Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki
Kierunek Informatyka
Studia stacjonarne II stopnia

Praca dyplomowa magisterska

Analiza porównawcza algorytmów metaheurystycznych do rozwiązywania wybranego problemu optymalizacyjnego

Kierujący projektem:
dr inż. Henryk Josiński

Autor:
Sebastian Nalepka

Gliwice 2017

Spis treści

Wstęp

W otaczającym nas świecie obecnych jest wiele problemów, z którymi zmagają się ludzie. Część z nich jest problemami prostymi, do rozwiązania których wystarczy wyłącznie nieduży nakład czasu. Istnieją jednak problemy trudniejsze, które wymagają długotrwałych przemyśleń i obliczeń po których nie zawsze otrzymujemy najlepsze rozwiązanie. Do prostych problemów możemy zaliczyć codzienne decyzje podejmowane przez każdego człowieka dotyczące. Dla przykładu jeśli chcemy dojechać z punktu A do punktu B komunikacją miejską, wystarczy, że sprawdzimy rozkład jazdy i wybierzemy połączenie, które będzie pasowało nam w kontekście godziny przyjazdu na dane miejsce. Przedstawiony problem posiada z goła inną skalę trudności ze strony przewoźnika. Wyznaczenie optymalnych tras przewozowych dla określonej liczby środków transportu, w celu obsłużenia danego zbioru klientów, którzy rozlokowani są w różnych punktach jest kwestią bardzo skomplikowaną. Znalezienie optymalnych tras, które umożliwią przetransportowanie jak największej ilości osób w celu zmaksymalizowania zysku, przy zachowaniu możliwie najkrótszych tras, których zamysłem jest minimalizowanie kosztów poniesionych z transportem jest złożonym problemem, znanym jako jedna z odmian problemu marszrutyzacji, który jest z kolei rozwinięciem bardzo popularnego problemu komiwojażera polegającego na znalezieniu najkrótszej drogi łączącej wszystkie zdefiniowane uprzednio punkty, zaczynając i kończąc w tym samym miejscu. Problemy, do rozwiązania których potrzebne są ogromne nakłady obliczeniowe definiowane są jako problemy optymalizacyjne. W problemach takich liczba możliwych rozwiązań w przestrzeni poszukiwań z reguły jest tak duża, że niemożliwe jest przeszukiwanie wyczerpujące w celu znalezienia najlepszego z nich.

Cel pracy

Celem pracy jest dokonanie analizy porównawczej algorytmu symulowanego wyżarzania, algorytmu genetycznego oraz roju cząstek, która przeprowadzona będzie dzięki zaimplementowanej dedykowanej aplikacji bazodanowej, której przeznaczeniem jest zautomatyzowanie procesu szukania minimum globalnego zadanych dla

funkcji testowych.

Zawartość pracy

Dla osiągnięcia wyżej wymienionego celu zrealizowana została praca składająca się z siedmiu rozdziałów oraz wniosków przeprowadzonych doświadczeń wraz z podsumowaniem. W rozdziale pierwszym przedstawiony został wybór problemu optymalizacyjnego wraz z jego opisem oraz listą funkcji testowych, które użyte zostały do przeprowadzenia badań. Rozdział drugi przedstawia możliwe sposoby rozwiązania wybranego problemu optymalizacyjnego wraz z opisem użytych algorytmów metaheurystycznych. Na bazie dwóch pierwszych rozdziałów utworzony został w rozdziale trzecim projekt automatyzacji przeprowadzanych doświadczeń, na podstawie którego bazowała budowana aplikacja. Kolejny rozdział to zastosowane rozwiązania technologiczne oraz opis wykorzystanych technologii, bibliotek zewnętrznych, a także narzędzi. Na ich podstawie utworzona została architektura aplikacji bazodanowej, której specyfikacja umiejscowiona została w rozdziale piątym. Kolejny rozdział prezentuje już ściśle aspekt badawczy, który zawiera opis metod porównawczych zastosowanych algorytmów oraz wyniki doświadczeń dla użytych funkcji testowych, których analiza przeprowadzona została w rozdziale siódmym zawierającym wnioski.

1. Wybór badanego problemu optymalizacji

1.1. Opis minimalizacji funkcji ciągłych wielu zmiennych

W prezentowanej pracy dokonano porównania algorytmów metaheurystycznych odnosząc się do problemu minimalizacji funkcji wielu zmiennych. Problem ten polega na znalezieniu minimum globalnego rzeczywistej funkcji poprzez systematyczne wybieranie parametrów wejściowych z dozwolonego zakresu i obliczaniu ich wartości. Trudność problemu sprowadza się do wielkości przestrzeni przeszukiwania. Traktując problem jako czysto matematyczny, funkcja n zmiennych posiada nieskończenie wiele wartości w każdym wymiarze. Mamy więc nieskończenie wielką przestrzeń poszukiwań. Biorąc jednak pod wzgląd aspekt technologiczny i to, iż komputery bazują na danych skończonych, można w prosty sposób przedstawić skalę trudności. W czasie implementacji algorytmu, którego celem jest znalezienie minimum globalnego funkcji, należy wziąć pod uwagę dostępną dokładność obliczeniową maszyny. Zakładając, iż dokładność ta wynosi osiem miejsc po przecinku, każda zmienna, ograniczona w przedziale $[0, 100]$ może przyjąć $100 * 10^8$ różnych wartości. Już dla funkcji dwóch zmiennych, wielkość przestrzeni przeszukiwania wynosi $(100 * 10^8)^2 = 10^{20}$.

1.2. Funkcje testowe

Problem minimalizacji funkcji ciągłych bardzo dobrze nadaje się do porównania algorytmów metaheurystycznych przez wzgląd na powszechnie dostępne funkcje testowe. Funkcje te, posiadają pewny specyficzny element, dzięki któremu możliwe jest porównanie wyników otrzymanych przez dany algorytm. Element ten to znajomość minimum globalnego dla danej funkcji testowej. Dzięki znajomości wartości najlepszej (najmniejszej) dla danej funkcji, wiadomo jak szybko oraz czy w ogóle badany algorytm znalazł rozwiązanie. Posiadając tą informację można zestawić otrzymane rezultaty wszystkich algorytmów pod kątem czasowym lub liczby wyliczeń wartości funkcji dla ustalonych przez algorytm punktów. Do analizy wybranych zostało pięć funkcji testowych, których dobór brał pod uwagę stopień ich skomplikowania. Każda

z funkcji posiada specyficzne właściwości, które zostaną wzięte pod uwagę podczas porównania rezultatów algorytmów heurystycznych.

1.2.1. Funkcja Bochachevsky'ego N. 1

1.2.2. Funkcja Rosenbrocka

1.2.3. Funkcja Easoma

1.2.4. Funkcja Eggholdera

1.2.5. Funkcje Griewanka

1.2.6. Funkcje testowe

Opis funkcji wraz z ich właściwościami opisać po definitywnym ich wyborze (poczekać na fazę testów)

2. Metody rozwiązania wybranego problemu optymalizacji

2.1. Algorytmy dokładne

Klasycznym podejściem do znalezienia minimalnej wartości zadanej funkcji testowej jest próba porównania wartości funkcji dla każdych możliwych parametrów wejściowych i wybrania w ten sposób optymalnego rozwiązania. W rzeczywistości jednak takie rozwiązanie nie jest praktyczne przez wzgląd na olbrzymią możliwą liczbę takich parametrów. Jak już wspomniano w rozdziale 1.1 liczba rozwiązań dla funkcji ograniczonej już do dwóch zmiennych może wynieść 10^{20}, a w przypadku trzech zmiennych – 10^{30}. Zakładając, iż możliwe by było wyliczenie miliarda wartości funkcji na sekundę, to w godzinę wartość ta wynosiłaby $3.6 * 10^{12}$, a w rok – $3.2 * 10^{16}$. Prowadząc dalej obliczenia wychodzi, iż wyliczenie 10^{20} wartości funkcji trwałoby około 3125 lat. Widać, iż liczba kombinacji jest tak duża, iż takie podejście jest niemożliwe do wykonania w akceptowalnym czasie.

2.2. Metaheurystyki optymalizacyjne

Analizując podejście z rozdziału 2.1, może przyjść na myśl sposób, który polegać będzie na wyliczaniu wartości funkcji dla wyrywkowych parametrów. Skąd jednak wiadomo które punkty wybrać? Na podstawie czego bazować? W przypadku problemów, w których przez wzgląd na czas niemożliwe jest dojście do rozwiązania na ratunek przychodzą algorytmy heurystyczne, które umożliwiają skrócenie czasu obliczeń. Ceną którą trzeba jednak za to zapłacić jest otrzymanie potencjalnie gorszego rozwiązania od rozwiązania najlepszego. Samo pojęcie heurystyki pochodzi od greckiego słowa heuresis, które znaczy ‘odnaleźć’. Metody heurystyczne polegają na użyciu reguł oraz faktów, które uzyskane na drodze badania danego problemu, umożliwiają jego rozwiązanie lub zbliżenie się do poprawnej odpowiedzi. Podejście heurystyczne stosowane może być w sposób piętrowy, tworząc metaheurystyki. Metaheurystyka jest to ogólny algorytm do rozwiązywania problemów obliczeniowych,

który inspirację często bierze z mechanizmów biologicznych lub fizycznych. Określenie to oznacza tak zwaną heurystykę wyższego poziomu, co wynika z faktu, iż algorytmy tego typu bezpośrednio nie rozwiązują żadnego problemu, a wyłącznie podają metodę na utworzenie odpowiedniego algorytmu.

2.2.1. Metoda optymalizacji rojem cząstek

Metoda roju cząstek (PSO – Particle Swarm Optimization) jest przykładem optymalizacji z kategorii metod inteligencji stadnej. Powstała ona w wyniku inspiracji biologicznej, której źródłem był układ lotu stada ptaków tworzony w celu znalezienia pożywienia lub gniazda oraz uniknięcia drapieżników. Zastosowanie prostych zasad umożliwia ptakom zsynchronizowany oraz bezkolizyjnych ruch, który daje efekt zachowania jednego organizmu. Ruch stada ptaków, czy ławicy ryb jest wypadkową działania wszystkich osobników i koncentruje się na utrzymaniu optymalnego dystansu od swoich sąsiadów, przy jednoczesnym podążaniem za liderem. Badania nad optymalizacją roju cząstek zapoczątkowano od próby graficznego zasymulowania zachowań takich grup. Bardzo szybko okazało się, iż stworzony matematyczny model może być również zastosowany jako metoda optymalizacyjna. W optymalizacji rojem cząstek rozwiązania (cząstki) współpracują ze sobą w celu odnalezienia cząstki optymalnej. W czasie procesu optymalizacji następuje zmiana położenia każdej cząstki w przestrzeni rozwiązań poprzez wyznaczenie wektora prędkości. Wektor ten jest modyfikowany przy użyciu informację o historii poszukiwań danej cząstki oraz jej cząstek sąsiednich. Metoda PSO w problemie optymalizacji funkcji wielowymiarowych dąży do otrzymania cząsteczki, która reprezentuje jak najmniejszą wartość funkcji i może być opisana dwoma równaniami:

$$v = W * v + c_1 * r_1 * (p - x) + c_2 * r_2 * (g - x) \tag{1}$$

gdzie,

v - aktualny wektor prędkości cząstki

W – parametr z zakresu $[0, 1]$, który determinuje wpływ poprzedniego położenia cząstki na jej obecną pozycję

p - najlepsze rozwiązanie dla cząstki

g – najlepsze rozwiązanie dla sąsiedztwa cząstek

r_1, r_2 – losowe liczby z zakresu $[0, 1]$

c_1, c_2 – parametry skalujące z zakresu $[0, 1]$

Nawiązując do powyższych równań, każda cząstka roju przeszukuje przestrzeń rozwiązań, zmieniając położenie na podstawie swoich najlepszych rozwiązań p_i, jednocześnie wykorzystując informację o najlepszym rozwiązaniu w sąsiedztwie p_t. Parametry skalujące umożliwiają kontrolę wpływu danych części wektora prędkości na wynik. W przypadku, w którym c_1 będzie równe zero, cząstka będzie wykorzystywała tylko i wyłącznie informację o najlepszym rozwiązaniu w roju. Z kolei jeśli wartość parametru c_2 zostanie ustawiona na zero, cząstka będzie poszukiwała rozwiązania samodzielnie, bez uwzględnienia rozwiązań, które uzyskane zostały przez inne cząstki.

2.2.2. Symulowane wyżarzanie

Algorytm symulowanego wyżarzania po raz pierwszy został opisany w 1953 roku przez Nicolasa Metropolisa. Sposób działania algorytmu jak i również jego nazwa odnosi się do procesów fizycznych, które wykorzystywane są w metalurgii. Proces wyżarzania polega na rozgrzaniu ciała stałego do określonej temperatury, a następnie jego powolnym studzeniu. Konsekwencją tego działania jest zmiana struktury krystalicznej materiału, który poddany został wyżarzaniu. W czasie procesu ochładzania metali dostrzeżono, iż cząsteczki ciała wraz z jego powolnym schładzaniem tworzą bardziej regularne struktury, niż w przypadku szybszego obniżenia temperatury, kiedy to chłodzone cząsteczki nie potrafią znaleźć optymalnego położenia. Algorytm symulowanego wyżarzania jest usprawnieniem starszych metod iteracyjnych, które polegały na ciągłym ulepszaniu istniejącego rozwiązania do momentu braku możliwości jego poprawy. W metodach tych zatrzymanie algorytmu mogło nastąpić przy rozwiązaniu pseudo-optymalnym – lokalnym minimum. Nie istniała wówczas możliwość wyjścia z owego lokalnego minima i kierowania się w kierunku minimum globalnego. Bardzo ważną cechą opisywanego algorytmu jest możliwość wyboru, z pewnym prawdopodobieństwem, gorszego rozwiązania. Dzięki temu problem utknięcia w lokalnym minimum nie jest już groźny. Za wybór gorszego rozwiązania ma wpływ podstawowy parametr przeniesiony z podstaw termodynamicznych algorytmu – temperatura. Im jest ona wyższa, tym większe istnieje prawdopodobieństwo wyboru i zaakceptowania gorszego rozwiązania. W czasie działania algorytmu, temperatura obniża się i działanie algorytu zbliża się w ten sposób do typowych metod iteracyjnych. W celu wykonania algorytmu symulowanego wyżarzania w kontek-

ście optymalizacji funkcji wielu zmiennych należy na początku losowo wygenerować punkt startowy, który mieści się na płaszczyźnie poszukiwań, wyliczyć dla niego wartość funkcji oraz wybrać maksymalną temperaturę startową z dostępnego zakresu [0,100]. Każda iteracja polega na wyborze losowego rozwiązania z sąsiedztwa, wyliczenia dla niego wartości funkcji i porównaniu z obecnie najlepszym rezultatem oraz obniżeniu temperatury. W przypadku, w którym wartość funkcji nowego punktu jest mniejsza (lepsza), jest on zaklasyfikowany jako najlepszy. W przeciwnej sytuacji punkt nie jest natychmiastowo odrzucany. Algorytm akceptuje gorsze rezultaty bazując na funkcji akceptacyjnej, która prezentuje się następująco:

$$\frac{1}{1 + exp(\frac{\Delta}{max(T)})} \tag{2}$$

gdzie,

Δ – różnica wartości starego i nowego punktu

T – wartość temperatury

W sytuacji, w której Δ i T są wartościami dodatnimi, prawdopodobieństwo akceptacji mieści się pomiędzy 0 i $\frac{1}{2}$. Niższa temperatura prowadzi do mniejszego prawdopodobieństwa zaakceptowania gorszego rezultatu. Podobnie jest z deltą – im większa delta tym mniejsza szansa na zaakceptowanie.

2.2.3. Algorytm genetyczny

Model algorytmu genetycznego po raz pierwszy zaprezentowany został w 1975 roku przez Johna Hollanda, który w pracy „Adaptation in Natural and Artifical Systems" przedstawił fundamenty założeń dotyczących adaptacji darwinowskiej teorii ewolucji w systemach informatycznych. W opisie algorytmu genetycznego posługuje się powszechną terminologią biologiczną. Z tego też powodu mówi się, iż algorytmy genetyczne przetwarzają populację osobników, którzy reprezentują rozwiązanie danego problemu. Każdy element populacji nazywany jest chromosomem, a jego składowe genami. Allele z kolei, są to możliwe stany (wartości) genu, które umiejscowione są na pozycjach zdefiniowanych jako locus. W badanych modelach komputerowych, osobniki (chromosomy) mogą być opisane jako różne struktury – zaczynając na łańcuchach binarnych, a kończąc na bardzo złożonych obiektach. W określonej iteracji zwanej zamiennie pokoleniem albo generacją, dane chromosomy na bazie określonej miary ich dostrojenia podlegają ocenie. Ocena ta skutkuje wyborem najlepiej

przystosowanych osobników, które wezmą udział w kolejnych iteracjach algorytmu. Nim jednak wybrane osobniki populacji utworzą nową generację, zostają poddane modyfikacjom spowodowanym podstawowymi operacjami genetycznymi – krzyżowaniem, selekcją oraz mutacją. W kontekście problemu optymalizacji funkcji wielu zmiennych, inicjalizacja algorytmu genetycznego polega na wygenerowaniu populacji początkowej, która złożona jest z określonej liczby chromosomów. Każdy chromosom reprezentowany w populacji posiada taką samą długość, która ustalona jest zależnie od rozwiązywanego problemu na etapie implementowania algorytmu. Przed nastąpieniem etapu generowania musi być jednak określony sposób kodowania informacji zawartej w chromosomie, która dotyczy rozwiązania. W algorytmie Hollanda nie było domyślnie zdefiniowanego sposobu kodowania chromosomów. Powszechnie uznaje się jednak, iż w algorytmie genetycznym stosuje się kodowanie binarne. Takie też kodowanie jest zastosowane w kontekście omawianego problemu optymalizowania funkcji wielu zmiennych. Kolejnym etapem, który następuje po wygenerowaniu populacji początkowej oraz wyborze kodowania chromosomów jest wyznaczenie jakości chromosomów danej populacji. W tym celu obliczana jest wartość tak zwanej funkcji oceny, która definiuje poziom dopasowania konkretnego chromosomu. Tym sposobem można stwierdzić, które chromosomy lepiej rozwiązują dane zagadnienie, a które gorzej. Znalezienie rozwiązania danego problemu sprowadza się do znalezienia ekstremum wspomnianej funkcji oceny. Kolejną częścią algorytmu jest zastosowanie mechanizmu selekcji, który definiuje sposób wyboru rozwiązań rodzicielskich, z których tworzone będą tak zwane rozwiązania potomne użyte w następnej generacji. Podstawowy algorytm genetyczny w operacji selekcji stosuje metodę ruletki. Metoda ta polega na przydzieleniu każdemu chromosomowi z danej populacji prawdopodobieństwa według wzoru:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{3}$$

gdzie, f_i - wartość funkcji oceny chromosomu i-tego p_i – prawdopodobieństwo reprodukcji

W celu wybrania puli rodzicielskiej, koło ruletki o obwodzie jeden dzielone jest na części o długości p_i, a następnie z zakresu $[0, 1]$ losowana jest liczba, która jednoznacznie identyfikuje punkt na ruletce, a co za tym idzie konkretny chromosom. Chromosom ten brany będzie pod uwagę w procesie następnej reprodukcji, a lo-

sowanie powtarzane jest tak długo, aż wylosowana zostanie ustalona liczba chromosomów. W problemie minimalizacji funkcji wielu zmiennych istnieje możliwość, iż wartość funkcji oceny będzie ujemna. W celu zniwelowania problemu ujemnego prawdopodobieństwa powyższy wzór został zmodyfikowany stosując skalowanie przystosowania:

$$p_i = \frac{f_i - f_{min}}{\sum_{j=1}^{N} f_j - f_{min}} \tag{4}$$

gdzie f_{min} jest wartością funkcji przystosowania najgorszego chromosomu.

Po etapie selekcji pozostaje do zdefiniowania kwestia wymiany pokoleń. W implementacjach często stosowana jest metoda całkowitego zastępowania, w której cała aktualna populacja, podlega operacjom krzyżowania i mutacji. Innym sposobem jest metoda zastępowania częściowego, w której część najlepszych chromosomów obecnej populacji przechodzi do populacji potomnej beż żadnych zmian, a pozostałe elementy z kolei biorą udział w operacji krzyżowania i mutacji. Często stosowaną praktyką jest zastosowanie zastępowana elitarnego, w którym na podstawie parametru określającego wielkość elity, część najlepszych osobników jest kopiowana do nowej generacji już na samym początku. Umożliwia to 'pamiętanie' najlepszych chromosomów które mogły być zmienione poprzez działanie operatorów genetycznych.

Pierwszym z takich operatorów jest krzyżowanie, które jest operacją umożliwiającą tworzenie nowych rozwiązań. Jego koncept bazuje na procesie rozmnażania organizmów, w czasie trwania których dziecko dziedziczy część genów rodziców. W kontekście omawianego algorytmu genetycznego, krzyżowanie polega na przecięciu chromosomów w ustalonym punkcie i ich wzajemnego zamienienia. Drugim operatorem genetycznym jest mutacja, która umożliwia wprowadzenie nowego elementu do populacji poprzez tworzenie różnorodności. Analogicznie jak w otaczającym nas świecie, tak i w algorytmie genetycznym, mutacje zdarzają się rzadko. Ich skala zazwyczaj zależy od parametru, który przyjmuje niskie wartości. W odniesieniu do chromosomów w postaci binarnej, mutacja może polegać na losowej zamianie losowego genu na wartość przeciwną. Tak utworzona nowa generacja ponownie przechodzi przez wszystkie punkty algorytmu. Dzieje się tak aż do czasu, w którym spełnione zostaną warunki zatrzymania, które w kontekście przedstawianego zagadnienia opisane zostały w rozdziale 6.1 opisującym metody porównawcze wybranych algoryt-

mów obejmujące w swojej treści warunki stopu opisanych trzech metaheurystyków.

3. Automatyzacja przeprowadzanych badań

4. Wykorzystane rozwiązania technologiczne

TODO

4.1. Zastosowane technologie

Obecny rozdział obejmuje opis wykorzystanych technologii, które umożliwiły implementację aplikacji. Opis ten dotyczy języka programowania, w którym napisany został backend aplikacji oraz silnika graficznego aplikacji wraz z językiem zapytań bazy danych.

4.1.1. .NET Framework/C#

.NET Framework jest to platforma programistyczna wydana przez firmę Microsoft w 2002 roku. Przeznaczona ona jest do wytwarzania oprogramowania przeznaczonego dla systemów operacyjnych z rodziny Windows. Główną składową przedstawianej platformy są kompilatory języków wysokiego poziomu, które umożliwiają przeprowadzenie operacji kompilacji programów napisanych w językach Visual Basic, F#, C++/CLI. Platforma .NET wspiera również jeden z najpopularniejszych języków programowania na świecie - C#, w którym napisana została aplikacja odnosząca się do prezentowanej pracy magisterskiej. Według rankingu TIOBE Software z grudnia 2015 roku, język C# uplasowany jest na piątym miejscu, spośród pięćdziesięciu najpopularniejszych języków, które zostały wzięte pod uwagę. C# jest to nowoczesny, zorientowany obiektowo język programowania stworzony przez firmę Microsoft. Jego pierwsza wersja wydana została już w połowie 2000 roku, przy dużej zasłudze głównego projektanta języka - duńskiego inżyniera oprogramowania Andersa Hejlsberga. Język ten może zostać użyty w celu pisania aplikacji webowych, desktopowych oraz przeznaczonych na urządzenia przenośne. Programy w nim napisane, kompilowane są do pośredniego kodu, który zapisany jest w CIL (ang. Common Intermediate Language) i wykonany w środowisku uruchomieniowym .NET Frameworka. Poziom trudności nauki C# uznawany jest za stosunkowo niski, głównie z powodu posiadania licznych udogodnień oraz modułów, które upraszczają

pracę programistyczną. Do elementów tych można zaliczyć brak konieczności dodawania plików nagłówkowych, które niezbędne były w języku C++, automatyczne zwalnianie dynamicznie przydzielonej pamięci za które odpowiedzialny jest Garbage Collection, inicjalizowanie zmiennych ich domyślnymi wartościami oraz wprowadzenie dodatkowych elementów składowych klas, takich jak indeksery oraz właściwości (ang. properties).

4.1.2. WPF / XAML

Windows Presentation Foundation (WPF) jest to silnik graficzny, który wprowadzony została wraz z trzecią wersją środowiska .NET. Okna w aplikacjach zaimplementowanych w WPF wyświetlane są za pomocą grafiki wektorowej, która to wspomagana jest przez akceleratory grafiki 3D. API w technologi WPF opiera się na języku XML, a konkretniej jego odmianie - XAML. Extensible Application Markup Language (XAML) jest to deklaratywny język znaczników, którego przeznaczeniem jest opis interfejsu użytkownika implementowanego w WPF. Pozwala on zaprojektowanie oraz rozmieszczenie wszelakich elementów wizualnych oraz umożliwia zrównoleglenie pracy programistów pracujących nad logiką biznesową budowanej aplikacji oraz grafików, którzy odpowiedzialni są za stworzenie graficznego interfejsu użytkownika. Zdarza się, iż graficy przez wzgląd na zakres swoich obowiązków nie znają żadnego języka programowania. Problem ten zanika dzięki językowi XAML, który pozwala na zrozumienie przez osoby nietechniczne zasady działania oraz powiązań poszczególnych okien, a także umożliwia projektowanie GUI w prosty sposób z poziomu drzewiastej struktury lub dedykowanego programu graficznego Expression Blend. Aplikacja ta umożliwia przeprowadzenie wszelkich operacji z poziomu graficznego środowiska.

4.1.3. TSQL

Transaction-SQL (T-SQL) jest to rozwinięcie standardowego języka SQL, który to utworzony został w latach siedemdziesiątych specjalnie na potrzeby relacyjnych baz danych przez firmę IBM. T-SQL pozwala na tworzenie konstrukcji takich jak instrukcje warunkowe i pętle oraz umożliwia stosowanie zmiennych. Aktualnie stosowany on jest do tworzenia zapytań bazodanowych przez firmy, które potrzebują bardziej zaawansowanych struktur niż tych, które dostępne są w SQL. T-SQL wprowadził możliwość stosowania bazodanowych wyzwalaczy, procedur oraz funkcji

składowanych, które to przy rozbudowanej strukturze tabel ułatwiają i zwiększają efektywność podczas pracy bazy danych.

4.2. Zastosowane narzędzia

W celu efektywnej pracy z technologiami, które opisane zostały w rozdziale 4.1 warto użyć dedykowanych narzędzi, które ułatwiają zastosowanie potencjału wspomnianych technologii. W tym celu w trakcie pracy nad prezentowanych projektem magisterskim, użytych zostało kilka narzędzi, które pozwoliły skrócić czas potrzebny na implementację aplikacji, przeprowadzenie badań oraz zagregowanie ich rezultatów.

4.2.1. Matlab

4.2.2. SQL Server Management Studio 17

W czasie pracy nad aplikacją automatyzującą proces przeprowadzania testów wykorzystano darmowe narzędzie, którego przeznaczeniem jest zarządzanie bazą danych - SQL Server Management Studio 17. Potrzeba użycia narzędzia wynikła z istniejącej infrastruktury aplikacji, która opiera się na rozwiązaniach Microsoftu. Zastosowane narzędzie znacznie ułatwiło pracę i wspomogło proces projektowania bazy danych dzięki funkcjonalności generowania diagramów, które umożliwiły podgląd struktury bazy oraz relacji pomiędzy poszczególnymi tabelami. Kolejną olbrzymią zaletą SQL Management Studio jest możliwość tworzenia zapytań bazodanowych, które pozwalają w bardzo elastyczny sposób pogrupować olbrzymie ilości danych w wybrany przez użytkownika sposób. Umożliwia to w szybkim tempie stworzenie statystyk zastosowanych algorytmów, które dynamicznie będą się aktualizowały wraz z napływem kolejnych danych do bazy.

4.2.3. Visual Studio 2015

4.3. Wykorzystane biblioteki zewnętrzne

4.3.1. Matlab Application Type Library v.1.0

5. Architektura budowanej aplikacji

5.1. Architektura aplikacji

5.1.1. Wzorce architektoniczne oprogramowania

5.1.2. Zastosowany wzorzec architektoniczny – MVVM

5.1.3. Model aplikacji

5.2. Architektura bazy danych

5.2.1. MS-SQL – Zastosowany system zarządzania bazą danych

5.2.2. Budowa bazy danych użytej w projekcie

5.3. Komunikacja bazy danych z projektem programistycznym

5.3.1. Mapowanie obiektowo-relacyjne

5.3.2. Zastosowane narzędzie ORM – Entity Framework

5.4. Komunikacja Matlaba z projektem programistycznym

5.5. Opis okna użytkownika

6. Badania eksperymentalne

6.1. Metody porównawcze wybranych algorytmów

6.2. Opis przeprowadzonych badań

6.3. Wyniki doświadczeń dla zadanych funkcji testowych

6.3.1. Funkcja Bochachevsky'ego

6.3.2. Funkcja Beale'a

6.3.3. Funkcja Rosenbrocka

6.3.4. Funkcja Easoma

6.3.5. Funkcja Eggholdera

6.3.6. Funkcja Griewanka

7. Wnioski

8. Podsumowanie