POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

# KIERUNEK: Elektronika i Telekomunikacja

SPECJALNOŚĆ: Zastosowanie inżynierii komputerowej w technice

## PROJEKT INŻYNIERSKI

Implementacja równoległa wybranego algorytmu globalnej optymalizacji stochastycznej

Parallel implementation of some global stochastic optimization algorithm

AUTOR:

Paweł Sawicz

PROWADZĄCY PRACĘ:

# dr hab. inż. Przemysław Śliwiński

OCENA PRACY:

### WROCŁAW 2014

Spis treści

[KIERUNEK: Elektronika i Telekomunikacja 1](#_Toc405079533)

[PROJEKT INŻYNIERSKI 1](#_Toc405079534)

[dr hab. inż. Przemysław Śliwiński 1](#_Toc405079535)

[WROCŁAW 2014 1](#_Toc405079536)

[**Rozdział 1 Wstęp** 4](#_Toc405079537)

[1.1 Cele Projektu 5](#_Toc405079538)

[**Rozdział 2 Wprowadzenie teoretyczne** 6](#_Toc405079539)

[2.1 Zasady działania algorytmu 7](#_Toc405079540)

[2.2 Random Search 8](#_Toc405079541)

[2.3 Kiefer – Wolfowitz 8](#_Toc405079542)

[2.4 Przegląd innych algorytmów optymalizacji lokalnej 10](#_Toc405079543)

[2.5 Przegląd innych metod optymalizacji globalnej 10](#_Toc405079544)

[2.6 Charakterystyki użytych języków 11](#_Toc405079545)

[2.6.1 Język R 11](#_Toc405079546)

[2.6.2 Język Haskell 11](#_Toc405079547)

[2.7 Programowanie równoległe 12](#_Toc405079548)

[**Rozdział 3 Implementacja algorytmu.** 13](#_Toc405079549)

[3.1 Opis stanowiska oraz środowiska 13](#_Toc405079550)

[**Rozdział 4 Badania** 14](#_Toc405079551)

[4.1 Przebieg oraz badanie implementacji w języku R 15](#_Toc405079552)

[4.2 Przebieg oraz badanie implementacji w języku Haskell 15](#_Toc405079553)

[4.3 Badanie implementacji w języku Haskell zrównoleglone na CPU 15](#_Toc405079554)

[**Rozdział 5 Podsumowanie** 16](#_Toc405079555)

[5.1 Dodatkowe uwagi oraz plany 16](#_Toc405079556)

[**Bibliografia** 16](#_Toc405079557)

[**Spis rysunków** 16](#_Toc405079558)

Podziękowania…

# **Rozdział 1 Wstęp**

Żyjemy w czasach w których generujemy niezmierne ilości danych, głównym czynnikiem jest powszechny dostęp do Internetu. Dodatkowo coraz bardziej wszelakie instytucje udostępniają swoje zbiory danych do zastosowania publicznego. Niegdyś akwizycje i przetwarzanie danych przeprowadzano w dużych przedsiębiorstwach lub na uczelniach, dzisiaj każdy może pobrać dowolne dane z Internetu chociażby zużycie prądu w Wielkiej Brytanii oraz poszukać hrabstwa w którym jest najmniejsze zużycie prądu. Tak narodziła się nowa nauka w świecie programowania która jest dumnie nazywana „Big Data”.

Zespoły „Big Data” zazwyczaj są budowane przez analityków oraz ludzi specjalizujących się w statystyce, optymalizacji. Zadaniem takiego zespołu jest dostarczenie odpowiedzi biznesowych na podstawie posiadanych danych.

Dane mogą być przedstawiać różne informacje np. czas oraz wartość wpłaconych pieniędzy dla fundacji charytatywnej, ile użytkownik zebrał pieniędzy od swoich przyjaciół którzy używają portalu Facebook.com. Gdy jesteśmy w posiadaniu tych wielkich zasobów informacji może spróbować zamodelować matematyczny model dla danego zachowania się rynku, wtedy posiadamy dużo zmiennych które tworzą nam następne wymiary naszej funkcji i nasz problem staje się coraz bardziej trudniejszy do rozwiązania.

Jedną z pomocnych nauk przy pracowaniu i modelowaniu funkcji opartych na „big data” jest na pewno optymalizacja.

Optymalizacja towarzyszyła człowiekowi od bardzo dawna, prawie zawsze człowiek chciał dany problem zminimalizować lub zmaksymalizować produkcje danego dobra. Optymalizacja znajdzie zastosowanie w każdej dziedzinie życia poczynając od medycyny, przetwarzania sygnałów, transportu a kończąc na produkcji ciężkiego przemysłu, możemy zoptymalizować wszystko co można opisać jako model matematyczny.

Istnieje

## 1.1 Cele Projektu

Celem projektu jest implementacje równoległa algorytmu do optymalizacji globalnej, metodą stochastyczna. Następnie zbadanie wydajności kilku implementacji takiego algorytmu oraz wyciągniecie wniosków na temat zrównoleglania takich algorytmów. Algorytm będzie napisany w dwóch językach R oraz Haskell. Wybranym algorytmem będzie połączenie metody Random Search oraz przy wyszukiwaniu minimum lokalnego będziemy używać algorytmu stochastycznej aproksymacja Kiefer – Wolfowitz.

# **Rozdział 2 Wprowadzenie teoretyczne**

Optymalizacja odnosi się do problemu znalezienia maksimum lub minimum zadanej funkcji celu.

Problem ten można opisać w następujący sposób.

Należy znaleźć taki że dla każdego zachodzi następująca nierówność . W przypadku poszukiwania maksimum zmienia się tylko znak nierówności przy funkcji.

Rysunek 2.1 (Przykładowe optimum).

Poszukiwanie ekstremum może odbywać się na pewnej przestrzeni która posiada jedno ekstremum wtedy mówimy o optymalizacji lokalnej. Jednak możemy także poszukiwać ekstremum na funkcji posiadającej wiele ekstremów lokalnych wtedy mówi się o optymalizacji globalnej.

Dodatkowo optymalizacje możemy podzielić na dwie zasadnicze grupy, programowanie liniowe oraz programowanie nieliniowe. W przypadku programowania liniowego funkcja celu jak i funkcje ograniczające są sformułowane w postaci liniowej, wtedy rozwiązanie lokalne jest także optimum globalnym. Natomiast w przypadku programowani nieliniowego funkcja celu jak i funkcje ograniczające nie muszą być funkcjami liniowymi, różniczkowalnymi lub nawet ciągłymi, wtedy nasz funkcja celu posiada wiele ekstremów lokalnych.

Algorytm który został wybrany jest wzorowany na algorytmie który opisał Sid Yakowitz, na potrzeby pracy inżynierskiej zrezygnowaliśmy z etapu uczenia się.

## 2.1 Zasady działania algorytmu

Dane (? The Components w SIAMIE)

– funkcja celu

- przestrzeń globalnej optymalizacji

– podzbiór w czasie

– losowy punkt generowany na etapie RS z całej

– optymalny punkt w czasie

– kandydat

– ilość iteracji

Inicjalizacja

(n – numer iteracji)

dla

Procedura

1. Jeśli , pobierz nowy losowy punkt
   1. Wykonaj K-W przy oraz i wynik przypisz do
   2. Jeśli , wtedy
2. Jeśli zwróć

W następnych punktach są opisane szczegółowo algorytmy Random Search oraz Kiefer – Wolfowitz.

## 2.2 Random Search

Random Search (dalej RS) – Jest to rodzina numerycznych optymalizacji która nie wymaga gradientu do optymalizacji, dodatkowo RS można zastosować przy funkcjach które się są ciągłe oraz różniczkowalne. Ta metoda optymalizacji jest znana również jako bezpośrednie szukanie, wolne od różniczkowania (?).

RS działa na zasadzie iteracyjnego posuwania się na lepszą pozycje w którą możemy próbkować jako .

Niech będzie funkcją celu którą chcemy zoptymalizować. Niech wtedy RS możemy opisać następującym pseudo-kodem.

* Wygeneruj z dziedziny w której chcemy zoptymalizować
* Powtarzaj poniższe kroki dopóki nie będzie spełniony warunek zakończenia poszukiwania, może to być limit iteracji lub błąd bezwzględny.
  + Wygeneruj nowy losowy punkt
  + Jeśli wtedy
* Teraz jest naszym rozwiązaniem

## 2.3 Kiefer – Wolfowitz

Kiefer-Wolfowitz (dalej K-W) – Jest to algorytm z rodziny stochastycznej optymalizacji która wyszukuje ekstrema funkcji których nie można obliczyć bez pośrednio, a jedynie wyestymować poprzez obserwacje szumu.

Niech będzie funkcją która posiada minimum w punkcie , zakładamy że jest nieznane, jednak z pewniej obserwacji . Struktura algorytmu jest podobna do gradientowych algorytmów. Możemy opisać algorytm jako następujący ciąg.

Aktualna implementacja algorytmu w projekcie inżynierskim nieco się różni od podstawowej formy.

Components

Initialization

Procedure

1. Jeśli , wykonuj
   1. Jeśli , wtedy (jak oznaczyć lewa / prawą stronę zbioru ?)
   2. Sss
2. Ssssss

Można zauważyć że parametry cn oraz an różnią się między sobą, w tym przypadku mamy dowolność i sami możemy sobie dobierać te parametry w zależności od wyglądu funkcji.

Parametry te muszą spełniać dwa następujące warunki

W bardzo łatwy sposób możemy zbadać czy nasze parametry zbiegają do zera, z pomocą przychodzi nam twierdzenie o granicy ciągu.

## 2.4 Przegląd innych algorytmów optymalizacji lokalnej

Jak już wcześniej zostało wspomniane nasz algorytm składa się w zasadzie z dwóch innych algorytmów, czyli Random Search oraz Kiefer-Wolfowitz, w takim przypadku nasz algorytm jest dość mocno modularny i możemy dowolnie zmieniać metodę poszukiwania minimum lokalnego.

W tym miejscu należy wymienić pozostałe algorytmy optymalizacji lokalnej.

* Algorytmy optymalizacji w kierunku
* Algorytmy optymalizacji bez ograniczeń
* Algorytmy optymalizacji z ograniczeniami

## 2.5 Przegląd innych metod optymalizacji globalnej

Należy wspomnieć także o innych metodach optymalizacji globalnej oraz powiedzieć kilka zdań na ich temat. Metody te można podzielić na dwie grupy, metody deterministyczne oraz niedeterministyczne, gdzie operacje losowe są istotnym elementem.

* Metody deterministyczne
  + siatki
  + trajektorii cząstki
  + kary
* Metody niedeterministyczne
  + poszukiwań losowych
  + wykorzystujące grupowanie
  + z wykorzystaniem stochastycznego modelu funkcji celu
  + algorytmy ewolucyjne

Z dotychczasowych badań, zostało wykazane iż metody niedeterministyczne są o wiele szybsze od deterministycznych, jednak wymagają dużych zasobów komputera dlatego zalecane jest stosować programowanie równoległe.

## 2.6 Charakterystyki użytych języków

### 2.6.1 Język R

Język ten jest dość mocno abstrakcyjny dzięki czemu można w bardzo szybki oraz przystępny sposób stworzyć prototyp algorytmu. Środowisko posiada bardzo wielką bazę zewnętrznych pakietów (np. sieci neuronowe), jest to pomocne ponieważ można się skupić na implementacji naszego algorytmu.

„R” jest darmowym językiem wydanym na licencji GNU PL, dlatego też jest to jeden z czynników dla czego ten język został wybrany.

### 2.6.2 Język Haskell

Haskell jest z rodzimy języków funkcyjnych, sam haskell jest językiem czysto funkcyjnym. Język ten jest intensywnie rozwijany przy Uniwersytecie Glasgow najbardziej popularnym kompilatorem jest GHC (Glasgow Haskell Compilator).

Najbardziej charakterystyczne cechy tego języka to :

* Leniwe wartościowanie
* Monady
* Statyczny polimorfizm
* Klasy typów (ang. Typeclass)
* Strażnicy (ang. Guards)
* Rozwijanie funkcji (ang. currying) oraz częściowe funkcje (ang. partial functions)

Główną różnicą pomiędzy językiem funkcyjnym a imperatywnym jest formowanie problemu oraz zapis. W języku imperatywnym nasza funkcja posiada kilka kroków do wykonania i może zmieniać swój stan w zależności jakie zmienne przyjmiemy w ciele funkcji, kiedy w języku funkcyjnym nasza funkcja nie może zmieniać swojego stanu w trakcie jej wykonywania, dodatkowo zmienne są „niezmienne” (ang. unmutable) np. kiedy będziemy chcieli każdy element listy pomnożyć przez dwa, wtedy wynik zawsze będzie reprezentowany przez nową listę ponieważ mamy możliwości zmieniać stanów istniejących zmiennych.

Ponadto kompilator GHC wspiera pisanie programów równolegle oraz współbieżne, na potrzeby mojego projektu inżynierskiego algorytm będzie tylko zrównoleglony bez zarządzania wątkami jak to się ma przy programowaniu współbieżnym. (do sprawdzenia!!!!)

## 2.7 Programowanie równoległe

Haskell wspiera programowanie równoległe dla dwóch typów procesorów

* CPU
* GPU

Temat programowania równoległego można traktować jako kontrargument na aktualne limity w częstotliwości taktowania procesora, oraz brak postępu w tym temacie.

//Równoległe – oznacza to że nasz program będzie pracował na wielu procesorach jeśli takowe komputer posiada, zazwyczaj zrównoleglanie odbywa się „nie widocznie” oraz bez większych zmian implementacyjnych.

Współbieżne – cos

# **Rozdział 3 Implementacja algorytmu.**

## 3.1 Opis stanowiska oraz środowiska

Całą pracę wykonałem na laptopie o następującej specyfikacji, procesor - AMD A8-3500M, pamięć ram - 6 GB 1333 MHz, dysk TOSHIBA MK6459GSXP SATA, system operacyjny Windows 8.

Pisząc program w języku R skorzystałem z środowiska o nazwie RStudio które pozwoliło mi na dość szybką implementacje algorytmu oraz pomaga przy zarzadzaniu różnymi bibliotekami.

Przy pisaniu algorytmu w języku Haskell korzystałem z kompilatora GHC oraz programu Cabal do zarządzania zewnętrznymi bibliotekami. Dodatkowo do diagnostyki oraz pomiaru algorytmu w trybie zrównoleglonym używałem narzędzia o nazwie „ThreadScope”, który pozwala obejrzeć ilość tworzonych wątków, czasy procesora czy zużycie jest to dość zaawansowanie narzędzie.

# **Rozdział 4 Badania**

Przy badaniu wydajności algorytmu każda z implementacji została uruchomiona dziesięć razy przy następujących konfiguracjach.

RandomPoints – jest to zmienna która określa nam ile losowych punktów będzie wygenerowanych w E-sieci.

SubsetWidth – wartość która określa nam szerokość podprzedziału.

lGlobalDoman oraz rGlobalDomain – zakres przedziału do przeszukiwania globalnego minimum.

kwSteps – ilość iteracji dla algorytmu Kiefer – Wolfowitz

functionToSolve – wzorcowa funkcja w której poszukujemy minimum globalne.

## 4.1 Przebieg oraz badanie implementacji w języku R

W pierwszej kolejności zdecydowałem się zaimplementować algorytm w języku R, jak już wcześniej wspomniałem chciałem jak najszybciej dostać działająca wersje algorytmu.

Na początku zaimplementowałem Random Search bez poszukiwania lokalnego minimum metoda Kiefer-Wolfowitza w podzbiorach Rd.

Następnie kiedy już metoda RS zostałą poprawnie zaimplementowana zacząłem pracę nad implementacja algorytmu do poszukiwania minimum / maksimum lokalnego. Na tym etapie prac nie napotkałem żadnych większych problemów. Dodatkowo trzeba tutaj nadmienić iż algorytm na razie działa dla funkcji jednej zmiennej.

## 4.2 Przebieg oraz badanie implementacji w języku Haskell

Kiedy skończyłem implementacje algorytmu w R posiadałem już pogląd jak algorytm ma wyglądać oraz jakie wyzwania na mnie czekają przy przepisywaniu na następny język. Od początku nie było łatwo ponieważ wcześniej nie wiedziałem nic na temat Haskella i praktycznie uczyłem się jego na potrzeby pracy inżynierskiej. Największym wyzwaniem była zmiana myślenia oraz pojmowania problemu.

Pierwszym z widocznych rezultatów przepisywania wszystkich algorytmów numerycznych na języki funkcyjne jest prostota zapisu. Implementacja w haskell jest o wiele bardziej czytelna i krótsza.

## 4.3 Badanie implementacji w języku Haskell zrównoleglone na CPU

Przy zrównoleglaniu algorytmu używałem biblioteki „Parallel” która zainstalowałem poprzez program „Cabal”

Kompilator pozwala nam na zrównoleglenie naszego programu na symetrycznych procesorach (WTF?!). Dodatkowo oprócz specjalnej implementacji na etapie kompilacji trzeba zaznaczyć kompilatorowi że chcemy skompilować nasz program w trybie „threaded”, w tym przypadku musimy skompilować program wraz z flagą „-threaded”

# **Rozdział 5 Podsumowanie**

W pracy inżynierskiej zostały wykonane wszystkie zadania które zostały założone.

Program napisany w R rozwiązuje problem optymalizacji najwolniej, najprawdopodobniej dlatego iż jest to język skryptowy bez kompilatora, jak można zauważyć już samo przepisanie algorytmu w na język Haskell bardzo mocno przyspiesza nasz program. Jednak dopiero program zrównoleglony daje najlepsze rezultaty, i w pełni wykorzystuje zasoby komputera, w tym przypadku wszystkie. (Opisać także test na 16 procesorach)

## 5.1 Dodatkowe uwagi oraz plany

asdasdasdas

# **Bibliografia**

aasddasasd

# **Spis rysunków**

asdasdasd