).

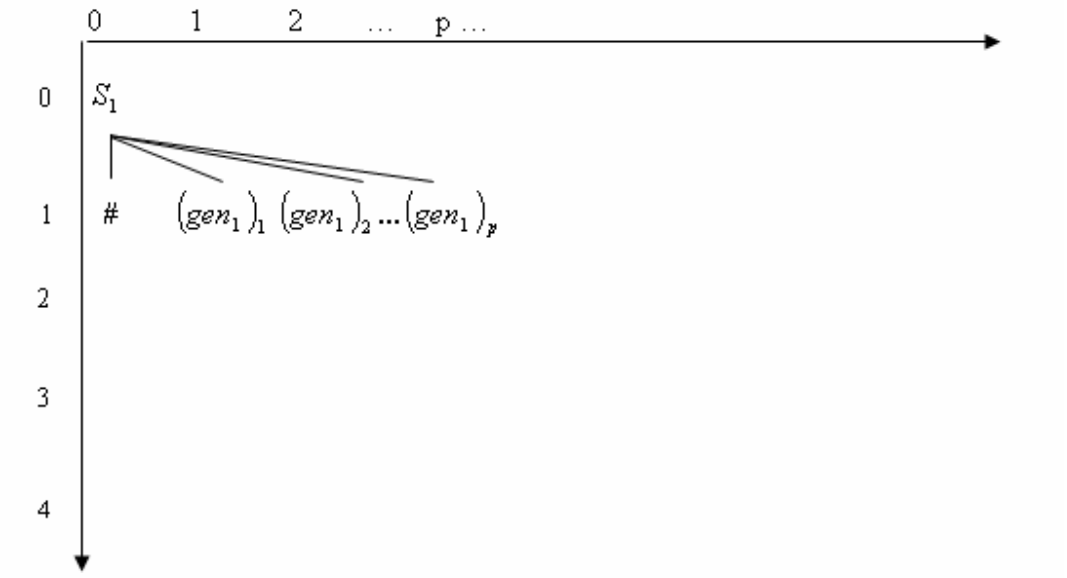


Rys. 7.7 Etap drugi tworzenia hiperschematu L(H,(0,1),(0,2)).



Rys.7.8 Ostatni etap tworzenia hiperschematu L(H,(0,1),(0,2)).

Na Rys. 7.9 został przedstawiony przykładowy hiperschemat $U(H,i)$ dla schematu H z Rys. 7.5.



Rys. 7.9 Hiperschemat U(H,(0,1)) dla schematu.

Twierdzenie 7.1

Prawdopodobieństwo, że w epoce t nowo stworzony osobnik pasuje do schematu H o stałym, określonym rozmiarze i kształcie, przy krzyżowaniu hierarchicznym i braku mutacji wynosi

$$\alpha(H,t) = (1 - p_{x_0})\, p(H,t) + p_{x_0} \sum_{h_1} \sum_{h_2} p(h_1,t)\, p(h_2,t) \tag{7.4}$$

$$\times \sum_{i \in H} \sum_{j} p(i, j \mid h_1, h_2)[h_1 \in U(H,i)][h_2 \in L(H,i,j)]$$

W twierdzeniu 7.1 mamy równość (zamiast nierówności występującej w twierdzeniach dla większości teorii schematów). Równość ta wynika z faktu, że jedyną możliwością otrzymania poprzez operację krzyżowania osobnika należącego do schematu H jest sytuacja, gdy krzyżujemy w punkcie i osobnika z $U(H,i)$ z dowolnym osobnikiem z $L(H,i,j)$ w punkcie j.

Otrzymany wynik można wykorzystać do obliczenia efektywnego przystosowania dla algorytmu genetycznego z hierarchicznym krzyżowaniem wiedząc, że efektywne przystosowanie schematu jest zdefiniowane przez następujące równanie:

$$E\left[\frac{m(H,t+1)}{M}\right] = \frac{m(H,t)}{M} \frac{f_{eff}(H,t)}{\bar{f}(t)} \tag{7.5}$$

przy założeniu, że została użyta selekcja proporcjonalna do przystosowania.

Powyższa definicja efektywnego przystosowania zaproponowana przez Ch. Stephensa oraz H. Walebroeka uwzględnia istotny wpływ operatorów genetycznych (w naszym przypadku krzyżowania) na efektywny krajobraz, w którym populacja ewaluuje. Krajobraz skojarzony tylko z selekcją niewiele mówi o prawdziwej ewolucji populacji. W rzeczywistości do populacji mogą należeć nawet osobniki o zerowym przystosowaniu.

Ponieważ $E\left[\frac{m(H,t+1)}{M}\right] = \alpha(H,t)$ oraz $\frac{m(H,t)}{M\bar{f}(t)} = \frac{p(H,t)}{f(H,t)}$ otrzymujemy:

$$f_{eff}(H,t) = \frac{\alpha(H,t)}{p(H,t)} f(H,t) \tag{7.6}$$

Efektywne przystosowanie schematu H w epoce t ($f_{eff}(H,t)$ można interpretować jako wartość przystosowania w chwili t wymagana do zwiększenia (lub zmniejszenia) wartości $\frac{m(H,t)}{M}$ dla algorytmu z samą selekcją o taką samą wartość, jak dla algorytmu operatorów uwzględnieniem wszystkich operatorów genetycznych.

Jeśli $f_{eff}(H,t) > f(H,t)$ to operatory genetyczne inne niż selekcja zwiększają sukces reprodukcji osobników należących do schematu *H*.

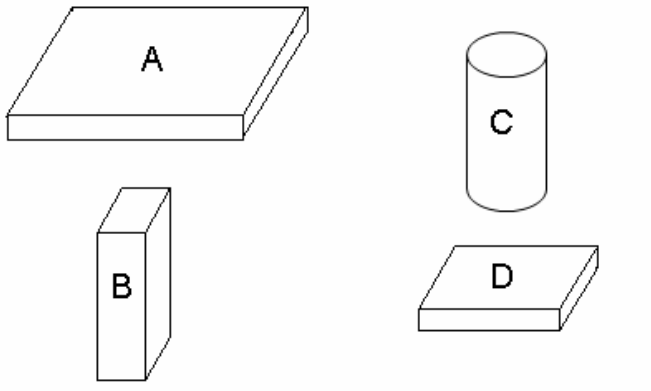
Twierdzenie 7.2

Efektywne przystosowanie dla schematu *H* o stałym kształcie i rozmiarze, z krzyżowaniem hierarchicznym i przy braku mutacji dane jest wzorem:

$$f_{eff}(H,t) = \frac{f(H,t)}{p(H,t)} * \left((1-p_{x_0})p(H,t) + p_{x_0} \sum_{h_1} \sum_{h_2} p(h_1,t)p(h_2,t) \sum_{i \in H} \sum_{j} p(i,j \mid h_1,h_2)[h_1 \in U(H,i)][h_2 \in L(H,i,j)] \right) \tag{7.7}$$

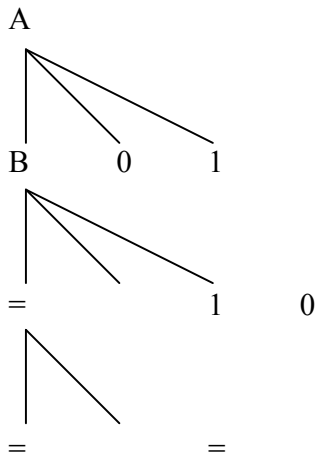
W ostatniej części rozdziału przedstawiony został przykład zastosowania teorii schematów Polie'go dla algorytmu genetycznego z hierarchicznym chromosomem.

Przykład 7.1 Algorytm genetyczny z hierarchicznym chromosomem będzie użyty do projektowania kolumn. Zakłada się, że składowe kolumny mogą należeć do klas *A, B, C* oraz *D.* Przykłady składowych należących do poszczególnych klas przedstawione zostały na Rys. 7.10.



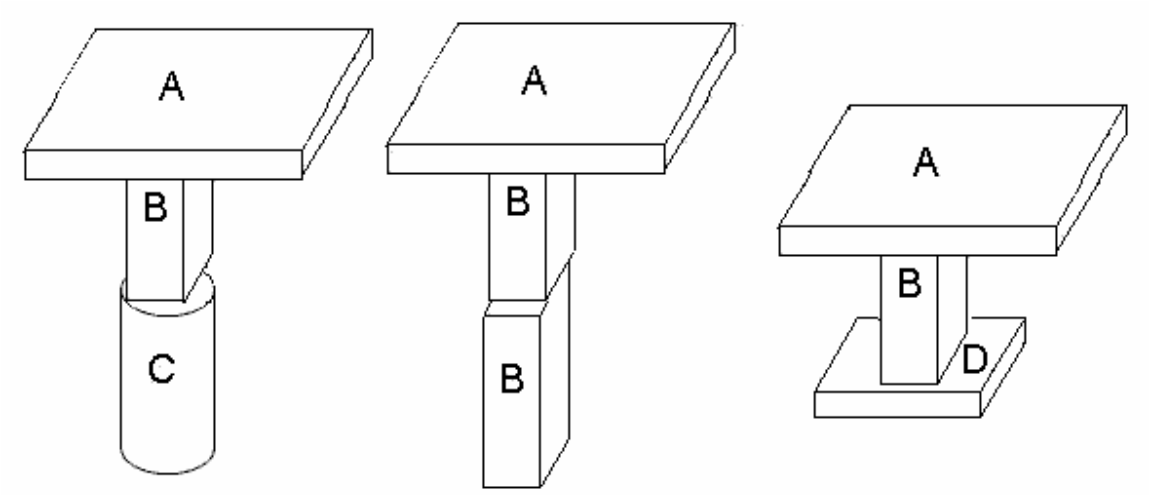
Rys. 7.10 Przykłady składowych należących do klas *A, B, C, D*.

Niech schemat *H* będzie postaci jak na Rys.7.11:



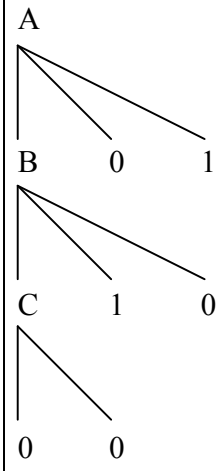
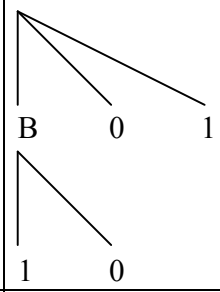
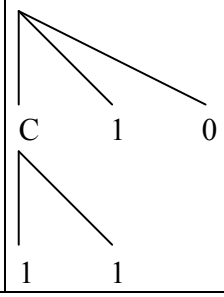
Rys. 7.11 Schemat *H*.

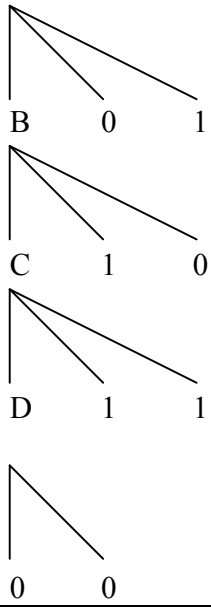
Na Rys. 7.12 przedstawione zostały przykładowe artefakty należące do schematu *H*.



Rys. 7.12 Przykładowe artefakty należących do schematu *H*.

Niech populacja w chwili *t* będzie składała się z osobników przedstawionych w Tab. 7.1.

numer osobnika	**osobnik**	**liczba wystąpień**
d_1		4
d_2		2
d_3		2

numer osobnika	osobnik	liczba wystąpień
d_4	A  B 0 1 C 1 0 D 1 1 0 0	6

Tab. 7.1 Populacja w epoce *t*

W Tab. 7.2 przedstawione zostały niezerowe wartości rozkładu prawdopodobieństwa wyboru punktów krzyżowania *i* u osobnika h_1 oraz *j* u osobnika h_2, obliczone na podstawie wzoru 7.3.

h_1	h_2	i	j	$P_{scgp}(i, j \mid h_1, h_2)$
d_1	d_1	(0,0)	(0,0)	$\frac{1}{3}$
d_1	d_1	(1,0)	(1,0)	$\frac{1}{3}$
d_1	d_1	(2,0)	(2,0)	$\frac{1}{3}$
d_2	d_2	(0,0)	(0,0)	$\frac{1}{2}$
d_2	d_2	(1,0)	(1,0)	$\frac{1}{2}$
d_3	d_3	(0,0)	(0,0)	$\frac{1}{2}$
d_3	d_3	(1,0)	(1,0)	$\frac{1}{2}$
d_4	d_4	(0,0)	(0,0)	$\frac{1}{4}$

d_4	d_4	(1,0)	(1,0)	$\frac{1}{4}$
d_4	d_4	(2,0)	(2,0)	$\frac{1}{4}$
d_4	d_4	(3,0)	(3,0)	$\frac{1}{4}$
d_1	d_2	(0,0)	(0,0)	$\frac{1}{2}$
d_1	d_2	(1,0)	(1,0)	$\frac{1}{2}$
d_1	d_3	(1,0)	(0,0)	$\frac{1}{2}$
d_1	d_3	(2,0)	(1,0)	$\frac{1}{2}$
d_1	d_4	(0,0)	(0,0)	$\frac{1}{3}$
d_1	d_4	(1,0)	(1,0)	$\frac{1}{3}$
d_1	d_4	(2,0)	(2,0)	$\frac{1}{3}$
d_2	d_1	(0,0)	(0,0)	$\frac{1}{2}$
d_2	d_1	(1,0)	(1,0)	$\frac{1}{2}$
d_2	d_3	(1,0)	(0,0)	1
d_2	d_4	(0,0)	(0,0)	$\frac{1}{2}$
d_2	d_4	(1,0)	(1,0)	$\frac{1}{2}$
d_3	d_1	(0,0)	(1,0)	$\frac{1}{2}$
d_3	d_1	(1,0)	(2,0)	$\frac{1}{2}$
d_3	d_2	(0,0)	(1,0)	1
d_3	d_4	(0,0)	(1,0)	$\frac{1}{2}$
d_3	d_4	(1,0)	(2,0)	$\frac{1}{2}$

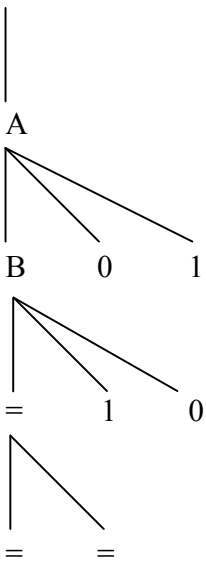
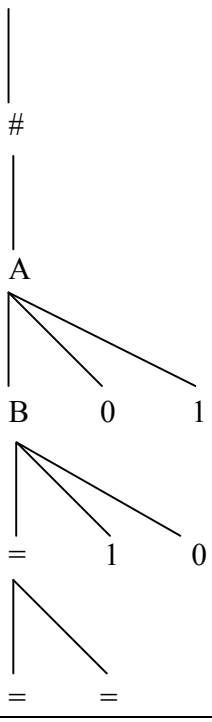
d_4	d_1	(0,0)	(0,0)	$\frac{1}{3}$
d_4	d_1	(1,0)	(1,0)	$\frac{1}{3}$
d_4	d_1	(2,0)	(2,0)	$\frac{1}{3}$
d_4	d_2	(0,0)	(0,0)	$\frac{1}{2}$
d_4	d_2	(1,0)	(1,0)	$\frac{1}{2}$
d_4	d_3	(1,0)	(0,0)	$\frac{1}{2}$
d_4	d_3	(2,0)	(1,0)	$\frac{1}{2}$

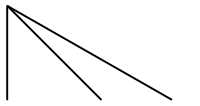
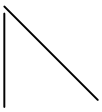
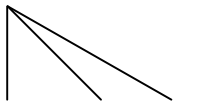
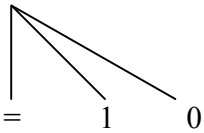
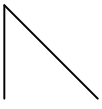
Tab. 7.2 Niezerowe wartości rozkładu prawdopodobieństwa wyboru punktów krzyżowania *i* oraz *j* u osobników h_1 oraz h_2

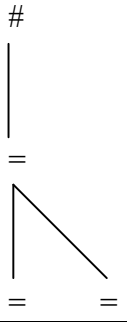
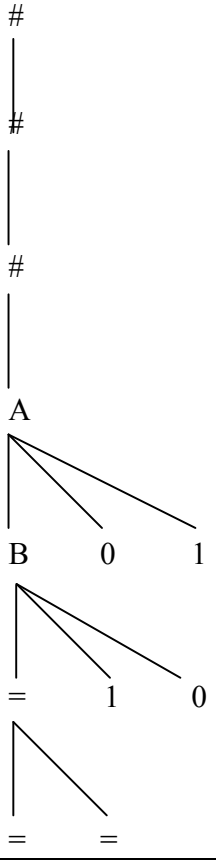
W Tab. 7.3 oraz 7.4 przedstawione zostały hiperschematy *U(H,i)* oraz *L(H,i,j)* dla wybranych węzłów *i* oraz *j*. Uwzględnione zostały tylko te węzły, które z prawdopodobieństwem większym od zera mogą zostać wybrane jako punkty krzyżowania.

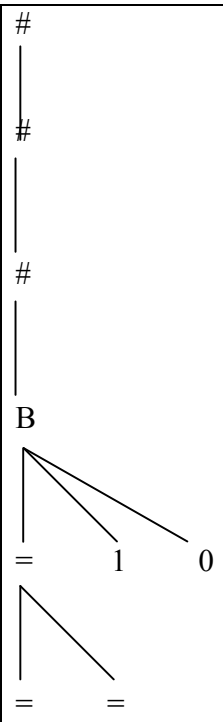
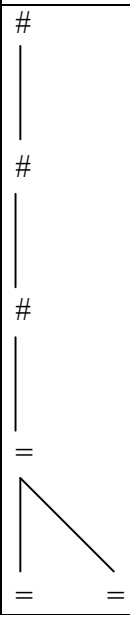
i	***U(H,i)***
(0,0)	#
(1,0)	A # 0 1
(2,0)	A B 0 1 # 1 0

Tab. 7.3 Hiperschematy *U(H,i)*

i	***j***	***L(H,i,j)***
(0,0)	(0,0)	*H*
(0,0)	(1,0)	
(0,0)	(2,0)	

(1,0)	(0,0)	B  = 1 0  = =
(2,0)	(0,0)	=  = =
(1,0)	(1,0)	# B  = 1 0  = =
(1,0)	(2,0)	# # B  = 1 0  = =

(2,0)	(1,0)	
(2,0)	(2,0)	
(0,0)	(3,0)	

(1,0)	(3,0)	
(2,0)	(3,0)	

Tab. 7.2 Hiperschematy *L(H,i,j)*

Dla schematu *H* z Rys. 7.11 i populacji początkowej podanej w Tab. 7.1, po wyznaczeniu hiperschematów *U(H,i), L(H,i,j)* oraz rozkładu prawdopodobieństwa wyboru punktów krzyżowania, ze wzoru 7.4 obliczono prawdopodobieństwo, że w epoce *t* nowo stworzony osobnik pasuje do schematu *H* o stałym, określonym rozmiarze i kształcie, przy krzyżowaniu hierarchicznym i braku mutacji.

Prawdopodobieństwo to wynosi:

$$\alpha(H,t) = \frac{4}{14}\frac{4}{14}1 + \frac{2}{14}\frac{2}{14}(0+0) + \frac{2}{14}\frac{2}{14}(0+0) + \frac{6}{14}\frac{6}{14}(0+0+0) + \frac{4}{14}\frac{2}{14}(0+0) + \frac{4}{14}\frac{2}{14}\left(\frac{1}{2}+\frac{1}{2}\right) +$$
$$+ \frac{4}{14}\frac{6}{14}(0+0+0) + \frac{2}{14}\frac{4}{14}\left(\frac{1}{2}+\frac{1}{2}\right) + \frac{2}{14}\frac{2}{14}1 + \frac{2}{14}\frac{2}{14}(0+0) + \frac{2}{14}\frac{4}{14}(0+0) + \frac{2}{14}\frac{2}{14}(0) + \frac{2}{14}\frac{6}{14}(0+0) +$$
$$+ \frac{6}{14}\frac{4}{14}\left(\frac{1}{3}+\frac{1}{3}+\frac{1}{3}\right) + \frac{6}{14}\frac{2}{14}(0+0) + \frac{6}{14}\frac{2}{14}\left(\frac{1}{2}+\frac{1}{2}\right) = \frac{72}{14^2} \approx 0{,}36734693877551$$

Efektywne przystosowanie obliczone ze wzoru 7.7 wynosi:

$$f_{eff}(H,t) = \frac{\alpha(H,t)}{p(H,t)} f(H,t) = \frac{\alpha(H,t)}{\frac{4}{14}} f(H,t) \approx 1{,}2857 f(H,t)$$

Wniosek 7.1 Dla zadanej populacji przedstawionej w Tab. 7.1 krzyżowanie hierarchiczne zwiększa sukces reprodukcji osobników należących do schematu *H* z Rys. 7.11.

Wynika to z faktu, że $f_{eff}(H,t) > f(H,t)$.

8 Zorientowana obiektowo aplikacja oparta na algorytmie genetycznym z hierarchicznym chromosomem

Rozdział przedstawia pewne statyczne i dynamiczne aspekty systemu zorientowanego obiektowo, za pomocą diagramów zaprojektowanych przy użyciu języka UML.
Wśród diagramów statycznych wyróżniamy:
- diagramy komponentów,
- diagramy klas.

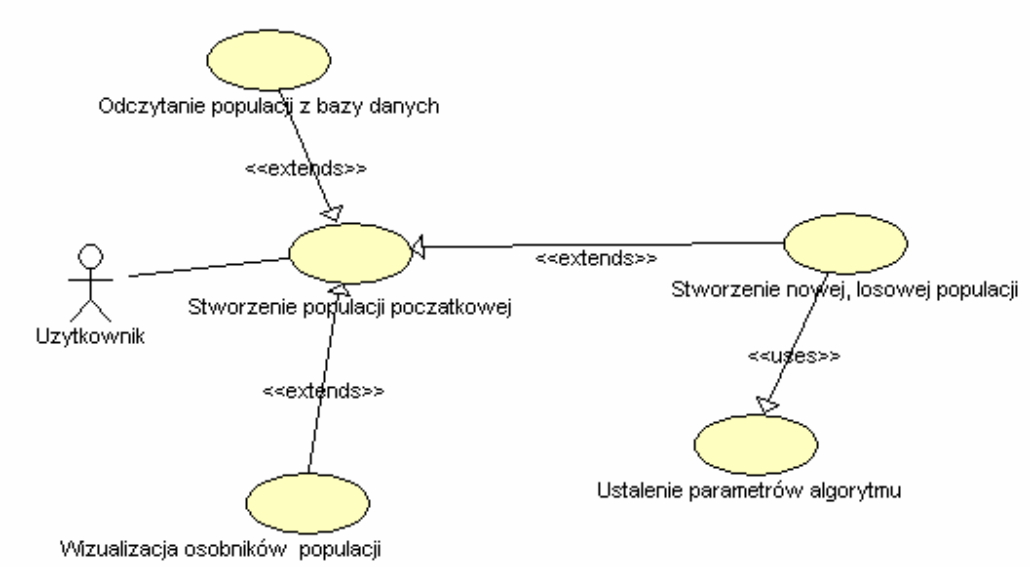
Wśród diagramów dynamicznych wyróżniamy:
- diagramy przypadków,
- diagramy interakcji (sekwencyjne i kolaboracji),
- diagramy przejść stanów.

Prezentowany system został zaprojektowany na bazie założeń Rational Unified Process [14]. Na początku zostały utworzone diagramy przypadków użycia opisujące pewne główne scenariusze działania aplikacji. Następnie zaprojektowane zostały diagramy klas, definiujące metody i atrybuty klas konieczne do realizacji scenariuszy opisanych przez diagramy przypadków użycia. W następnym etapie opracowane zostały diagramy dynamiczne, takie jak diagramy interakcji oraz diagramy przejść stanów. Diagramy te zostały użyte do testowania pewnych dynamicznych zachowań klas zdefiniowanych na diagramach statycznych. Po analizie diagramów dynamicznych, diagramy statyczne zostały poprawione. W końcowym etapie zostały wygenerowane nagłówki klas za pomocą narzędzi BoUML. Wygenerowany kod został rozwinięty za pomocą narzędzia Visual C++.

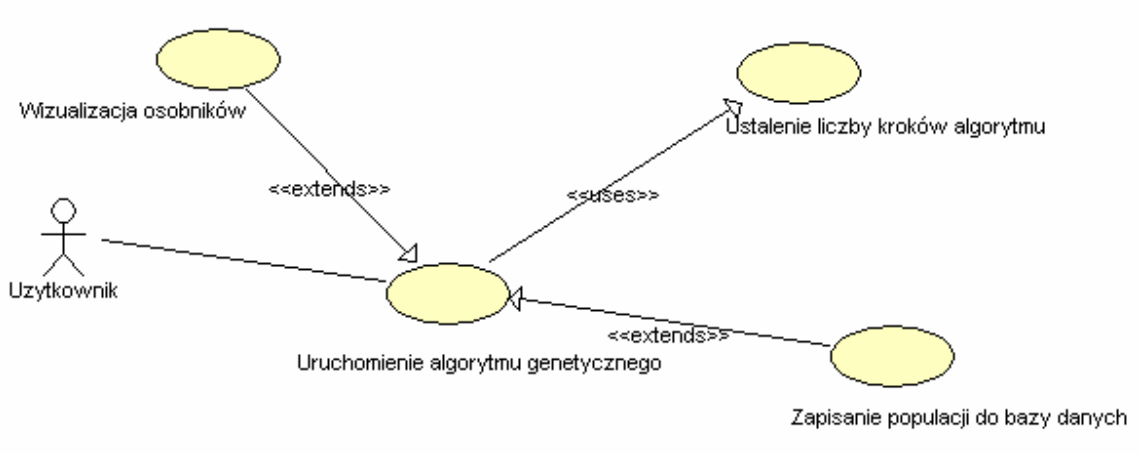
8.1 Diagramy przypadków użycia

Wymagania interfejsu użytkownika

Na Rys. 8.1. oraz Rys. 8.2. zostały przedstawione diagramy przypadków użycia opisujące wymagania interfejsu użytkownika.



Rys. 8.1 Diagram przypadków użycia opisujący proces tworzenia populacji początkowej.



Rys. 8.2 Diagram przypadków użycia opisujący proces uruchomienia algorytmu genetycznego.

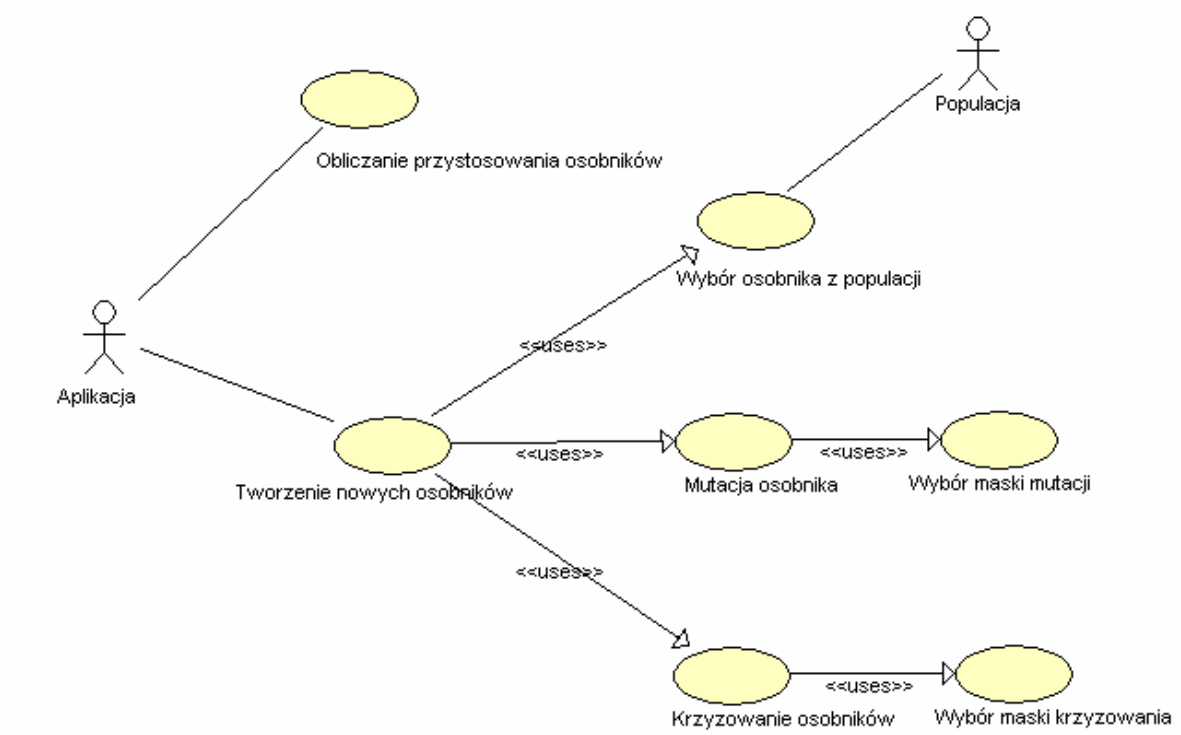
Scenariusz 8.1

Użytkownik wybiera populację początkową dla algorytmu genetycznego. Populacja ta może zostać odczytana z bazy danych lub zostać stworzona losowo. Przy tworzeniu populacji losowej użytkownik podaje liczbę osobników populacji oraz maksymalną dopuszczalną liczbę

składowych osobnika. Użytkownik może obejrzeć osobniki populacji używając modułu graficznego aplikacji. Następnie użytkownik ustala liczbę kroków algorytmu. Po zdefiniowaniu wartości wszystkich powyższych parametrów, algorytm genetyczny zostaje uruchomiony przez użytkownika.

Wymagania aplikacji

Na Rys. 8.3 przedstawiony został diagram przypadków użycia opisujący wymagania aplikacji.



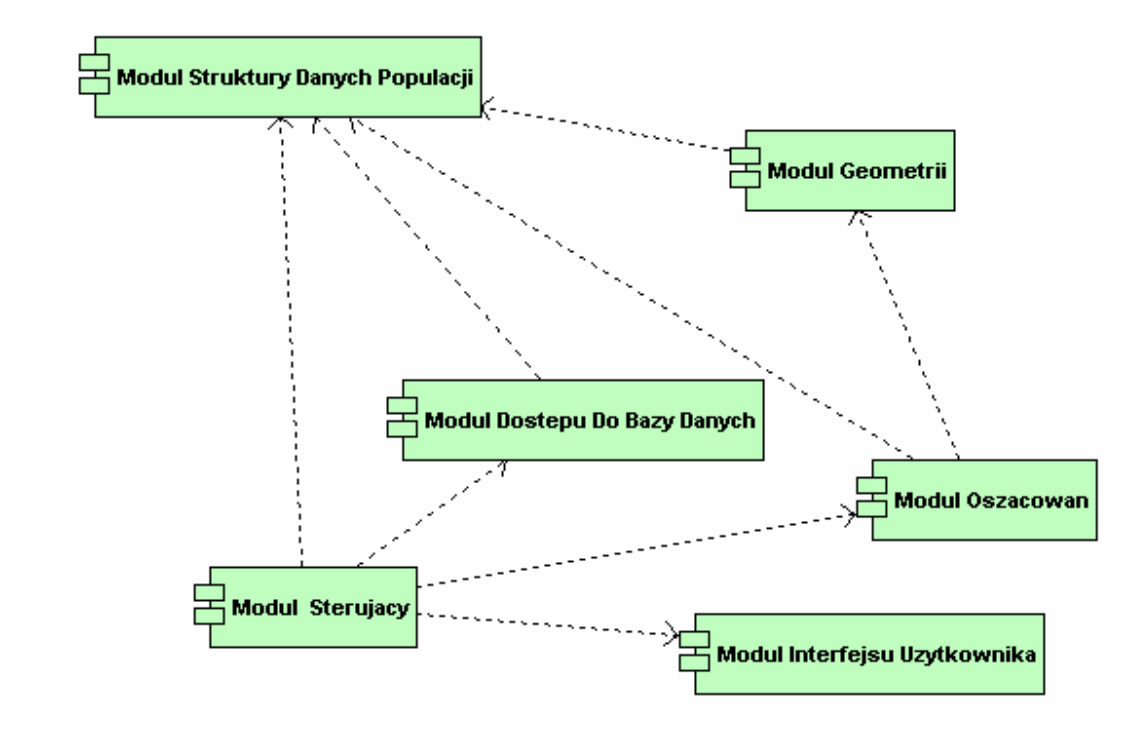
Rys. 8.3 Diagram przypadków użycia opisujący wymagania aplikacji.

Scenariusz 8.2

Aplikacja może obliczyć wartość przystosowania istniejących osobników oraz utworzyć nowe osobniki. Aby utworzyć nowe osobniki populacja wybiera osobniki przeznaczone do reprodukcji a następnie wykonuje mutację oraz krzyżowanie – zgodnie z wybranymi maskami mutacji i krzyżowania.

8.2 Komponenty

Aby spełnić wymagania podane w Rozd. 8.2 zaproponowany został następujący podział aplikacji.



Rys. 8.4 Diagram komponentów.

Aplikacja została podzielona na następujące moduły: Moduł Sterujący, Moduł Oszacowań, Moduł Geometrii, Moduł Dostępu Do Bazy Danych, Moduł Interfejsu Użytkownika oraz Moduł Struktury Danych Populacji, co przedstawiono na Rys. 8.4.

Wszystkie moduły zostały zaimplementowane jako biblioteki ładowane dynamicznie (DLL - Dynamic Link Libraries). Jedynie Moduł Interfejsu Użytkownika został zaimplementowany jako plik wykonywalny. Relacje przedstawione na powyższym diagramie odpowiadają zależnościom pomiędzy modułami.

Nazwa modułu:

Moduł Struktury Danych Populacji

Opis:

Moduł składa się z klas reprezentujących strukturę danych populacji

Klasy:

PopulacjaZewnętrzna, PopulacjaWewnętrzna, OsobnikDojrzały, OsobnikNiedojrzały, Prymityw, Gen

Nazwa modułu:

Moduł Oszacowań

Opis:

Moduł składa się z klas odpowiedzialnych za obliczanie wartości funkcji przystosowania osobników populacji

Klasy:

OszacowanieGlobalne, OszacowanieUogólnione, Poziom, Fragmentacja, Rozmiar, Siedzisko, Podłoga, Środek

Nazwa modułu:

Moduł Geometrii

Opis:

Moduł składa się z klas opisujących geometrię artefaktów (osobników), które są wykorzystywane przez moduł Oszacowan do obliczenia wartości funkcji przystosowania

Klasy:

Ściana, kontur3d, kontur2d, krawedź3d, krawędź2d, punkt3d, punkt2d

Nazwa modułu:

Moduł Sterujący

Opis:

Moduł składa się z klasy odpowiadającej za działanie algorytmu genetycznego

Klasy:

Aplikacja

Nazwa modułu:

Moduł Dostępu Do Bazy Danych

Opis:

Moduł jest odpowiedzialny za zdefiniowanie interfejsu bazy danych. Jest on używany do przechowywania informacji o osobnikach populacji w poszczególnych epokach. Dane zapisane w bazie danych mogą być danymi wejściowymi przy kolejnym uruchomieniu aplikacji

Klasy:

ZarządcaBazyDanych

Nazwa modułu:

Moduł Interfejsu Użytkownika

Opis:

Moduł używany do uruchomienia algorytmu genetycznego oraz zdefiniowania jego parametrów

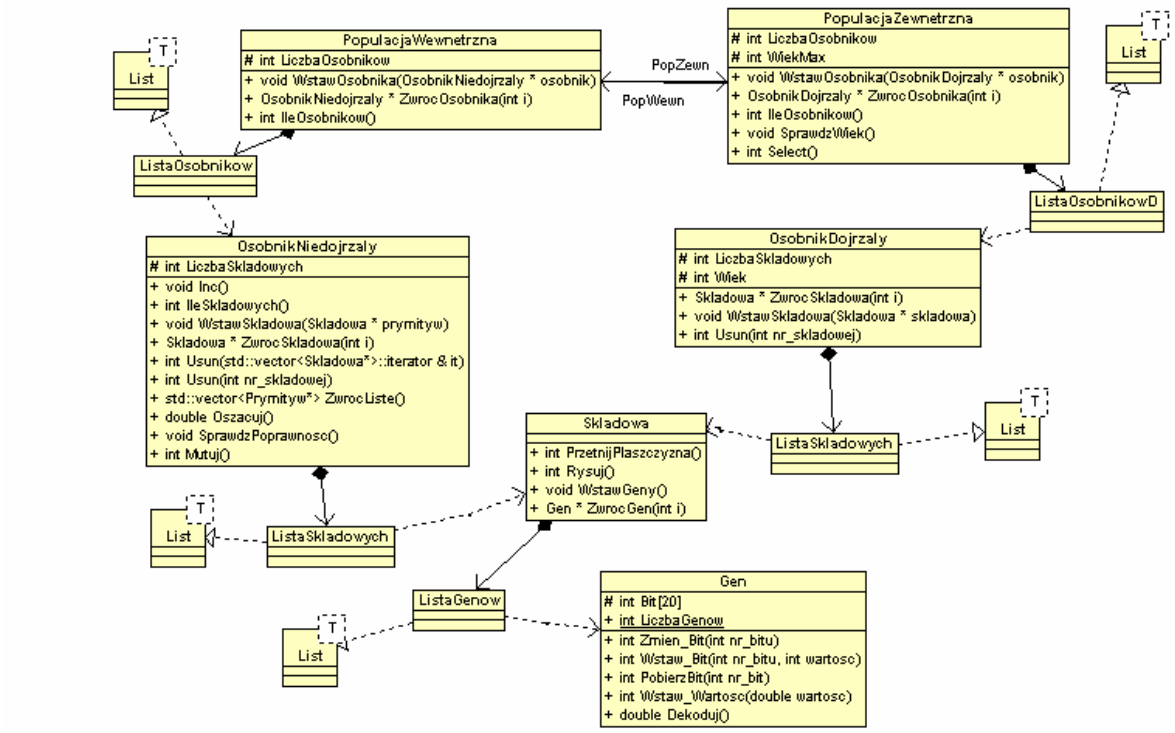
Klasy:

Aplikacja z wieloma klasami wygenerowanymi przez Visual C++ definiującymi graficzny interfejs użytkownika.

8.3 Diagramy klas

W rozdziale tym przedstawione zostaną diagramy klas wybranych modułów. Opisane zostaną również podstawowe metody poszczególnych klas.

Na Rys.8.5 przedstawiony został diagram klas modułu opisującego strukturę danych populacji.



Rys. 8.5 Klasy modułu Struktura Danych.

Najważniejszą częścią aplikacji stanowi moduł Struktury Danych Populacji, odpowiedzialny za zdefiniowanie modelu populacji. Moduł ten został zaprojektowany za pomocą paradygmatu „bottom-up". Model populacji składa się z bazowych elementów, takich jak klasa Gen, oraz bardziej złożonych klas, takich jak Składowa, która jest utworzona z elementów bazowych, oraz – jeszcze bardziej złożonych klas, takich jak OsobnikDojrzały i OsobnikNiedojrzały.

Klasa:

PopulacjaWewnetrzna

Najważniejsze metody:

void WstawOsobnika(OsobnikDojrzały* osobnik);

Wstawia osobnika do listy uporządkowanej rosnąco zgodnie z wartościami przystosowania osobników.

OsobnikDojrzały* ZwróćOsobnika(int i);

Zwraca wskaźnik do osobnika

int IleOsobników();

Zwraca liczbę osobników w populacji

Uwagi:

PopulacjaWewnętrzna składa się z osobników z już obliczonymi wartościami przystosowania. Spośród nich wybierane są osobniki do krzyżowania i mutacji.

Klasa:

PopulacjaZewnętrzna

Najważniejsze metody:

void WstawOsobnika(OsobnikNiedojrzały* osobnik);

Wstawia do listy nieuporządkowanej nowo utworzonego osobnika OsobnikNiedojrzały*

ZwróćOsobnika(int i);

Zwraca wskaźnik do osobnika

int IleOsobnikow();

Zwraca liczbę osobników w populacji

Uwagi:

PopulacjaZewnętrzna składa się z osobników (potomków) nowo utworzonych w wyniku operacji rekombinacji. W zależności od wybranej strategii, albo wszystkie osobniki, albo tylko najlepiej przystosowane zostaną umieszczone w PopulacjiWewnętrznej i staną się kandydatami do dalszej reprodukcji.

Klasa:

OsobnikNiedojrzaly

Najważniejsze metody:

void WstawSkładową(Składowa* składowa);

Składowa* ZwróćSkładową(int i);

int Usuń(std::vector<Składowa*>::iterator& it);

double Oszacuj();//oblicza globalny fitness

int Mutuj();

Uwagi:

OsobnikNiedojrzały to nowo utworzony osobnik, który nie ma jeszcze obliczonej wartości funkcji przystosowania

Klasa:

OsobnikDojrzały

Najważniejsze metody:

Składowa* ZwróćSkładową(int i);

int Usuń(int nr_skladowa);

void WstawSkładową(Składowa* składowa);

Uwagi:

OsobnikDojrzały to osobnik, który już posiada obliczoną wartość przystosowania. W zależności od wartości przystosowania oraz od użytej strategii, zostanie on usunięty lub stanie się jednym z kandydatów, spośród których wybierane zostaną osobniki do reprodukcji w następnej epoce.

Klasa:

Składowa

Najważniejsze metody:

void WstawGeny();

Gen* ZwrócGen(int i);

Uwagi:

Każda składowa zawiera listę genów (parametrów opisujących jej geometrię). Do podstawowych operacji należą operację zmieniające wartości tych parametrów.

Klasa:

Gen

Najważniejsze metody:

int Pobierz_Bit(int nr_bitu);

int Zmien_Bit(int nr_bitu);

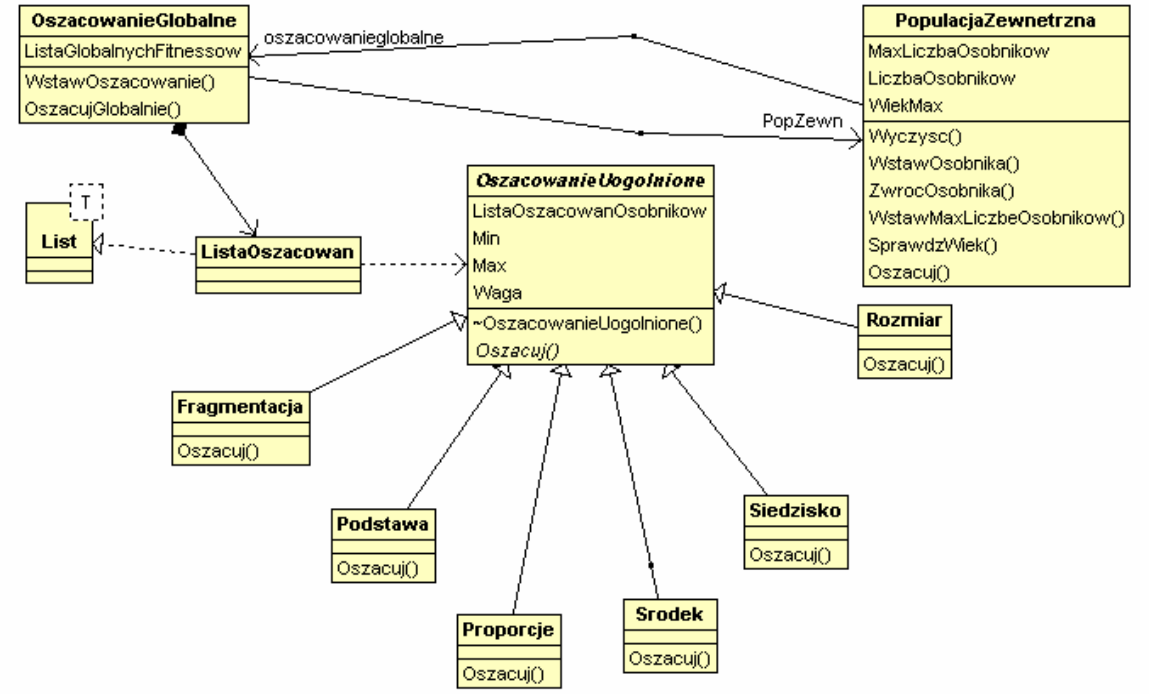
int Wstaw_Bit(int nr_bitu, int wartość);

Uwagi:

Podstawowym polem tej klasy jest tablica bitów kodujących wartość genu.

Podstawowymi operacjami tej klasy są metody działające na tablicy bitów.

Drugim istotnym modułem jest moduł Oszacowań. Diagram klas tego modułu przedstawiony jest na Rys 8.6.



Rys. 8.6 Diagram klas modułu Moduł Oszacowań.

Aplikacja została zaprojektowana jako otwarta. Utworzona została klasa OszacowanieUogólnione. Klasa ta poprzez użycie metod wirtualnych umożliwia rozszerzanie systemu poprzez definiowanie nowych kryteriów oszacowań. Ze względu na przykładowe zastosowanie aplikacji – projektowanie platformy oraz projektowanie krzeseł - zaimplementowane zostały następujące kryteria : fragmentacja, rozmiar, proporcje, siedzisko, podstawa, środek.

Klasa:

OszacowanieGlobalne

Najważniejsze metody:

void WstawOszacowanie(OszacowanieUogólnione oszacowanie);

OszacujGlobalnie();

Uwagi:

Klasa OszacowanieGlobalne odpowiedzialna jest za obliczenie globalnej wartości przystosowania każdego osobnika z populacji. W tym celu wywoływana jest metoda Oszacuj wszystkich obiektów znajdujących się na liście ListaOszacowań. W celu obliczenia globalnej wartości przystosowania osobników została zastosowana metoda SWGR (ang. Sum of Weighted Global Ratios).

Klasa:

OszacowanieUogólnione

Najważniejsze metody:

Oszacuj ();

Uwagi:

Klasa OszacowanieUogolnione zapewnia otwartość aplikacji – umożliwia dodawanie nowych kryteriów oszacowań.

Klasa:

Fragmentacja

Najważniejsze metody:

Oszacuj ();

Uwagi:

Klasa Fragmentacja to jedno z zaimplemenetowanych kryteriów oszacowań.

Metoda Oszacuj() klasy Fragmentacja sprawdza, czy projekt jest sfragmentowany, tworząc graf składowych i powiązań między nimi, a następnie przeglądając je rekurencyjnie. Jeżeli jakaś składowa nie została odwiedzona, to oznacza to, że ta składowa nie jest częścią głównego projektu i że projekt jest sfragmentowany. Funkcja zwraca *0* dla osobnika nie sfragmentowanego, a dla sfragmentowanego sumę odległości prymitywów nie należących do głównego projektu od głównego projektu.

Klasa:

Siedzisko

Najważniejsze metody:

Oszacuj ();

Uwagi:

Metoda Oszacuj() klasy Siedzisko sprawdza, czy istnieje płaska powierzchnia do siedzenia oraz czy nie jest ona przykryta przez inne składowe.

Klasa:

Rozmiar

Najważniejsze metody:

Oszacuj ();

Uwagi:

Użytkownik podaje minimalne i maksymalne dopuszczalne wymiary .

Metoda Oszacuj() klasy Rozmiar zwraca różnicę między wartościami rozciągłości podanymi przez użytkownika, a tymi obliczonymi dla danego osobnika.

Klasa:

Podstawa

Najważniejsze metody:

Oszacuj ();

Uwagi:

Metoda Oszacuj() klasy Podstawa sprawdza, czy wszystkie składowe wchodzące w skład podstawy krzesła dotykają podłoża.

Klasa:

Proporcje

Najważniejsze metody:

Oszacuj ();

Uwagi:

Metoda Oszacuj() klasy Proporcje sprawdza, czy w danym artefakcie (osobniku) zostały zachowane wymagane proporcje pomiędzy składowymi poszczególnych klas (siedziska, podstawy i oparcia).

Klasa:

Środek

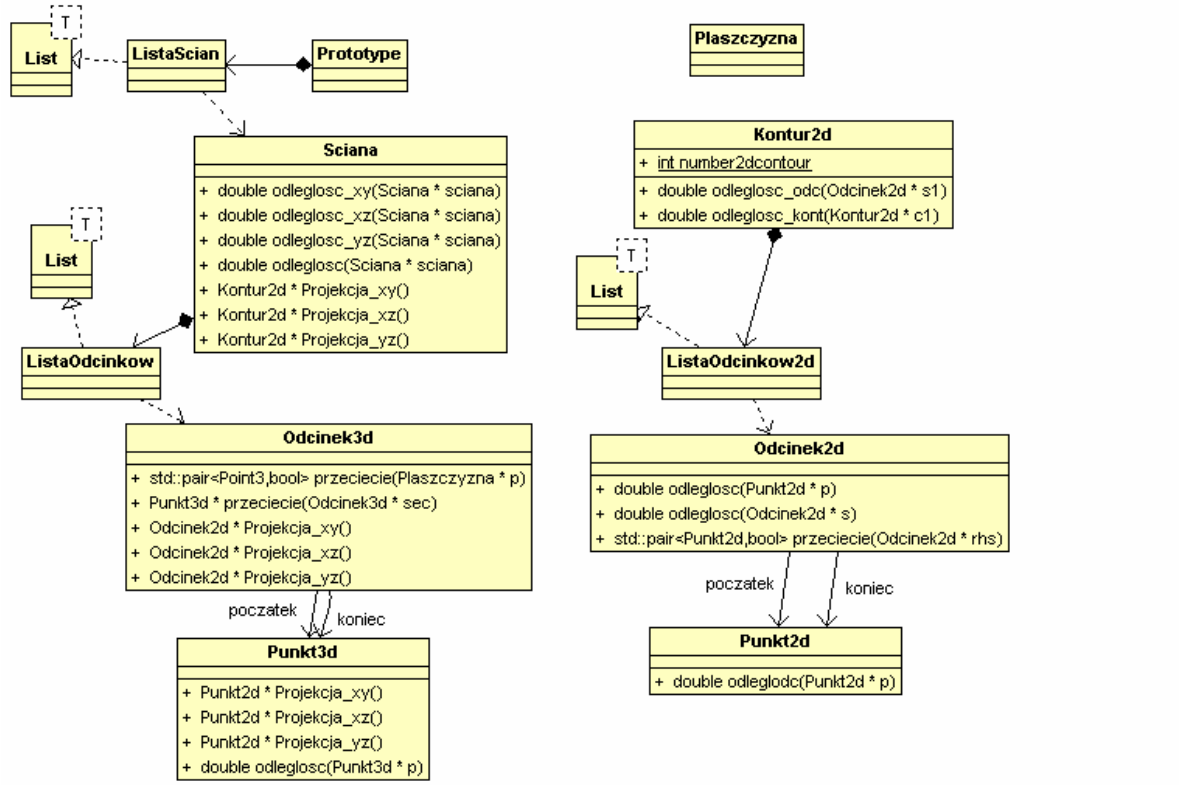
Najważniejsze metody:

Oszacuj ();

Uwagi:

Następnym istotnym modułem jest moduł geometrii, który jest wykorzystywany przez moduł oszacowań. Moduł ten również został zaprojektowany za pomocą paradygmatu „bottom-up". Model geometrii składa się z bazowych elementów, takich jak klasy Punkt3d, Punkt2d oraz bardziej złożonych klas, takich jak Odcinek2d, Odcinek3d, Kontur, Ściana oraz Prototyp.

Na Rys. 8.7 przedstawiony został diagram klas modułu geometrii.



Rys. 8.7 Diagram klas modułu geometrii.

Klasa:

Punkt3d

Najważniejsze metody:

Punkt2d * Projekcja_xy ();

Punkt2d * Projekcja_xz ();

Punkt2d * Projekcja_yz ();

double odległość(Punkt3d * p);

Uwagi:

Najważniejszymi metodami tej klasy są metody rzutujące punkt na płaszczyzny wyznaczone przez osie układu współrzędnych oraz metody obliczające odległości pomiędzy punktami.

Klasa:

Odcinek3d

Najważniejsze metody:

Odcinek2d * Projekcja_xy ();

Odcinek 2d * Projekcja_xz ();

Odcinek 2d * Projekcja_yz ();

Punkt3d * Przecięcie (Odcinek3d *sec);

std::pair<Punkt3d,bool> Przecięcie (Płaszczyzna *p);

Uwagi:

Każdy obiekt klasy Odcinek3d posiada dwa pola: *początek* oraz *koniec*, które są wskaźnikami do obiektów klasy Punkt3d – współrzędnych końców odcinka. Najważniejszymi metodami są metody rzutujące odcinek na płaszczyzny wyznaczone przez osie układu współrzędnych oraz metody obliczająca punkt przecięcia z odcinkiem oraz płaszczyzną.

Klasa:

Ściana

Najważniejsze metody:

Kontur2d * Projekcja_xy ();

Kontur2d * Projekcja_xz ();

Kontur2d * Projekcja_yz ();

double odległość_xy(Ściana * ściana);

double odległość_xz(Ściana * ściana);

double odległość_yz(Ściana * ściana);

Uwagi:

Najważniejszymi metodami tej klasy są metody rzutujące ścianę na płaszczyzny wyznaczone przez osie układu współrzędnych oraz metody obliczająca odległości pomiędzy ścianami.

Klasa:

Prototyp

Najważniejsze metody:

Kontur2d * Projekcja_xy ();

Kontur2d * Projekcja_xz ();

Kontur2d * Projekcja_yz ();

double odległość_xy(Ściana * ściana);

double odległość_xz(Ściana * ściana);

double odległość_yz(Ściana * ściana);

Uwagi:

Każdy obiekt klasy prototyp posiada wektor wskaźników do ścian z których jest zbudowany.

Klasa:

Punkt2d

Najważniejsze metody:

double odleglość(Punkt2d * p);

Uwagi:

Klasa Punkt2d posiada pola odpowiadające współrzędnym punktu w przestrzeni dwuwymiarowej. Najważniejszą metodą jest metoda obliczająca odległość pomiędzy punktami.

Klasa:

Odcinek2d

Najważniejsze metody:

double odleglość(Punkt2d * p);

double odleglość(Odcinek2d * s);

std::pair<Punkt2d,bool> Przecięcie (Odcinek2d *rhs);

Uwagi:

Do najważniejszych metod należą metody obliczające punkty przecięcia z odcinkiem oraz metody obliczające odległość od punktu oraz odcinka

Klasa:

Kontur2d

Najważniejsze metody:

double odległosc_odc(Odcinek2d * s1);

double odległosc_kont(Kontur2d * c2);

Uwagi:

Do najważniejszych metod należą metody obliczające odległość od odcinka oraz od konturu.

8.4 Diagramy aktywności

Diagram aktywności przedstawiony na Rys. 8.8 opisuje stany modułu sterującego w czasie działania algorytmu genetycznego. Wyróżnione zostały następujące stany:

Inicjalizacja systemu

Wygenerowanie genotypów osobników w populacji początkowej w sposób losowy

Obliczanie wartości funkcji przystosowania

Odwzorowanie genotypów na fenotypy w populacji wewnętrznej

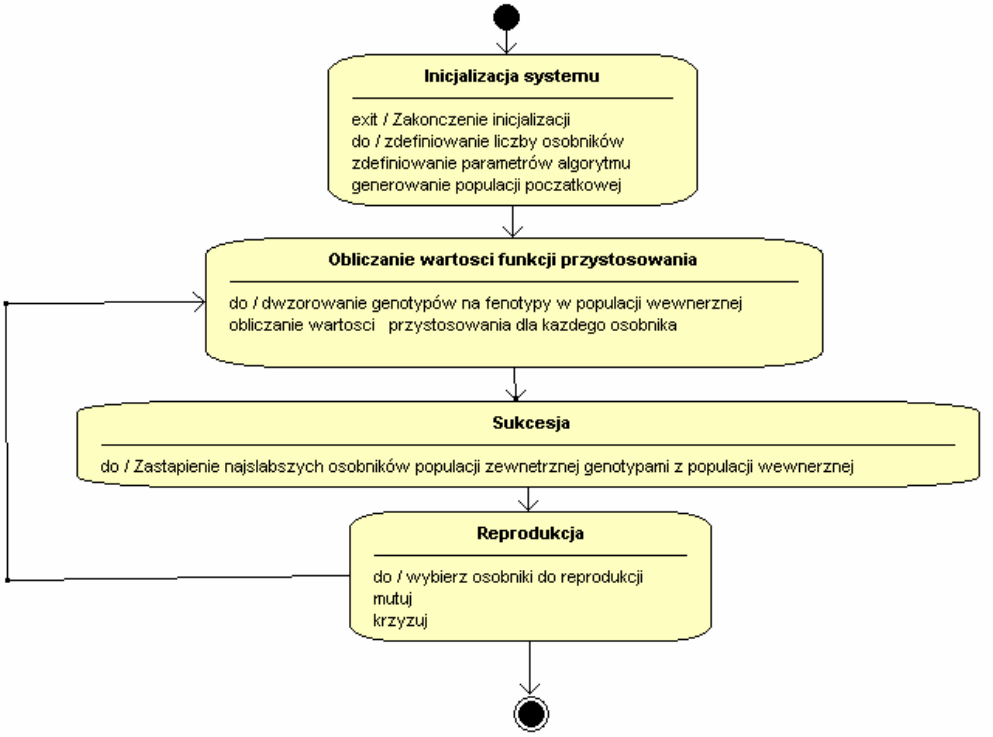
Obliczanie wartości funkcji przystosowania dla każdego osobnika populacji z uwzględnieniem wszystkich używanych kryteriów oszacowań

Sukcesja

Zastąpienie najsłabszych osobników populacji zewnętrznej osobnikami z populacji wewnętrznej

Reprodukcja

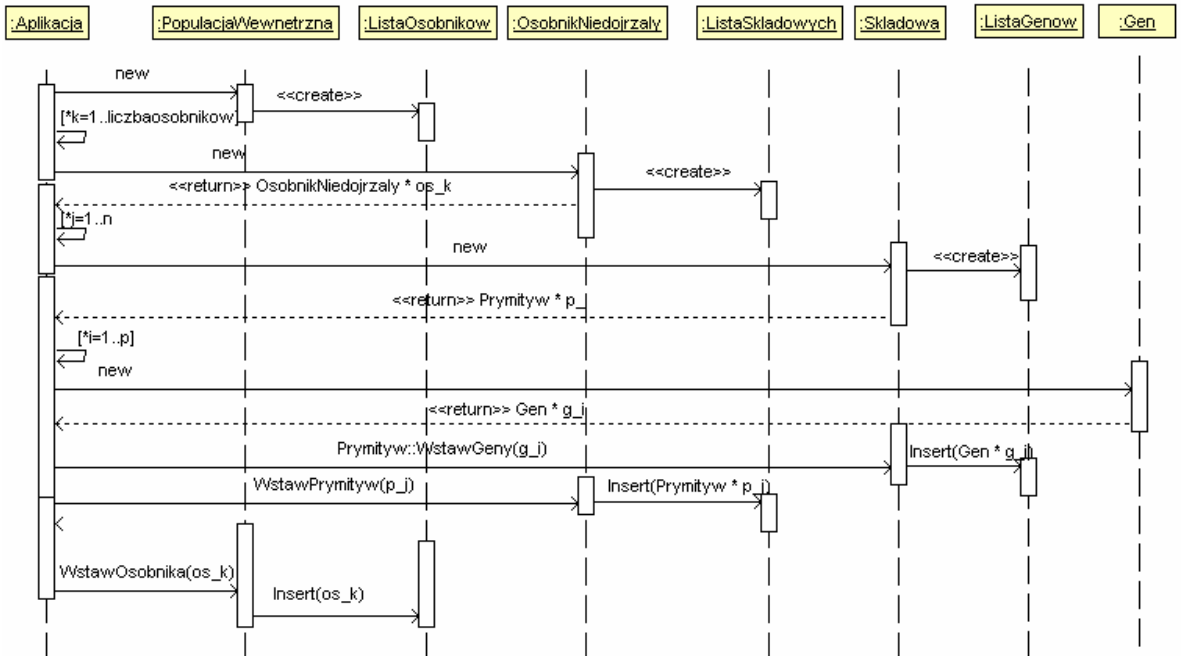
Generowanie potomstwa z wybranych osobników poprzez użycie operatorów krzyżowania i mutacji.



Rys. 8.8 Diagram aktywności.

8.5 Diagramy interakcji

Na Rys. 8.9 przedstawiony został diagram sekwencyjny obrazujący sposób tworzenia populacji wewnętrznej. Obiekt klasy Aplikacja z modułu sterującego odpowiedzialny jest za stworzenie obiektu klasy PopulacjaWewnętrzna, obiektów klasy OsobnikNiedojrzały, obiektów klasy Składowa oraz obiektów klasy Gen oraz umieszczenie obiektów na odpowiednich listach.



Rys. 8.9 Diagram sekwencyjny.

8.6 Formalizacja problemu optymalizacji projektu

Optymalizacja projektu polega na otrzymaniu najlepszego rozwiązania zadanego celu przy spełnieniu pewnych ograniczeń. Często istnieje wiele kryteriów, które muszą być wzięte pod uwagę. Wtedy problem jest formułowany jako wielokryterialny, w którym celem jest minimalizacja (albo maksymalizacja) kilku funkcji celu równocześnie.

Formalizacja problemu optymalizacji polega na stworzeniu modelu matematycznego, który opisuje zachowanie systemu fizycznego.

Wielkości, dla których wartości mają zostać znalezione, nazywać będziemy **zmiennymi**.

Aby otrzymane rozwiązanie było dopuszczalne, muszą zostać spełnione pewne **ograniczenia.**

W procesie wyboru dobrego rozwiązania spośród zbioru wszystkich rozwiązań, muszą być zdefiniowane **kryteria**, które umożliwią porównywanie rozwiązań. Najczęściej używaną funkcją w optymalizacji projektów jest objętość.

Gdy mamy już zdefiniowane zmienne $x_1, x_2, ..., x_N$, ograniczenia $g_1(x) \geq 0, ..., g_K(x) \geq 0$ oraz kryterium *f*, definiujemy problem optymalizacji w sposób następujący:

Zaleźć wektor $x^T = [x_1^*, x_2^*, ..., x_N^*]^T$, który spełnia ograniczenia

$g_k(x) \geq 0$ dla $k = 1,2,...,K$ i optymalizuje funkcję $f(x) = [f_1(x), f_2(x), ..., f_l(x)]^T$

Dla problemów wielokryterialnych, każda funkcja przystosowania osiąga minimum w innym punkcie. Dlatego wprowadza się pojęcie Pareto optymalności**.**

Definicja 8.1 Niech $X \subset R^N$ i niech *f* będzie funkcją zdefiniowaną nad X.

Punkt $x^* \in X$ jest **pareto optimum** wtedy i tylko wtedy, gdy nie istnieje żaden punkt $x \in X$ taki, że

$f_i(x) \leq f_i(x^*)$ dla $i = 1,2,...,l$ oraz

$f_i(x) < f_i(x^*)$ dla przynajmniej jednego *i*

Większość wielokryterialnych metod optymalizacji sprowadza problem do problemu optymalizacji skalarnej za pomocą funkcji preferencji, która jest następnie minimalizowana. W tej sytuacji rozwiązanie problemu polega na znalezieniu $x^* \in X$, takiego, że:

$P\lfloor f(x^*) \rfloor = \min_{x \in X} P[f(x)]$ gdzie $P[f(x)]$ jest funkcją preferencji.

W prezentowanej pracy użyta została metoda SWGR (ang. Sum of Weighted Global Ratios) do zdefiniowania funkcji preferencji. Metoda ta umożliwia przy wielu kryteriach o różnych efektywnych przeciwdziedzinach jednakowe traktowanie wszystkich kryteriów.

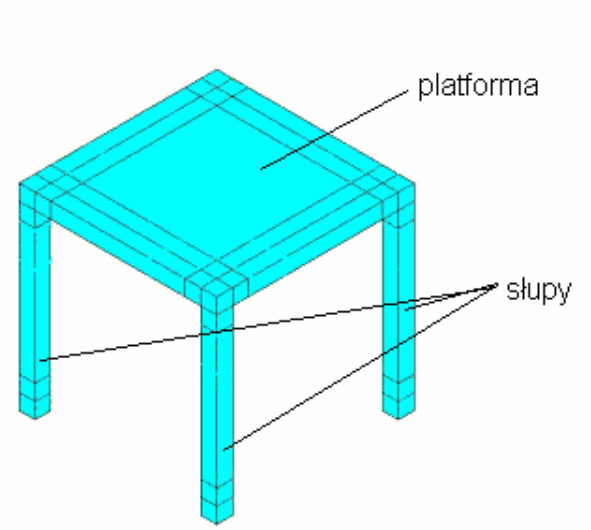
8.7 Otrzymane wyniki – problem optymalizacji kształtu platformy

Algorytm genetyczny z hierarchicznym chromosomem znajduje zastosowanie zwłaszcza w problemach projektowych o nieznanej optymalnej liczbie składowych artefaktu.
Pierwszym zastosowaniem aplikacji bazującej na algorytmie genetycznym z hierarchicznym chromosomem był problem optymalizacji kształtu platformy. Opis problemu wraz z rozwiązaniami problemu znajduje się w niniejszym rozdziale.

Problem optymalizacji dotyczy minimalizacji sumarycznej objętości składowych artefaktu poprzez znalezienie wartości następujących zmiennych:

1. liczby składowych,
2. współrzędnych x, y środków składowych (rozmieszczenia składowych),
3. długości, szerokości oraz wysokości poszczególnych składowych.

Przykładowy artefakt znajduje się na Rys. 8.10



Rys. 8.10 Przykładowy artefakt.

Przekrój poprzeczny platformy ma wymiary 100×100 cm. Wysokość platformy oraz wymiary przekrojów poprzecznych słupów i ich rozmieszczenie mogą być dowolne.

Uwzględnione zostały następujące ograniczenia:

- fragmentacja,
- naprężenia.

Opis algorytmu sprawdzającego, czy osobnik jest sfragmentowany znajduje się w Rozd. 8.4.

Rozkład naprężeń w modelowanym obszarze opisany jest równaniem liniowej sprężystości

$$\sum_{j=1}^{3} \frac{\partial \sigma_{ij}}{\partial x_j} = 0 \tag{8.1}$$

gdzie σ_{ij} oznacza tensor naprężeń zdefiniowany z pomocą następującej zależności:

$$\sigma_{ij} = 2\mu\varepsilon_{ij} + \lambda\delta_{ij}\varepsilon_{kk} \tag{8.2}$$

We wzorze 8.2 ε_{ij} oznacza tensor odkształceń zdefiniowany z pomocą pochodnych cząstkowych z pola wektorowego przemieszczeń wektora przemieszczeń u_i.

$$\varepsilon_{ij} = 0.5\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{8.3}$$

Współczynniki μ oraz λ oznaczają stałe Lamego i wynoszą odpowiednio:

$$\mu = \frac{E}{2(1+\nu)} \tag{8.4}$$

$$\lambda = \frac{\nu E}{1+\nu}$$

Współczynniki μ oraz λ są zależne od modułu Younga E i współczynnika Poissona ν. Moduł Younga i współczynnik Poissona definiują właściwości materiału, z którego wykonana jest platforma. W powyższym przykładzie założono, że platforma wykonana jest ze stali węglowej

konstrukcyjnej wyższej jakości, ogólnego przeznaczenia, dla której $E = 206*10^9$ [Pa] oraz $\nu = 0.3$.

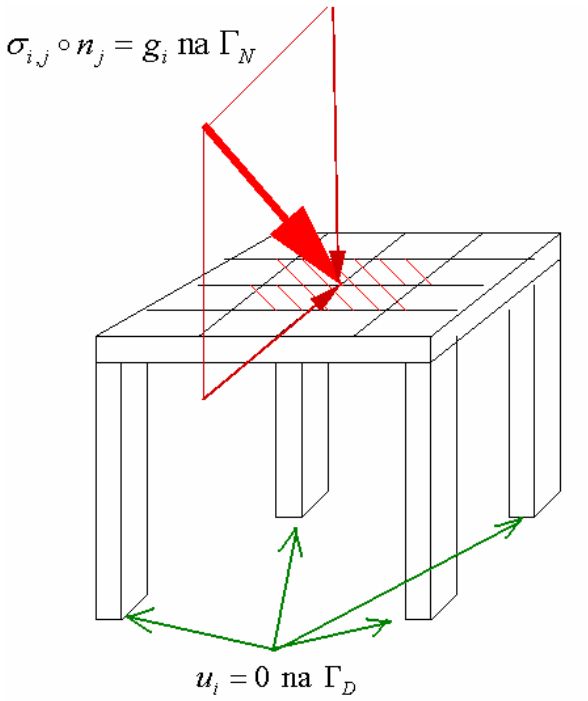
W modelowanym przykładzie założono warunki brzegowe przedstawione na Rys. 8.11. Przyjęto, że słupy platformy są umocowane do powierzchni podłogi, przyjmując warunek brzegowy Dirichleta

$$u_i = 0 \; na \; \Gamma_D \tag{8.5}$$

gdzie brzeg Γ_D reprezentuje spód słupów platformy. Założono, że środkowa część platformy, zaznaczona na rysunku prostokątem rysowanym czerwoną przerywaną linią, obciążona została siłą $g = 30$ kN, działającą pod kątem 45 stopni. W tym celu przyjęto warunek brzegowy Neumanna

$$\sigma_{i,j} \circ n_j = g_i \text{ na } \Gamma_N \tag{8.6}$$

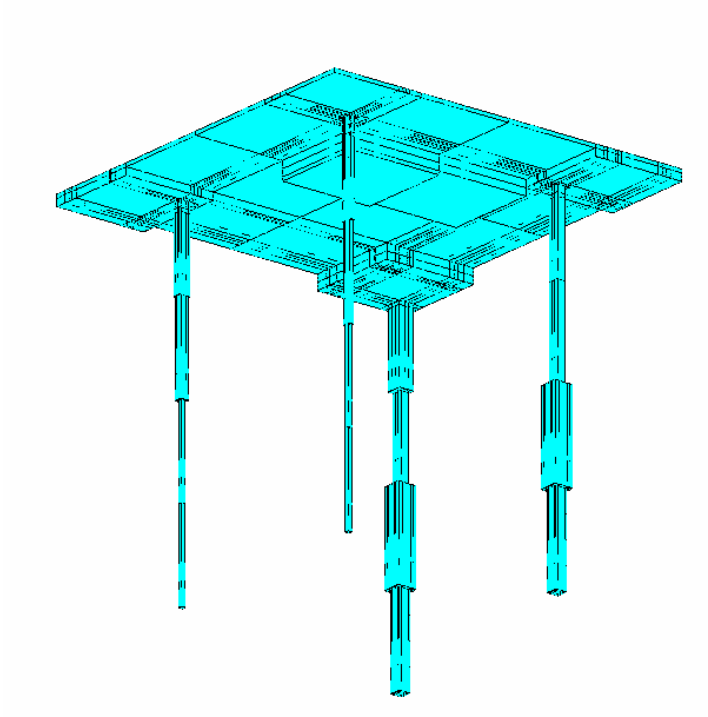
gdzie brzeg Γ_N reprezentuje środkową część platformy.



Rys. 8.11 Warunki brzegowe.

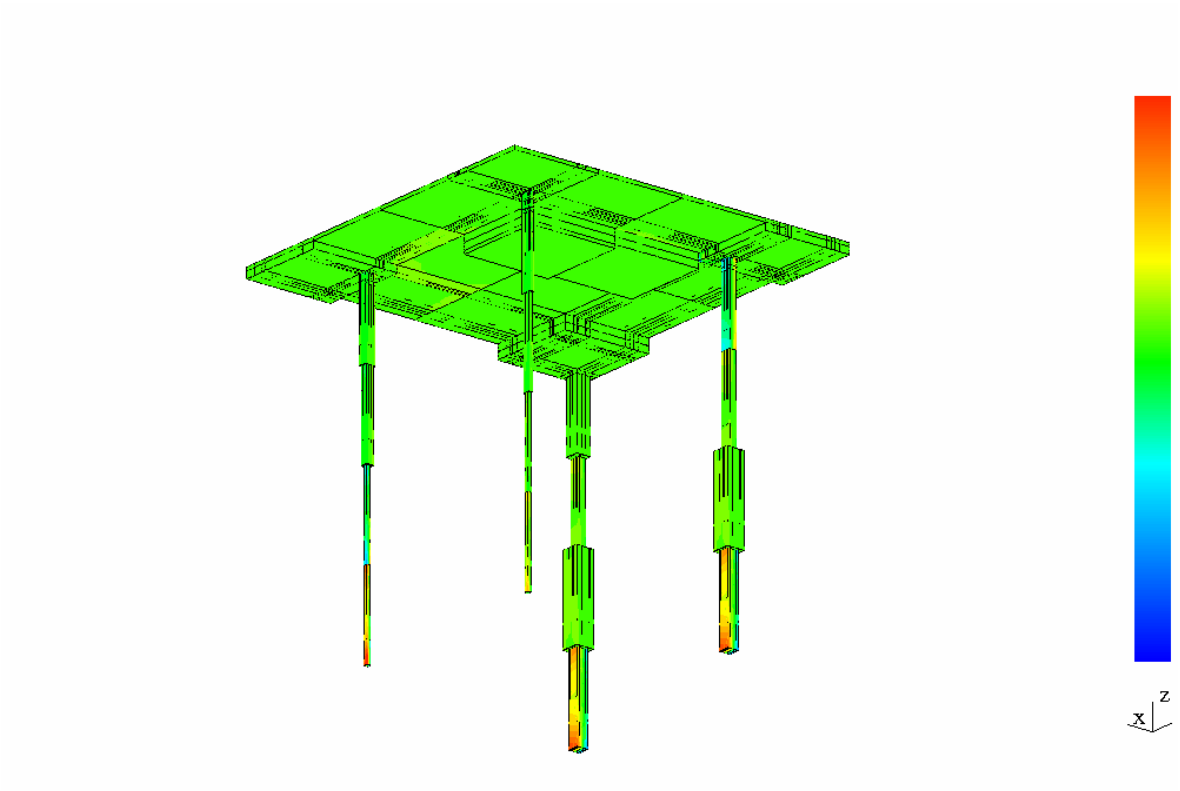
Do rozwiązania numerycznego równania 8.1 z warunkami brzegowymi 8.5, 8.6 (obliczenia naprężeń w modelu platformy) użyta została Metoda Elementów Skończonych (MES) [19]. W tym celu aplikacja bazująca na algorytmie genetycznym z hierarchicznym chromosomem sprzężona została z aplikacją *parhp3D* [69] Metody Elementów Skończonych z *hp* adaptacją.
Użyty algorytm genetyczny jest zgodny ze schematem (μ+λ), gdzie μ=λ. Potomkowie powstają w wyniku krzyżowania hierarchicznego, mutacji alleli oraz mutacji grup alleli.
Populacja początkowa składała się z 50 osobników (μ=50).

Na Rys. 8.12 przedstawiony został najlepszy osobnik otrzymany po 50 epokach.



Rys. 8.12 Najlepszy osobnik po 50 epokach.

Składowe platformy, na które działała największa siła: środkowe oraz z przymocowanymi słupami mają większą wysokość od pozostałych części platformy. Słupy, na które działała największa siła posiadają składowe o większym polu przekroju poprzecznego (aby zapobiec wyboczeniu).
Na Rys. 8.13 przedstawiony został ten sam osobnik wraz z zaznaczonymi wartościami naprężeń.



Rys. 8.13 Najlepszy osobnik po 50 epokach z zaznaczonymi wartościami naprężeń.

Największe moduły naprężeń występują w najniższych składowych słupów oraz w miejscach połączeń pomiędzy składowymi o różnych przekrojach poprzecznych.

9 Wnioski i dalsza praca

W pracy zaprezentowane zostały wyniki badań, których głównym celem było rozszerzenie istniejącego modelu markowowskiego dla prostego algorytmu genetycznego o dodatkowe operatory genetyczne: przesunięcie i permutację. Na podstawie rozszerzonego modelu markowowskiego wykazane zostały istotne własności algorytmu, takie jak ergodyczność łańcucha markowa opisującego dynamikę algorytmu genetycznego oraz zbieżność w sensie zbieżności miar. Udowodniono również, że prawdopodobieństwo osiągnięcia żądanej populacji z danej populacji początkowej po skończonej liczbie kroków jest równe jeden.

Powyższy model markowowski jest podstawą do badania własności algorytmów genetycznych bardziej złożonych od prostego algorytmu genetycznego.

Za jego pomocą zamodelowano i otrzymano analogiczne wyniki teoretyczne dla używanego w projektowaniu wspomaganym komputerowo algorytmu genetycznego z hierarchicznym chromosomem. Dodatkowo w celu obliczenia efektywnego przystosowania oraz prawdopodobieństwa, że stworzone osobniki pasują do schematu zastosowano również drugie podejście - Mikroskopową Teorię Schematów Poli'ego. Prezentowany algorytm genetyczny bazujący na hierarchicznym chromosomie został zaimplementowany w języku C++. Ponieważ aplikacja została zaprojektowana jako otwarta, może być stosowana do rozwiązywania wielu różnorodnych problemów optymalizacji. W pracy przedstawione zostały rozwiązania dwóch przykładowych problemów: optymalizacji kształtu platformy oraz projektowania krzeseł.

Przedstawione w mojej pracy wyniki stanowią jedynie wstęp do dalszej pracy nad teorią algorytmów ewolucyjnych stosowanych w projektowaniu graficznym wykorzystującym struktury hierarchiczne. Prezentowany model markowowski dostarcza sposobu kodowania struktur hierarchicznych za pomocą ciągów binarnych oraz modelowania hierarchicznych operatorów genetycznych. Zastosowanie powyższego modelu do znanych algorytmów ewolucyjnych opartych na strukturach hierarchicznych umożliwi otrzymanie wyników teoretycznych dla tych algorytmów. W przyszłości prezentowany model zostanie zastosowany do modelowania i badania własności algorytmów genetycznych bazujących na grafach oraz na grafach hierarchicznych.

Bibliografia

[1] Angeline P. J., Kinnear A. E., *Advances in Genetic Programming 2,* MIT Press, Cambridge, MA, USA, 1996

[2] Arabas J., *Wykład z algorytmów ewolucyjnych*, WNT, 2001

[3] Axelord, R., The evolution of strategies in the iterated prisoner's dillema, In L. Davis, editor, *Genetic Algorithms and Simulated Annealing,* p. 32-41, Pitman, London 1987

[4] Back, T., Fogel, D., B., Michalewicz, Z., *Evolutionary Computation 1,* Institute of Physics, 2000

[5] Back, T., Fogel, D., B., Michalewicz, Z., *Evolutionary Computation 2,* Institute of Physics, 2000

[6] Back, T., Hoffmeister F., , Schwefel, H., P., A survey of evolution strategies, *4th International Conference of Genetic Algorithms,* p. 2-9, 1991

[7] Bentley, P. J. , Wakefield, J. P. Hierarchical Crossover in Genetic Algorithms. In *Proceedings of the 1st On-line Workshop on Soft Computing (WSC1)*, 37-42. Nagoya University, Japan. 1996

[8] Bentley, P. J. , Wakefield, J. P. Finding Acceptable Pareto-Optimal Solutions using Multiobjective Genetic Algorithms. Submitted to *Soft Computing*, Springer Verlag Ltd. Research Report RN/98/66. 1998

[9] Bentley, P. J. , Wakefield, J. P., Conceptual evolutionary design by genetic algorithms. In *Engineering Design and Automation Journal,* vol.2 , John Willey and Sons, 1996

[10] Bentley, P.J. *Genetic Evolutionary Design of Solid Objects using a Genetic Algorithm*. Ph.D. Thesis, UCL London, 1997

[11] Bentley, P. , O'Reilly , U. Ten Steps to Make a Perfect Creative Evolutionary Design System, In *Proceedings of the Genetic and Evolutionary Computation Conference.* San Francisco, Calif.: Morgan Kaufmann. 2001

[12] Bickel, A.S., Bickel, R.W. Tree structured rules in genetic algorithms. *Genetic Algorithms and Simulated Annealing*. Pittman, 1987

[13] Booch, G., *Object-oriented analysis and design with applications,* The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1994

[14] Booch, G., Jaconson, I., Rumbaugh, J., *The Unified Modeling LanguageUser Guide,* Adison-Wesley Pub. Co., 1998

[15] Bouchard, E., E., Kidwell, G., H., Rogan. J., E., The application of artificial Intelligence technology to aeronautic system design, in *AIAA, AHS and ASEE, Aircraft Design, Systems and Operations Meeting,* p. 7-9 Atlanta, Georgia, 1998

[16] Burczyński, T., Beluch, W., Długosz, A., Kuś, W., Ornatek, P., Evolutionary design in computer aided engineering, In *Proceedings of 5th KAEGiOG Conference,* Polanica Zdrój, 2000

[17] Burczyński, T., Kuś, W., Nowakowski, M., Ornatek, P., Evolutionary algorithms In nondestructive identification of internal defects, In *Proceedings of 5th KAEGiOG Conferenc,e* Jastrzębia Góra, *2001, p. 48-55*

[18] Chakrabarti, A., Function: A software for synthesis of mechanical design, In *Engeenering Computer Newsletter 59,* p. 18-25, 1995

[19] Ciarlet, P., *The Finite Element Method for Elliptic Problems,* Society for Industrial & Applied Mathematics, 2002

[20] Davidor, Y., *Genetic algorithms and robotics,* 1991

[21] Davis, E., D., Principe, J., C., A markov chain framework for the simple genetic algorithm, *Evolutionary Computation,* 1(3):269-288, 1993

[22] Davis, L. , *The Handbook of Genetic Algorithms,* Van Nostrand Reinhold, New York 1991

[23] Davis, L. (ed), *Genetic Algorithms and Simulated Annealing,* Pitman, London 1987

[24] Dixon, J., R., Simmons, M., K., An architecture for application of artificial intelligence to design. In *ACM IEEE 21st Conference on Design Automation,* p. 634-640, Albuquerque, New Mexico, 1984

[25] Feller, W. *Wstęp do rachunku prawdopodobieństwa,* Państwowe Wydawnictwo Naukowe, Warszawa 1981

[26] Fisher, K., A., The application of genetic algorithms to optimising the design of an engine block for low nosie, In *Genetic algorithms in Engineering Systems: Innovation and Applications,* p. 18-22, Sheffield, 1995

[27] Flake G., W., *Computational Beauty of Nature,* MIT Press, 1998

[28] Fogel D., B., *Evolutionary Computation. Towards a New Philosophy of Machine Intelligence,* IEEE Press, 1995

[29] Fogel D., B., Fogel, L., J., Continuous evolutionary programming. Analysis and experiments, *Journal of Cybernetics and Systems,*(26):79-90, 1995

[30] Fogel L., Owens A., J., Walsh, M., J., *Artificial Intelligence Through Simulated Evolution.* Wiley, New York, 1996

[31] Fogel L., J., *Biotechnology: Concepts and Applications.* Eanglewood Cliffs NJ, 1963

[32] Foley, J.D, van Dam, A., Feiner S.K., Hughes J.F, Philips R.L., *Wprowadzenie do grafiki komputerowej*, Wydawnictwa Naukowo-Techniczne, Warszawa 1995

[33] Furata, H., Maeda, K., Watanabe, W., Application of genetic algorithm to aesthetic design of bridge structures, In *Microsomputers in Civil Engeenering,* vol. 10, p. 415-421, 1995

[34] Garipov, V., Diakov, E., Semenkin, E., Vakhtel, S., Adaptive search methods in spacecraft system optimal design. In *Adaptive Computing in Engeenering Design and Control,* p. 194-201, 1994

[35] Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, MA, Addison-Wesley, 1989

[36] Grabska E. Theoretical concepts of graphical modelling part one: realisation of cp-graphs, *Machine Graphics and Vision,* 2(1):3-38, 1993

[37] Grabska E. Theoretical concepts of graphical modelling part two: cp-graphs grammars and languages, 2(2):149-178,1993

[38] Grabska, E., Palacz, W., Szyngiera P., Hierarchical graphs in creative design. *Machine Graphics & Vision*, 9(1/2): 115-122, 2000

[39] Gritz, L., Hahn, J., K., Genetic programming evolution of controllers for 3-D character animation, In John R. Koza, Kalyanmoy Deb, Marco Dorgio, David B. Fogel, Max H. Garzon, Hithoshi Iba and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedinges of the Second Annual Conference,* p. 139-146, 1997

[40] Holland, J. Genetic Algorithms, *In Scientific American*, pages 66-72,1992

[41] Holland, J. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, Massachusetts, second edition, 1992

[42] Holland, J. *, Hidden Order*. MIT Press, 1999

[43] Horner, A., Goldberg, D., E., Genetic algorithms and computer-assisted music composition, In *4th International Conference on Genetic Algorithms,* p. 437-441, 1991

[44] Horst, R, Pardalos, P., M., *Handbook of Global Optimization,* Kluver*,* 1995

[45] Husband, P, Jeremy, G., McIlhaga, M., Ives. R., Two applications of genetic algorithms to component design, In *Selected Papers from AISB Workshop on Evolutionary Computing,* p. 55-61, 1996

[46] Iosifescu, M. *Skończone procesy Markowa i ich zastosowania,* Państwowe Wydawnictwo Naukowe, Warszawa 1988

[47] Kanal, L., Cumar, V., *Serach in Artificial Intelligence,* Springer-Verlag, 1998

[48] Keane, A., Brown, Bron., M., The design of a satellite boom with enhanced vibration performance using genetic algorithm techniques. In *Adaptative Computing in Engineering Design and Control,* p. 107-113, 1996

[49] Koehler, G.J., Vose, M.D. and Siddhartha Bhattacharyya. General Cardinality Genetic Algorithms. *Evolutionary Computation*, 5(4): 439-459, 1997

[50] Kowalenko, I. N., Kuzniecow, N. J., Szurienkow, W. M. *Procesy stochastyczne*, Państwowe Wydawnictwo Naukowe, Warszawa 1989

[51] Koza, J. R. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA, USA, 1992

[52] Krishnakumar, K, Goldberg, D., E., Control systems optimization using genetic algorithms, In *Journal of Guidance, Control and Dynamics,* vol. 15, p. 735 – 740, 1992

[53] Langdon W. B., Size Fair and Homologous crossovers for Tree Genetic Programming, 1999

[54] Langdon, W., B., Poli R., *Foundations of genetic programming,* Springer Verlag, Berlin 2002

[55] Luke S., Spector L., A revised Comparison of Crossover and Mutation in Genetic Programming

[56] Madsen, S., T., Exploring Similarities in Music Performances with an Evolutionary Algorithm, In *Proceedings of The Florida Artificial Intelligence Research Society (FLAIRS)* Conference, Clearwater Beach, Florida, USA, 2005

[57] McKay B., Willis M. J., Barton G. W. Using a tree structured genetic algorithm to perform symbolic regression. In A. M. S. Zalzala, editor, *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Aplications,* GALESIA, t. 414, s. 487-492. Sheffield, UK, 12-14 September 1995

[58] Michalewicz, Z., *Genetic Algorithms+Data structures=Evolution Programs,* Springer Verlag, Berlin Heidelberg 1992

[59] Montana, D., J., Strongly typed genetic programming. *Evolutionary Computation*. 3(2):199-230, 1995

[60] Nikodem, P., *Projektowanie za pomocą algorytmów genetycznych z wykorzystaniem teorii grafów,* praca magisterska, Kraków, lipiec 2000

[61] Nix, A.E, Vose, M.D. Modeling genetic algorithms with Markov chains. *Annals of Math. And Artificial Intelligence* 5(1) 79-88, 1992

[62] Obavashi, S., Takanashi, S., Genetic algorithm for aerodynamic inverse optimisation problem, In *Genetic Algorithms in Engineering Systems: Innovations and Applications,* p.7-12, Scheffield, 1995

[63] O'Reilly U., *An Analysis of Genetic Programming,* PhD Thesis, Carleton University, Ottawa-Carleton Institute for Computer Science, Ottawa, Ontario, Canada, 22 September 1995

[64] O'Reilly U., Oppacher F., The troubling aspects of a building block hypothesis for Genetic Programming. In L. Darrel Whitley and Michael D. Vose, editors, *Foundations of genetic Algorithms 3,* strony 73-88, Estes Park, Colorado, USA, 31 July-2 August 1994, Morgan Kaufmann, 1995

[65] Osyczka, A., *Evolutionary Algorithms for Single and Multicriteria Design Optimization,* Physica-Verlag, Heidelberg, 2002

[66] Page J., Poli, R., Longdon, W.B., Smooth Uniform Crossover with smooth Point Mutation in Genetic Programming: A Preliminary Study, In R. Poli, P. Nordin,W. B. Langdon, T. C. Fogarty editors, *Genetic Programming, Proceedings of EuroGP'99*, vol. 1598, p. 39-49, Goteborg, Sweden, 1999, Springer-Verlag

[67] Paszyńska, A., Markov Chain Model and Poli Based Schema Model for Bentley's Genetic Algorithm , In *Proceedings of The Florida Artificial Intelligence Research Society (FLAIRS)* Conference, Clearwater Beach, Florida, USA, 2005

[68] Paszyńska, A., An Extension of Vose's Markov Chain Model for Genetic Algorithms, Genetic and Evolutionary Computation Conference (GECCO), Washington, D.C. USA, 2005

[69] Paszyński, M., Demkowicz, L., Parallel Fully Automatic hp Adaptive 3D Finite Element Package, *Engineering with Computers,* in press, 2006

[70] Pieczara, J., *Rozprawy Monografie 134, Algorytmy genetyczne w mechanice konstrukcji,* Uczelniane Wydawnictwa Naukowe-Dydaktyczne, Kraków, 2004

[71] Pham, D., T., Yang, Y., A genetic algorithm based preliminary design system, In *Journal of Automobile Engineers,* vol. 207, p. 127-133, 1993

[72] Plucińska, A., Pluciński, E. *Probabilistyka: Rachunek prawdopodobieństwa, Statystyka matematyczna, Procesy stochastyczne,* Wydawnictwa Naukowo Techniczne, Warszawa 2000

[73] Poli, R. General Schema Theory for Genetic Programming with subtree-Swapping Crossover, Genetic Programming. In *Proceedings of EuroGP*, LNCS, (Milan): Springer-Verlag, 2001

[74] Poli, R., Longdon, W.B. Schema Theory for Genetic Programming with One-point Crossover and Point Mutation. *Evolutionary Computation*. 6(3):231-252 , 1998

[75] Poli, R., McPhee, N.F. Exact schema Theory for GP and Variable-length GAs with Homologous Crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference.* San Francisco, Calif.: Morgan Kaufmann, 2001

[76] Poli, R., Rove, J.E., McPhee, N.F. Markov Chain Models for GP and variable-length Gas with Homologous Crossover , In *Proceedings of the Genetic and Evolutionary Computation Conference.* San Francisco, Calif.: Morgan Kaufmann, 2001

[77] Poli, R., Longdon, W.B., On the Search Properties of Different Crossover Operators in Genetic Programming, *Proceedings of the Third International Conference on Genetic Programming, GP'98*, pp. 293-301, Madison, Wisconsin, Morgan Kaufmann, 1989

[78] Poli, R., Longdon, W.B., A Review of Theoretical and Experimental Results on Schemata in Genetic Programming, In W. Banzhaf, R. Poli, M. Schoenauer and T. Fogarty (Eds.), Proceedings of the First European Workshop on Genetic Programming - EuroGP'98, Paris, April 14-15, 1998, LNCS 1391, pp. 1-15, Springer-Verlag, Berlin, 1998

[79] Ramirez, R., Hazan, A., Modeling Expressive Music Performance in Jazz, In *Proceedings of The Florida Artificial Intelligence Research Society (FLAIRS)* Conference, Clearwater Beach, Florida, USA, 2005

[80] Rechenberg, I., *Evolutionsstrategien, Simulationsmethoden in der Medizin und Biologie,* Schneider B., Ranft, U., eds., Springer, 1978, p. 83-104

[81] Rechenberg, I., Evolutionsretegie: Optiemierung Technischer Systeme nach Prinzipien der biologischen Evolution, Fromman-Holzboog Verlag, 1973

[82] Reeves C., R., Rowe J., E., *Genetic Algorithms - Principles and Perspectives*, Kluwer, 2003

[83] Rudolph G., Convergence analysis of canonical genetic algorithm. *IEEE transaction on Neural Networks,* 5(1):96-101, 1994

[84] Rudolph G., Genetic algorithms. In Thomas Back, David B. Fogel, Zbigniew Michalewicz, editors, *Handbook of evolutionary Computation,* s. B2.4-20-27. Oxford University Press, 1997

[85] Rudolph G., Models of stochastic convergence. In Thomas Back, David B. Fogel, Zbigniew Michalewicz, editors, *Handbook of evolutionary Computation,* s. B2.3-1-3. Oxford University Press, 1997

[86] Rudolph G., Stochastic processes. In Thomas Back, David B. Fogel, Zbigniew Michalewicz, editors, *Handbook of evolutionary Computation,* s. B2.2-1-8. Oxford University Press, 1997

[87] Rudolph G., Evolution strategies. In Thomas Back, David B. Fogel, Zbigniew Michalewicz, editors, *Handbook of evolutionary Computation,* Oxford University Press, 2000

[88] Savic, D., A., Walters, G., A., Evolution programs in optimal design of hydraulic networks. In *Adaptive Computer Engineering Design and Control*, p. 146-150, 1994

[89] Schaefer, R., Krok, J., Leżajski, P., Orkisz, J., Przybylski, P., Basic concepts of an open distributed system for cooperative design and structure analysis, *CAMES 3,* p. 169-186, 1996

[90] Schaefer, R., Podstawy genetycznej optymalizacji globalnej, Wydawnictwo Uniwersytetu Jagiellońskiego, Kraków 2002

[91] Schaefer, R., Kołodziej, J., Paszyńska, A. Hierarchical genetic computations in optimal design. *A Journal of Theoretical and Applied Mechanics*. 3(42):519-538, 2004

[92] Schaefer, R., Detailed evaluation of the schemata cardinality modification at the single evolution step, *Proceedings of the 7th Conference on Evolutionary Algorithms and Global Optimization*, pp. 143-147, Kazimierz, 2004

[93] Schaffer, J., D., *Some experiments in machine learning using vector evaluated genetic algorithm*, PhD thesis, Vanderbildt University, Nashville, USA, 1984

[94] Schwefel, H., P., Numerische Optimierung von Computer Modellen Mittels der Evolutionsstrategie, *Interdisciplinary System Research 26*, Birkhaeuser, Basel, 1997

[95] Schwefel, H., P., *Numerical Optimization for Computer Models,* John Wiley, UK, 1981

[96] Schwefel, H., P., *Evolution and Optimum Seeking,* John Wiley, UK, 1995

[97] Smith, S., F., Adaptive learning systems, In *Expert Systems: Principles and Case Studies,* p. 169-189, 1984

[98] Strug, B., Phd Thesis, *Zastosowanie algorytmów ewolucyjnych i transformacji grafowych w projektowaniu graficznym wspomaganym komputerowo*, Phd Thesis, IPPT, Warszawa 2001

[99] Tackett, W., A., Genetic programming for feature discovery and image discrimination, In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, *ICGA-93,* p. 303-309, 1993

[100] Tennant, A., Chambers, B., Adaptive optimisation techniques for the design of microwave absorbers, In *Adaptive Computer Engineering Design and Control*, p. 44-49, 1994

[101] Vose, M.D. *The simple genetic algorithm*, MIT Press, 1999

[102] Vose, M.D., Liepins, G.E. Punctuated Equilibria in Genetic Search. *Complex System*, 5: 31-44

[103] Vose, M. D., Wright, A.H., The simple Genetic Algorithm and the Walsh Transform: part I, Theory, *Evolutionary Computation*, 6(3):253--273, 1998

[104] Watabe, H., Okino, N., A study on genetic shape design, In *Proceedings of the 5th International Conference on Genetic Algorithms, p. 445-450, 1993*

[105] Wentzell, A.,D. *Wykłady z teorii procesów stochastycznych,* Państwowe Wydawnictwo Naukowe, Warszawa 1980

[106] Whigham P.A., A schema theorem for context-free grammars. In 1995 *IEEE Conference on Evolutionary Computation*, tom 1, s. 178-181, Perth, Australia, 29 November – 1 December 1995, IEEE Press

[107] Whigham P.A., *Grammatical Bias for Evolutionary Learning,* PhD thesis, School of Computer Science, University College, University of New South Wales, Australian Defence Force Academy, 14 October 1996

[108] Whigham P.A., Seatch bias, language bias, and genetic programming. In John R. Koza, Dawid E. Goldberg, Dawid B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996, Proceednings of the First Annual Conference,* s. 230-237, Stanford University, CA, USA, 1996, MIT Press

[109] Whitle'y D., Rana S., Hackendorn R.B. Island Model Genetic Algorithms and Linerly Seperable Poblems, AISB, *Workshop on Evolutionary Computing*, Manchester, 1997

[110] Wierzba B., Semczuk A., Kołodziej J., Schaefer R., Hierarchical Genetic strategy with real number encoding*, KAEiOG'03,* 2003

[111] Wilson S. Hierarchical credit allocation in classifier system. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* 1987

[112] Wolpert D., H.,Macready W., G,. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation,* 1(1):67-82, 1997

[113] Wright, A., H., Stephens C.R., Rowe, J. E., Poli R., Bistability in a Gene Pool GA with Mutation*, FOGA 7 (Foundations of Genetic Algorithms 7)*, Morgan Kaufmann, 2003,

p. 63-80