General Scheduling Problem

Realizacja na podstawie Genetic Algorithm Framework

Spis Treści:

Rozdział 1

Wstęp

GAF – Genetic Algorithm Framework

Jest to specjalny framework wykorzystujący algorytm genetyczny. Za jego pomocą należało rozwiązać problem przydziału prac (General Scheduling Problem). Na samym starcie powstaje kilka pytań: Co to znaczy, że to „genetyczny" algorytm? Oraz Jak połączyć coś „genetycznego" z problemem, który wydaje się na pierwszy rzut oka być z innej dziedziny niż biologia?

Rozdział 2

Opis idei algorytmu GAF

Algorytm genetyczny (mówiąc w skrócie) bazuje na pojęciach związanych z genetyką. Możemy zatem spotkać się ze słowami takimi jak: „chromosom", „gen", „populacja". Szybko okaże się jednak, że pojęcia te są używane tylko w celu swobodniejszej komunikacji między programistami piszącymi w tej bibliotece. Dzięki tym pojęciom łatwiej określić, co kto ma na myśli. Nie wyklucza to oczywiście, że za pomocą biblioteki GAF mogą powstać programy związane prawdziwie z biologią. Nawet można by powiedzieć, że biolodzy przez podobieństwo pojęć mają większy przywilej do korzystania z danego algorytmu ;). Wracając do rzeczy, co właściwie rozumiemy przez pojęcia takie jak „chromosom", „gen", „populacja". Najprościej będzie je wytłumaczyć wgłębiając się właśnie w świat biologii. Zatem po kolei:

2.1 Chromosom

Mając naprawdę znikomą wiedzę o biologii napiszę tylko tyle, że chromosomy są strukturami kodującymi materiał genetyczny wewnątrz komórek. To właśnie o nich mowa kiedy mówimy o dziedziczeniu po rodzicach materiału genetycznego. Jeżeli urodziliśmy się zdrowi oznacza to, że odziedziczyliśmy 23 pary chromosomów po naszych rodzicach (po 23 chromosomy od każdego z nich). Chromosomy zawierają geny. Zatem w podobieństwie do świata informatycznego i pewnego wzorca architektonicznego algorytmu GAF, przez chromosom będziemy rozumieli pewien zestaw cech jakie możemy odziedziczyć lub zmodyfikować w miarę upływu czasu między kolejnymi pokoleniami.

2.2 Gen

Gen jest odcinkiem DNA warunkującym wystąpienie określonej cechy organizmu. Będziemy mogli zatem zrozumieć to pojęcie jako pewien item, który ma określone cechy. Zatem chromosom będzie zbiorem takich cech.

2.3 Populacja

W rzeczywistym świecie przez populację rozumiemy grupę osobników jakiegoś określonego gatunku żyjących równocześnie na tym samym obszarze. W algorytmie GAF możemy spotkać się z taką definicją populacji: „Populacja reprezentuje kolekcję możliwych rozwiązań". W praktyce jest to po prostu zbiór wielu chromosomów i o ile my ludzie mamy szczęście, że nikt nie określa nas takim chłodnym stwierdzeniem to w algorytmie genetycznym GAF populacja sprowadza się właśnie do takiego opisu :). Twórcy frameworka tak zaprogramowali kolekcję, że możemy utworzyć sobie populację o podanej liczbie osobników i długości chromosomów, które domyślnie są wartościami binarnymi – ciągiem skończonej ilości zero-jedynek (ciągi te są losowo kreowane).

. . .

Zatem mamy pewien obraz pojęć używanych w algorytmie GAF. Gen jest pewną częścią chromosomu, a chromosom jest częścią populacji. Pozostaje zagwost-

ka czemu ma służyć takie podobieństwo do świata biologii. Algorytm GAF ma za zadanie znaleźć zbliżone do optymalnego rozwiązanie danego problemu. Problem przydziału prac jest właśnie idealnym zadaniem dla niego. Spójrzmy na to tak: populacja wraz z upływem czasu zgodnie z teorią ewolucji, ewoluuje ku coraz lepszym osobnikom, czyli przedstawicielom swoich gatunków. Zatem mamy do czynienia z pewnym progresem, który z czasem jest coraz lepszy i lepszy. Spójrzmy na to jeszcze z innej strony: Od lat możemy obserwować wyścigi konne. Każdy dżokej pragnie osiągnąć jak najlepszy czas pokonania całego toru. W miarę upływu dziesięcioleci czasy te są coraz krótsze. Jest to, co prawda związane z coraz większą wiedzą biologiczną jak dbać o konie by mogły takie czasy uzyskiwać a nie tylko związane z powolną ewolucją koni w ramach ich gatunku. Jednak każdy z nas zdaje sobie sprawę z tego, że istnieje jakaś niejednoznaczna granica czasu, w którym dżokej na koniu może obiec jedno okrążenie toru wyścigowego. Raczej żadnemu nie uda się w 2 sekundy pokonać długości 2,3 km jednak możliwe jest dążenie do coraz lepszych czasów w jakiś tam bliżej nieokreślonych granicach. Podobną rzecz obserwujemy również w zawodach lekkoatletycznych i moglibyśmy wymieniać dużo innych dziedzin sportowych. Algorytm genetyczny jest więc takim (powiedziałbym) trenerem, który na podstawie wprowadzonych danych „wyciska" co tylko najlepszego się da ze swojego zawodnika – przedstawiciela zbioru chromosomów.

Rozdział 3

Opis podejścia i realizacji GSP

W problemie przydziału prac mamy do czynienia ze zbiorem genów, czyli zadań. Każde z zadań ma swoją nazwę i czas, który jest potrzebny na jego wykonanie. W celu przybliżenia problemu do rzeczywistych realiów postanowiłem na podobieństwo mojego poprzedniego projektu (również dotyczącego problemu przydziału prac przy użyciu algorytmu symulowanego wyżarzania) zastosować odpowiednie nazewnictwo klas i struktur używanych dodatkowo przeze mnie poza komponentami związanymi ściśle z architekturą GAF. Dlatego znane wcześniej „Fabryki" należy rozumieć jako zbiór osób, które mają za zadanie uporać się z takimi ciężkimi zadaniami jak:

- sweeping
- painting
- ironing
- shopping
- washing
- writing
- cutting
- programming
- colouring
- cleaning
- drawing
- cooking

Każda z prac ma określony czas realizacji (umownie liczony w godzinach). Umówmy się, że trzech siostrzeńców ma za zadanie uporać się z zadaniami, które przydzielił im pewien oskarżający ich o lenistwo wujek. Nazwijmy ich roboczo: Hyzio, Zyzio i Dyzio. Przez znalezienie optymalnego przydziału prac będziemy rozumieć sytuację, gdy wszystkie prace zostaną wykonane i żaden z siostrzeńców nie będzie musiał czekać zbyt długo na swojego kuzyna aż ten skończy swój przydział prac. Każdy chce zrealizować całą listę dwunastu zadań tak szybko, by wszyscy razem mogli wyjść pograć w piłkę najszybciej jak się da. Przez słabe rozwiązanie będziemy rozumieli sytuację gdy pierwszy z siostrzeńców skończy swoją pracę i będzie musiał długo czekać na ostatniego, który zrealizuje ostatnie swoje zadanie. Tą sytuację nazwałem zmienną „Difference", która w miarę znajdywania lepszych rozwiązań powinna maleć. Difference = [czas skończenia pracy przez ostatniego siostrzeńca] – [czas skończenia pracy przez najszybszego]

Rozdział 4

Generalny opis kodu

Na wzór poprzedniego projektu utworzyłem klasę „Factory", która będzie reprezentować jednego siostrzeńca. Zawiera ona pola takie jak: Imię (siostrzeńca), listę prac do wykonania i całkowity czas wykonania. Nazwałem go „lokalnym", ponieważ w miarę postępu funkcji CalculateWorkTime() czas ten będzie ulegał wielu modyfikacjom. Zacznijmy od funkcji Main():

Utworzyłem listę Fabryk (czyli moich siostrzeńców) wzorem z poprzedniego projektu. Sama lista jest statyczna bym mógł mieć do niej dostęp z każdego miejsca kodu (przydaje się to w funkcji CalculateWorkTime()). Następnie rozpoczynają się linie kodu ściśle związane z architekturą algorytmu GAF. Należy utworzyć populację i nadać jej ilość. Następnie w pętli dla każdego elementu w populacji tworzony jest chromosom, który to z kolei w kolejnej zagnieżdżonej pętli ładuje informacje o genach – czyli o liście prac do wykonania i wszelkimi z nią informacjami (itemy). Biblioteka GAF korzysta z czegoś podobnego, co mogliśmy obserwować we wnioskowaniu Baysowskim – a mianowicie obecność pewnego silnika, który odpowiedzialny jest za działanie całego frameworka (silnik GA). Uruchamia go dokładnie ta linia:

```
var ga = new GeneticAlgorithm(population,
                              CalculateFitness);
```

W ten sposób uruchamiany jest silnik Genetic Algorithm bazujacy na populacji i kluczowej funkcji Calcu-

lateFitness() - liczącej pewnego rodzaju dopasowanie na podstawie zadanego chromosomu (zestawu danych – w praktyce to nasza lista zadań).

Funkcja CreateFitness():

To w niej będę starał się za pośrednictwem funkcji CalculateWorkTime() wyznaczyć optymalne rozwiązanie dla podziału prac. Sama zaś funkcja zwraca dopasowanie – fitness. Ciężko to przetłumaczyć ponieważ samo słowo angielskie lepiej oddaje, co ma zwracać i jakoś praktyczniej przybliża o co właściwie chodzi. Chodzi o to (na tyle ile rozumiem algorytm GAF), by wyznaczyć jakąś miarę, która będzie opisywać zbliżenie się do coraz lepszego rozwiązania. Wartość fitness zgodnie z życzeniem twórców algorytmu musi mieścić się w przedziale [0,1]. Postanowiłem więc, że za 1 zrozumiem osiągnięcie optymalnego rozwiązania dla moich danych a każda wartość mniejsza od jedynki i zbliżająca się do niej będzie oznaczać ile jeszcze brakuje danemu rozwiązaniu do bycia tak „super idealnym". Osiągam to za pomocą rachunku: 1 / (optimumTime + 1). Wartość optimumTime jest właśnie tą „różnicą" wspomnianą przeze mnie wcześniej w rozdziale „Opis podejścia realizacji GSP", która na outpucie programu nazwana jest słowem „Difference". Cały mechanizm wyznaczania tej wartości można prześledzić w funkcji CalculateWorkTime() oraz w klasie Work.cs w metodzie AssignVarianceForFactories(). Pożądaną wartością zmiennej optimumTime jest wartość 0. Zatem zwrócona wartość przez funkcję CalculateFitness będzie równa wówczas: 1 / (0 + 1) = 1.

Ostatnim zbiorem ważnych metod związanych z algorytmem genetycznym są:

- ga_OnRunComplete()
- ga_OnGenerateComplete()
- Terminate()

Pierwsze dwie odpowiedzialne są za wyświetlanie danych. ga_OnRunComplete() wywoływana jest przez silnik GA po zakończeniu wszelkich generacji populacji. Jest takim wielkim „Amen" dla całego programu. Funkcja ga_OnGenerateComplete() wywoływana jest przez silnik GA jak refren po zakończeniu analizowania jednej generacji. Czym jest ta generacja? Tutaj należy przejść do funkcji Terminate(), w której możemy określić wartość liczbową generacji. Im większa jest ta liczba tym więcej algorytm nagłowi się by w ramach jednej generacji znaleźć lepsze rozwiązania problemu przydziału prac. Podobnie zresztą ma się zmienna population, która również wpływa na szybkość znalezienia coraz lepszych rozwiązań w obrębie już może nie kilkunastu ale kilku generacji. Obie zmienne są ze sobą w ten sposób skorelowane.

Rozdział 5

Zakończenie

Program został napisany na podstawie oficjalnej strony frameworka GAF, na podstawie przykładowego kodu dla problemu Travelling Salesman. Natomiast w tym projekcie wzorem poprzedniej implementacji General Scheduling Problem zaprogramowałem problem w taki sposób by możliwe było podać na starcie programu liczbę fabryk przetwarzających dane. Tradycyjnie informacja o ich ilości została umieszczona w statycznej zmiennej „K". Na outpucie programu wyświetlam za pomocą zmiennej „bestWay" osobistą listę swoich prac dla każdego z siostrzeńców. Można w ten sposób sprawdzić poprawność otrzymanych wyników.