Podstawy Sztucznej Inteligencji
Projekt: Układanie elementów

Prowadzący: mgr inż. Jarosław Hurkała
Autorzy: Marcin Kubik, Mikołaj Markiewicz, Krzysztof Opasiak

Opis problemu:

Celem projektu była implementacja algorytmu ewolucyjnego układającego planszę dla układanki Tangram. Polega ona na dopasowaniu dostępnych elementów (zazwyczaj 7) tak, aby tworzyły pewien obrazek lub figurę. Program powinien wyświetlać wzór do ułożenia, informację o najlepszej adaptacji oraz statystyki dotyczące czasu działania czy liczby iteracji.

Decyzje projektowe:

Aby przyśpieszyć czas obliczeń i zmniejszyć liczbę możliwych kombinacji założono, że elementy układanki nie mogą być obracane. Dodatkowym założeniem (wynikającym z konstrukcji samej układanki) jest nienakładanie się elementów na siebie. Figura musi także być zwarta, tj. nie mogą występować elementy niepołączone z żadnym innym.

Rozwiązanie:

Rozwiązanie, zaimplementowany algorytm ewolucyjny, opiera się na przyporządkowywaniu wierzchołków finalnego obrazka do wierzchołków poszczególnych figur. Pojedynczy osobnik populacji symbolizuje zbiór segmentów wraz z wierzchołkami, do których poszczególne segmenty są doczepione. Działanie algorytmu, czyli krzyżowanie, mutacja i wybór najlepszych osobników sprowadza się do systematycznego dopasowywania segmentów do wierzchołków wyznaczanych przez ułożenie wzorcowe. Umożliwia to znajdowanie coraz lepszych, lepiej dopasowanych osobników oraz jednoznaczną eliminację osobników słabiej przystosowanych.

Algorytm:

Na początku program zapamiętuje wszystkie wierzchołki ułożenia docelowego (tzw. Multisegment). Wierzchołki te (poprzez współrzędne) wyznaczają konkretne segmenty. W jednym wierzchołku może (ale nie musi) zbiegać się 2 lub więcej segmentów. Wierzchołki te (zwane dalej wierzchołkami docelowymi) będą służyły do przyczepiania do nich wierzchołków poszczególnych segmentów. Przyporządkowanie segment-wierzchołek reprezentowane jest przez trójkę (wierzchołek docelowy, jeden z wierzchołków segmentu, id segmentu).

W pierwszym kroku algorytmu tworzona jest generacja n osobników. Tworzenie odbywa się poprzez wylosowanie dla każdego z segmentów jednego z jego wierzchołków i przyporządkowanie go do jednego z wierzchołków docelowych. Przyporządkowania mogą się powtarzać, tj. 2 różne segmenty mogą być doczepione do tego samego wierzchołka docelowego.

Po stworzeniu populacji początkowej, algorytm rozpoczyna krzyżowanie i mutację. W tym celu wybiera z populacji początkowej pewną liczbę x osobników (od 10% do 50% liczby wszystkich osobników), które będą podlegały krzyżowaniu. Następnie przeprowadza krzyżowanie na zasadzie „każdy-z-każdym”. Krzyżowanie 2 osobników przebiega następująco:

algorytm na podstawie oceny stopnia adaptacji każdego z dwójki osobników-rodziców wybiera z nich odpowiednią liczbę przyporządkowań, które zostaną włączone do nowo powstałego osobnika. Im lepsza adaptacja danego osobnika, tym więcej trójek zostanie przez niego przekazanych. Przykładowo, jeśli osobnik 1 jest zaadaptowany w 15% a osobnik 2 w 85%, to do powstającego z krzyżowania tej pary osobnika zostanie przekazana jedna (pierwsza) trójka z osobnika 1, a pozostałe trójki zostaną przekazane z osobnika 2. Pozwala to na premiowanie osobników lepszych, które mają większe szanse na to, że doprowadzą do pożądanego ułożenia.

Krzyżowanie każdy-z-każdym x osobników tworzy (x-1)*(x-1) nowych osobników. Następnie na każdym z nowo powstałych osobników wykonywana jest mutacja. Przebiega ona dla każdego osobnika w sposób następujący:

losowane jest jedno z połączeń, tj. przyporządkowanie jednego z segmentów do jednego z wierzchołków docelowych. Algorytm losuje następnie, do którego wierzchołka (w które miejsce) dany segment ma zostać przeniesiony. Mutacja polega zatem na przeniesienie jednego elementów w inne miejsce na planszy. Wierzchołek, którym segment zostanie przyczepiony do wierzchołka docelowego także jest losowany.

Po wykonaniu zarówno krzyżowania jak i mutacji powstałe w ich wyniku nowe osobniki-dzieci (ich liczba to wciąż (x-1)*(x-1)) zostają dołączone do populacji osobników-dorosłych. Następnym krokiem jest wybór spośród osobników (tak dorosłych jak i dzieci) zbioru osobników najlepszych, o liczności równej liczności populacji początkowej. Ponownie sprawdzane (i aktualizowane) są stopnie adaptacji poszczególnych osobników, po czym następuje wybór spośród nich najlepszych. Tworzą one nową generację. Następuje sprawdzenie, czy w tej generacji wystąpił osobnik odpowiadający oczekiwaniom, tj. o odpowiednim stopniu adaptacji. Jeśli tak, algorytm kończy swoje działanie a program prezentuje wyniki i statystyki. Jeśli nie, algorytm powraca do kroku krzyżowania i mutacji.

Aby uniknąć blokowania się algorytmu, czyli sytuacji, w której kolejne krzyżowania i mutacje nie powodują utworzenia żadnych nowych, lepszych osobników, algorytm dopuszcza dodatkową mutację. Realizacja tejże mutacji ma miejsce w momencie, gdy przez 5 kolejnych generacji adaptacja całej generacji (czyli stopień adaptacji najlepszego osobnika z danej mutacji) nie zmienia się i pozostaje na tym samym poziomie. W takiej sytuacji wszystkie osobniki o adaptacji równej lub bliskiej (z dokładnością do 0.1) adaptacji najlepszego osobnika ulegają dodatkowej mutacji zgodnej z przedstawioną w poprzednim akapicie zasadą.

Ocena adaptacji osobnika opiera się na porównywaniu pola zajmowanego przez danego osobnika z polem na planszy zajmowanym przez osobnika docelowego (czyli przez poprawnie ułożony obrazek). Im bardziej oba pola pokrywają się, tym lepsza jest adaptacja danego osobnika.

Struktura programu:

Program napisany został w języku Java. Do obliczania powierzchni przecinania się pól zajmowanych przez segmenty wykorzystana została biblioteka JTS Topology Suite (http://www.vividsolutions.com/jts/jtshome.htm). Program został podzielony na bloki, każdy z nich reprezentowany przez osobną klasę:

Population – klasa przechowująca wszystkie aktualnie przetwarzane osobniki. Zawiera informację na temat aktualnej generacji, listę wierzchołków docelowych, listę wszystkich osobników danej generacji oraz udostępnia funkcje służące do krzyżowania, mutowania i wyboru najlepszych osobników.

Entity – klasa reprezentująca pojedynczego osobnika. Przechowuje informacje na temat generacji, w której została utworzona, stopień adaptacji oraz listę połączeń (patrz: klasa Connector) segmentów z wierzchołkami docelowymi.

Segment – klasa reprezentująca pojedynczą figurę geometryczną. Zawiera listę wierzchołków (czyli współrzędne opisujące figurę) oraz id segmentu.

MultiSegment – klasa reprezentująca docelowe ułożenie. Wykorzystywana przy sprawdzaniu adaptacji. Przechowuje listę wierzchołków docelowych.

Connector – klasa reprezentująca omawianą w algorytmie trójkę (wierzchołek docelowy, wierzchołek segmentu, id segmentu). Wykorzystywana zarówno w procesie krzyżowania jak i mutacji.

Vertex – przechowuje koordynaty danego wierzchołka oraz numer będący id tego wierzchołka.

Powyższe klasy reprezentują interfejs algorytmu i wszystkie wykonywane w nim czynności.

Pozostałe klasy służą do pobierania oraz wyświetlania wyników działania algorytmu. Klasy te to:

Main – główna klasa programu, jej zadaniem jest pobieranie danych wejściowych oraz wywoływanie odpowiednich funkcji generujących i sprawdzających adaptacje.

Reader – klasa wczytująca z podanego pliku dane według reguł zapisanych w pliku inputfilesyntax.txt i tworząca z nich obiekt klasy MultiSegment.

Painter – klasa wyświetlająca na ekranie wyniki działania algorytmu, tj. najlepszego w każdej generacji osobnika oraz zarys ułożenia docelowego.

Statistics – klasa gromadząca informacje na temat przebiegu algorytmu, takie jak ilość generacji, czas wykonania i stopień adaptacji najlepszego osobnika z danej generacji.

Wadą wykorzystania przytoczonej powyżej biblioteki JTS jest dość długi czas działania programu. Wynika to z wewnętrznej implementacji tejże biblioteki oraz faktu, że zarówno etapy krzyżowania jak i mutacji wymagają tworzenia wielu nowych obiektów klas dostarczanych przez tę bibliotekę. Z uwagi jednak na problem oceny adaptacji (przystosowania) osobnika, którą to ocenę ta biblioteka znacząco ułatwia, została ona, pomimo wad, wykorzystana w projekcie.

Instrukcja do uruchomienia programu:

Program został spakowany do postaci archiwum .JAR. Otworzenie go normalne, tj. przez dwukrotne kliknięcie spowoduje uruchomienie algorytmu dla predefiniowanego obrazka. Dostępne są 4, utworzone wcześniej obrazki, na których algorytm może prezentować swoje działanie. Aby uruchomić pozostałe, konieczne jest uruchomienie programu z linii poleceń, komendą:
java -jar PSZT-TANGRAM.jar generation*X* <warunek>, gdzie *X* reprezentuje liczbę od 1 do 4. <warunek> jest opcjonalnym, niewymaganym parametrem reprezentującym oczekiwany przez użytkownika stopień adaptacji, po osiągnięciu którego program ma zakończyć swoje działanie. Zatem dostępne tryby uruchomienia programu to:

- z jednym parametrem, np.:

  java -jar PSZT-TANGRAM.jar generation2

- z dwoma parametrami, np.:

  java -jar PSZT-TANGRAM.jar generation4 0.95

W wersji jedno- lub bezparametrowej pożądany stopień adaptacji ustawiany jest na 0.99.

Wnioski:

Wykonany i zaimplementowany w toku tworzenia projektu algorytm wykonuje ruchy prowadzące do wygenerowania oczekiwanego osobnika, reprezentującego w przypadku gry TANGRAM odpowiednie ułożenie segmentów, zgodnie z ułożeniem docelowym. Wykonywane w każdej generacji operacje krzyżowania i mutacji zgodnie z oczekiwaniami powodują powstawanie coraz lepszych osobników (co widoczne jest w statystykach), co po określonej liczbie generacji skutkuje wytworzeniem pożądanego ustawienia. Dla założonej w fazie wstępnej 100-osobnikowej populacji wystarczającą liczbą generacji jest kilkanaście (czasami kilkadziesiąt) generacji. Konieczność wprowadzenia dodatkowej mutacji obrazuje jednak, iż algorytmy ewolucyjne nie zawsze są skuteczne w 100%. Istnieją sytuacje, w którym kolejne krzyżowania i mutacje nie powodują powstania lepszego od poprzednich osobnika. Wtedy taki algorytm nie jest w stanie znaleźć oczekiwanego osobnika i zawiesza się. Wprowadzenie wspomnianej już dodatkowej mutacji eliminuje ten problem. Algorytmy ewolucyjne okazują się być zatem bardzo dobrym, choć nie zawsze skutecznym, sposobem na rozwiązywanie problemów wymagających przeszukiwania zbiorów i optymalizacji wyników.