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](#_Toc404901010)

[PROJEKT INŻYNIERSKI 1](#_Toc404901011)

[dr hab. Inż. Przemysław Śliwiński 1](#_Toc404901012)

[WROCŁAW 2014 1](#_Toc404901013)

[**Rozdział 1 Wstęp** 4](#_Toc404901014)

[**1.1 Cele Projektu** 4](#_Toc404901015)

[**Rozdział 2 Wprowadzenie teoretyczne** 5](#_Toc404901016)

[**2.1 Zasady działania algorytmu** 5](#_Toc404901017)

[**2.2 Random Search** 5](#_Toc404901018)

[**2.3 Kiefer – Wolfowitz** 5](#_Toc404901019)

[**2.4 Przegląd innych algorytmów optymalizacji lokalnej** 5](#_Toc404901020)

[**2.5 Przegląd innych metod optymalizacji globalnej** 5](#_Toc404901021)

[**2.6 Charakterystyki użytych języków** 5](#_Toc404901022)

[2.6.1 Język R 5](#_Toc404901023)

[2.6.2 Język Haskell 6](#_Toc404901024)

[**2.7 Pisanie równoległe a współbieżne** 7](#_Toc404901025)

[**Rozdział 3 Implementacja algorytmu.** 7](#_Toc404901026)

[**3.1 Opis stanowiska oraz środowiska** 7](#_Toc404901027)

[**Rozdział 4 Badania** 8](#_Toc404901028)

[**4.1 Przebieg oraz badanie implementacji w języku R** 8](#_Toc404901029)

[**4.2 Przebieg oraz badanie implementacji w języku Haskell** 9](#_Toc404901030)

[**4.3 Badanie implementacji w języku Haskell zrównoleglone na CPU** 9](#_Toc404901031)

[**Rozdział 5 Podsumowanie** 10](#_Toc404901032)

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**

Został użyty algorytm który jest opisany w pracy Sid Yakowitz, w następującej implementacji zrezygnowaliśmy z etapu uczenia się Random Search.

## **2.1 Zasady działania algorytmu**

Algorytm random search można zapisać w postaci następującego pseudo kodu

## **2.2 Random Search**

Random Search (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 przestrzeni poszukiwania którą możemy próbkować jako podprzedział Rd.

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

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

## **2.4 Przegląd innych algorytmów optymalizacji lokalnej**

Jak już wcześniej zostało wspomniane nasz algorytm składa się w zasadzie z dwóch, Random Search oraz Kiefer-Wolfowirtz, w takim przypadku nasz algorytm jest dość mocno modularny i możemy zmieniać metodę do poszukiwać minimum lokalnego na Rd.

W tym wypadku należy wymienić oraz porównać pozostałe algorytmy optymalizacji lokalnej.

## **2.5 Przegląd innych metod optymalizacji globalnej**

Należy wspomnieć także o innych metodach optymalizacji globalnej oraz powiedzieć kilka zdań na ich temat.

## **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 Pisanie równoległe a współbieżne**

* 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**

Asdasd asdasdasdasdads