**Akademia Górniczo-Hutnicza**
**im. Stanisława Staszica w Krakowie**

Wydział Informatyki, Elektroniki i Telekomunikacji

KATEDRA INFORMATYKI


PRACA MAGISTERSKA

MARTA RYŁKO, ANNA SKIBA

# WYKORZYSTANIE VOLUNTEER COMPUTING W ŚRODOWISKU PRZEGLĄDAREK INTERNETOWYCH DO OPTYMALIZACJI TORU PRZEJAZDU W NARCIARSTWIE ALPEJSKIM

PROMOTOR:
dr inż. Roman Dębski

Kraków 2013

## OŚWIADCZENIE AUTORA PRACY

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie, i nie korzystałem ze źródeł innych niż wymienione w pracy.

.....

PODPIS

**AGH**
**University of Science and Technology in Krakow**

Faculty of Computer Science, Electronics and Telecommunication

DEPARTMENT OF COMPUTER SCIENCE


MASTER OF SCIENCE THESIS

MARTA RYŁKO, ANNA SKIBA

# WEB BROWSER-BASED VOLUNTEER COMPUTING FOR SKI-LINE OPTIMISATION IN ALPINE SKI RACING

SUPERVISOR:
Roman Dębski Ph.D

Krakow 2013

Serdecznie dziękuję …

# Spis treści


**1. Wstęp** ..... 7

1.1. Cele pracy ..... 8

1.2. Zawartość pracy ..... 8

**2. Wprowadzenie teoretyczne** ..... 9

2.1. Jazda po zadanym torze w narciarstwie alpejskim ..... 9

2.2. Fizyczny model narciarza ..... 10

2.3. Metody numeryczne rozwiązywania równań różniczkowych ..... 12

2.4. Optymalizacja ..... 12

2.4.1. Algorytm ewolucyjny ..... 12

2.4.2. Hill climbing ..... 16

2.5. Uczenie maszynowe ..... 17

2.5.1. Uczenie się ze wzmocnieniem ..... 17

2.6. Volunteer Computing ..... 20

2.7. Web Workers ..... 21

**3. Istniejące rozwiązania** ..... 23

3.1. Platformy do Volunteer Computing ..... 23

3.1.1. Great Internet Mersenne Prime Searchy ..... 23

3.1.2. Distributed.net ..... 24

3.1.3. Berkeley Open Infrastructure for Network Computing ..... 25

3.1.4. Urządzenia mobilne w rozproszonych obliczeniach naukowych ..... 26

3.1.5. Folding@home ..... 27

**4. Proponowane rozwiązanie** ..... 29

4.1. Model narciarza i środowiska ..... 29

4.2. Opis matematyczny modelu ..... 29

4.3. Numeryczne rozwiązanie problemu ..... 30

4.3.1. Rozwiązanie w 3D ..... 31

4.4. Optymalizacja toru przejazdu ..... 32

4.4.1. Algorytm ewolucyjny ..... 32

4.4.2. Hill climbing ..... 33

4.5. Modelowanie karania ..... 34

4.6. Uczenie maszynowe ..... 34

4.6.1. Uczenie ze wzmocnieniem ..... 34

4.6.2. Q-learning ... 35
4.6.3. Akcja ... 35
4.7. Architektura systemu ... 35
4.7.1. Architektura prototypowa ... 35
4.7.2. Architektura docelowa ... 36
4.7.3. Aplikacja kliencka ... 36
4.7.4. Aplikacja serwerowa ... 37
4.7.5. Efekt końcowy ... 37
**5. Wyniki** ... 42
5.1. Weryfikacja modelu ... 42
5.1.1. Masa i nachylenie stoku ... 42
5.1.2. Czas przejazdu w zależności od toru jazdy ... 46
5.1.3. Jazda po łamanej ... 47
5.2. Optymalizacja ... 48
5.2.1. Algorytm ewolucyjny ... 48
5.2.2. Lokalna optymalizacja ... 62
5.3. Architektura systemu ... 68
5.3.1. Wydajność przeglądarek dla intensywnych obliczeń ... 68
5.3.2. Skuteczność rekrutowania ochotników do obliczeń w przeglądarkowym modelu Volunteer Computing ... 71
5.4. Podsumowanie ... 73
**6. Podsumowanie** ... 75
6.1. Podrozdział ... 75

# 1. Wstęp

W zawodowym sporcie, trening wspomagany jest od dawna komputerami. Symulacje, analiza i porównywanie sekwencji ruchów, modelowanie sylwetki w celu minimalizacji oporów ruchu są metodami, którymi posługują się trenerzy w wielu dyscyplinach sportu. Narciarstwo nie jest wyjątkiem. Istnieją zaawansowane programy pozwalające analizować nagrania wideo z treningów. Nie ma jednak narzędzia, które pozwoliło by znaleźć optymalny, tj.najszybszy poprawny tor przejazdu po zadanym torze.

//tu jeszcze krótko coś o volunteer computing

Narciarstwo alpejskie to dyscyplina z długą historią. Rozwój sportowej wersji narciarstwa alpejskiego rozpoczął się w połowie XIX wieku, jednak nadal nie ma i prawdopodobnie nigdy nie będzie naukowej formuły opisującej tor po jakim należy się poruszać, aby zadaną trasę przejechać najszybciej. Ogromna ilość czynników, które wpływają na czas przejazdu znacznie utrudnia jej znalezienie. W sportowych dyscyplinach narciarstwa alpejskiego celem jest przejechanie w jak najkrótszym czasie wyznaczonej trasy od startu do mety, przejeżdżając przez wszystkie ustawione na trasie bramki - wymuszające skręty.

Problem, jakiego rozwiązania podejmujemy się w pracy, to problem optymalizacyjny rozwiązywany za pomocą symulacji komputerowej. Problem dotyczy znalezienia optymalnego toru przejazdu narciarza po trasie slalomu, który nakłada ograniczenia na ten tor w postaci bramek. Każda bramka ściśle narzuca, z której strony należy ją przejechać, a ominięcie chociaż jednej z nich powoduje dyskwalifikację zawodnika.

Zdefiniowany przez nas problem jest interdyscyplinarny - z pogranicza fizyki i informatyki. Do dobrego zrozumienia zjawisk zachodzących na stoku narciarskim cenne jest też posiadanie własnych doświadczeń z jazdy po trasach slalomu. Wymagania te powodują, że problem nie jest trywialny do rozwiązania i w celu badania go nieodłączne są osoby o różnych kompetencjach.

Obecnie nie udało nam się znaleźć publicznie dostępnych prac, które podchodziłyby do rozwiązania tego praktycznego problemu. Zdajemy sobie sprawę, że problem jest bardzo złożony i próby jego rozwiązania to tak naprawdę rozwiązanie uproszczone tego problemu. Dodatkowo, uwzględnić trzeba fakt, że wiele zmiennych występujących w równaniach wpływa na siebie nawzajem, powodując zmiany niekoniecznie widoczne natychmiast. Może to na przykład sprawiać, że niewielka zmiana dokonana na początku jazdy może mieć znaczący wpływ na ostateczny wynik, co znacznie utrudnia wszelką predykcję na temat wpływu zmian. Aby rozwiązać problem, stworzyłyśmy fizyczny model narciarza - zamodelowany jako punkt materialny o konfigurowalnych parametrach, co umożliwia porównanie wyników np. dla zawodników o różnych masach. Potraktowanie narciarza jako punktu materialnego jest pierwszym z zastosowanych uproszczeń, które zdecydowałyśmy się przyjąć w naszym rozwiązaniu. Stok modelowany jest jako płaszczyzna o zadanym kącie nachylenia, na której za pomocą współrzędnych oznaczamy miejsce występowania bramek. Dużym wyzwaniem było dobranie przybliżenia trasy przejazdu, aby umożliwić wystarczająco łatwe obliczenia i jednocześnie nie tracąc zbytnio na dokładności oddania realnej trasy. Łamana, którą wybrałyśmy jako rozwiązanie spełniające obydwa te wymagania, jest wystarczająco dobrym przybliżeniem jeśli narzucimy na nią dodatkowe ograniczenia jak eliminacja ostrych kątów załamania.

Kluczową częścią naszego rozwiązania jest wykorzystanie algorytmu genetycznego do wybrania pewnego lokalnego optimum trasy, a następnie przeprowadzamy lokalną optymalizację celem wygładzenia znalezionego

rozwiązania. Aby przyspieszyć obliczenia, zastosowałyśmy architekturę opartą o rozproszonych klientów wykonujących obliczenia i raportujących do głównego serwera. Na podstawie zebranych danych od klientów, serwer jest w stanie dostarczyć rozwiązanie szybciej oraz można mieć większą pewność, iż jest ono jeśli nie optymalne, to bardzo bliskie optymalnego. Obliczenia wykonywane są w środowisku przeglądarek internetowych w języku JavaScript.

Otrzymane rozwiązanie może mieć zastosowanie nie tylko w celu znajdowania optymalnej trasy przejazdu po zadanym slalomie. Przykładem może być wsparcie dla trenerów ustawiających takie slalomy w postaci aplikacji podpowiadającej gdzie ustawić kolejną bramkę, aby nie było problemów z jej przejechaniem. Dodatkowo, dokładając moduł wyliczający naprężenia i siły działające na stawy kolanowe, można by zredukować negatywny wpływ niefortunnie ustawionych bramek, powodujących wyjątkowe przeciążenia w kolanach, wykrywając to i przestawiając bramki.

## 1.1. Cele pracy

Celem poniższej pracy jest zapoznanie studentów z systemem LaTeX w zakresie umożliwiającym im samodzielne, profesjonalne złożenie pracy dyplomowej w systemie LaTeX.

## 1.2. Zawartość pracy

W rodziale **??** przedstawiono podstawowe informacje dotyczące struktury dokumentów w LaTeXu. Alvis [**?**] jest językiem

# 2. Wprowadzenie teoretyczne

Zrozumienie istoty przedstawionego w tej pracy rozwiązania u doboru optymalnej trasy narciarza, zarówno pod względem algorytmicznym jak i architektonicznym, wymaga zaznajomienia się z istotnymi pojęciami. W tym rozdziale, przedstawimy i opiszemy te pojęcia.

## 2.1. Jazda po zadanym torze w narciarstwie alpejskim

Sportowa jazda na nartach zjazdowych podzielona jest na kilak dyscyplin. Są to: slalom (SL), slalom gigant (GS), super gigant (GS) oraz zjazd (DH). Elementem wspólnym każdej z nich jest konieczność pokonania trasy, od startu do mety, w jak najkrótszym czasie i przy prawidłowym ominięciu każdej z bramek znajdującej się na trasie przejazdu. Szczegółowe zasady dotyczące parametrów stoku i sprzętu określa regulamin organizacji FIS (Federation International du Ski), z pośród których najbardziej istotnymi są:

- minimalna i maksymalna różnica wzniesień na trasie
- minimalna i maksymalna odległość pomiędzy kolejnymi bramkami
- ilość bramek jaka powinna się znaleźć na trasie - proporcjonalnie do różnicy wzniesień
- wymagana długość narty
- wymagany promień skrętu narty

Parametry te różnią się w zależności od dyscypliny. Najbardziej techniczną dyscypliną jest slalom, nazywany wcześniej slalomem specjalnym. Techniczność tej dyscypliny polega na dużej ilości bramek znajdujących się w niewielkiej odległości od siebie, co wymusza częste skręty. Zawodnicy jeżdżą slalom na nartach o bardzo małym promieniu skrętu, rzędu 11 metrów. Bramka w slalomie składa się z dwóch tyczek tego samego koloru. W slalomie gigancie, odległości między kolejnymi bramkami są większe, co implikuje szybszą jazdę. Bramka w tej dyscyplinie składa się z czterech tyczek tego samego koloru, po dwie na każdy koniec bramki. Dodatkowo każde dwie tyczki z końca bramki połączone są płachtą tego samego koloru co tyczki - czerwoną lub niebieską. Zawodnicy do tej dyscypliny używają nart o promieniu nawet 30 metrów. Kolejne dyscypliny są jeszcze szybsze a promienie nart coraz większe. Bramki zarówno w Super gigancie jak i Zjeździe, są analogiczne do tych gigantowych, różnią się tylko szerokością płacht. Super gigant, to bardziej „wypuszczony”slalom gigant, czyli slalom na którym odległości między bramkami są już bardzo duże, a prędkość proporcjonalnie rośnie. Zjazd to już typowa dyscyplina szybkościowa. Na trasie, zakręty wynikają prawie tylko z konfiguracji terenu. Dodatkowo zdarzają się na trasie elementy ukształtowania terenu na których zawodnicy wybijają się i skaczą po nawet 20 metrów.

Aby poprawnie przejechać przez bramkę na trasie, w każdej z opisywanych wyżej dyscyplin, należy przejechać pomiędzy tyczkami wyznaczającymi tzw. *światło bramki*. Bramki wymuszają skręty, ale nie jest tak, że ustawione

są zawsze rytmicznie tzn. wymuszając skręty raz w prawą, raz w lewą stronę o tym samym promieniu. Najczęściej spotykanym rodzajem bramki jest tzw. *bramka otwarta*. Bramka otwarta, charakteryzuje się tym, że światło bramki znajduje się prostopadle do linii spadku stoku. Poza bramkami otwartymi, bramki mogą być ustawione w tzw. figury slalomowe. Pierwszą z nich jest *przelot* i może występować w każdej z dyscyplin. Przelot polega na ustawieniu dwóch kolejnych bramek w taki sposób, że nie wymuszają skrętu pomiędzy nimi. Przelot stosowany jest w celu adaptacji trasy przejazdu do konfiguracji terenu, np. gdy na trasie naturalnie występuje dłuższy skręt w jedną ze stron lub napędzeniu narciarza. Kolejną figurą, stosowaną już tylko w slalomie, jest *łokieć*. Łokieć to dwie bramki ustawione w bliskiej odległości jedna pod drugą w linii spadku stoku. Są to bramki zamknięte, czyli światło bramki jest zgodne z linią spadku stoku. Figura ta wprowadza zmianę rytmu i również pozwala na dostosowanie trasy do konfiguracji terenu. Ostatnią z figur, również stosowaną tylko w slalomie, jest *wertikal*. Wertikal to bramki ustawione tak samo jak w wypadku łokcia, czyli jedna pod drugą, w linii spadku stoku, z tym, że zamiast dwóch, może być trzy, cztery czy nawet pięć kolejno tak ustawianych bramek.

Opisanie wyżej wymienionych elementów, było istotne by zrozumieć problem znajdowania optymalnej trasy. Najtrudniejsze jest bowiem optymalne przejechanie figur slalomowych. Podczas oglądania trasy, przed zawodami czy też podczas treningów, zawodnicy zwracają dużą uwagę na zapamiętanie występujących po sobie figur i przewidywanie, na podstawie doświadczenia, w jaki sposób należy najechać na daną figurę, by zmieścić się, czy też najszybciej pokonać kolejne, następujące po danej figurze bramki. Poprzez najazd na figurę, rozumiemy moment rozpoczęcia i sposób prowadzenia skrętu przed figurą. Im wcześniej narciarz zrobi najazd, tym szybciej, po poprawnym przejechaniu bramki, będzie mógł zmienić kierunek jazdy do kolejnych tyczek.

## 2.2. Fizyczny model narciarza

#### Siła oporu powietrza

Siła oporu powietrza jest przykładem siły tarcia płynu. Zależność wartości tej siły od prędkości może być bardzo złożona i skomplikowana i tylko w specjalnych przypadkach może być rozwiązana analitycznie. Dla bardzo małych wartości prędkości, dla małych cząstek, siła oporu powietrza jest wprost proporcjonalna do prędkości, a zależność ta może być opisana równaniem:

$$F_d = vb \tag{2.1}$$

v - prędkość narciarza

Dla większych prędkości i większych obiektów, siła oporu powietrza jest w dobrym przybliżeniu proporcjonalna do kwadratu prędkości i zależność tą, można opisać równaniem:

$$F_d = \frac{1}{2} C \rho A v^2 \tag{2.2}$$

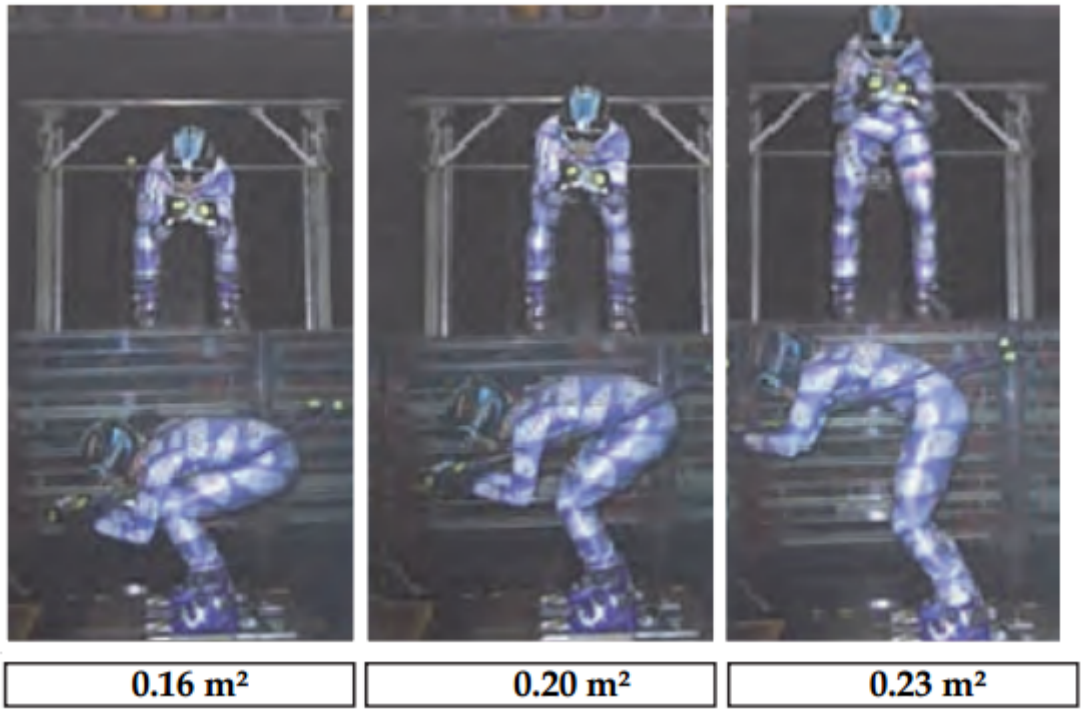
C - współczynnik oporu powietrza, typowe wartości oscylują między 0.4 a 1

$\rho$ - gęstość powietrza w jednostce $kgm^{-3}$. Wartości gęstości powietrza przy przeciętnym ciśnieniu wahają się między 1.26 a 1.42 w zależności od temperatury.

A - frontalna powierzchnia narciarza w projekcji prostopadłej do wektora prędkości narciarza wyrażona w $m^2$.

Na grafice widzimy wartość współczynnika $A$ w zależności od pozycji narciarza. Grafika pochodzi z badań prowadzonych w tunelu aerodynamicznym IAT we Francji, przez Caroline Barelle z National Technical University of Athens z Grecji.

Pozycja narciarza ma znaczący wpływ na wartość współczynnika $A$. Przyjęcie pozycji zjazdowej redukuje w stosunku do pozycji podniesionej, wartość współczynnika nawet o jedną trzecią. Warto nadmienić również, że narciarze, nawet na amatorskich zawodach ubrani są w specjalne kombinezony tzw. *gumy narciarskie*. Kombinezony




te są jednoczęściowe i ściśle przylegają do ciała. Nie posiadają żadnych odstających elementów, czasami zawierają tylko wewnętrzne ochraniacze w miejscach w których narciarz uderza przy każdym skręcie ciałem w tyczkę. Powodem dla którego strój ten jest tak popularny również w amatorskim sporcie jest fakt że znacząco mniejsza opór powietrza, w stosunku do klasycznego stroju narciarskiego i potrafi na kilkudziesięciu sekundowej trasie, gdzie liczy się każda setna sekundy, redukować czas przejazdu nawet o dziesiętne części sekundy.

**Interakcje miedzy śniegiem a nartami**

To, że narty ślizgają się na śniegu zawdzięczamy skomplikowanej fizyce interakcji między powierzchnią narty a śniegiem czy lodem. Ilość czynników jakie wpływają na jakość tej interakcji jest bardzo duża, a z pośród nich warto wymienić:

- materiał wykonania ślizgu narty
- rodzaj, jakość, sposób nakładania i kolejność nakładania smarów na ślizg narty
- gładkość ślizgu narty
- rodzaj i pochodzenie śniegu (naturalne/ sztuczne)
- temperatura i stopień zanieczyszczenia śniegu
- kąt nachylenia między ślizgiem a podłożem

Jest jeszcze jeden czynnik wpływający na tą interakcję, tzw. *water suction* (w wolnym tłumaczeniu zasysanie wody). W temperaturze powyżej -3 stopni Celsjusza, ciepło powstające na skutek tarcia, topi cienką warstwę śniegu pod nartami. Aby zredukować ten negatywnie wpływający na poślizg efekt, ślizg narty ma perforowaną strukturę, którą należy zachowywać i odkrywać po każdym smarowaniu, aby odprowadzać wodę.

Według badań przeprowadzonych przez Chris'a Talbot'a z European Space Agency [**?**], tarcie o śnieg ma dużo większy wpływ na czas przejazdu niż siły tarcia powietrza.

**Tarcie kinetyczne**

Tarcie kinetyczne ma dużo większe znaczenie niż tarcie statyczne ponieważ determinuje jak duża siła musi działać, żeby zachować porządaną prędkość podczas zjazdu. Współczynnik tarcia kinetycznego między śniegiem a nasmarowanymi nartami wynosi średnio 0.05. Współczynnik ten jednak może się znacząco zmieniać i w zależności od rodzaju smarów, sposobu smarowania oraz jakości śniegu wynosi między 0.001 a 0.3. Różnica w wartościach współczynników ma przełożenie w cenach smarów, które na wartości tych współczynników wpływają. Już na amatorskich zawodach, można zaobserwować staranne przygotowanie ślizgów przed każdym zjazdem i używanie smarów których cena wynosi nawet kilkaset złotych, a które są zużywane w ciągu jednego sezonu startów.

## 2.3. Metody numeryczne rozwiązywania równań różniczkowych

## 2.4. Optymalizacja

W tym podrozdziale opisane są metody optymalizacji użyte w zaproponowanym rozwiązaniu, czyli algorytm ewolucyjny oraz algorytm optymalizacji lokalnej - Hill climbing.

Zadaniem optymalizacji jest przeszukanie przestrzeni rozwiązań w celu znalezienia takiego, które jest najlepsze. Zatem mając daną funkcję, nazywaną funkcją celu, która każdemu punktowi reprezentującemu rozwiązanie problemu, poszukujemy takiego, dla którego wartość tej funkcji będzie jak najmniejsza (bądź jak największa). Trudność w znalezieniu takiego rozwiązania zależy od charakteru funkcji celu, a czasem także od nieznajomości jej analitycznej postaci.

**Optymalizacja lokalna i globalna**

W przypadku funkcji z jednym optimum do znalezienia najlepszego rozwiązania wystarczy przeszukiwanie lokalne. Polega ono na iteracyjnym sprawdzaniu rozwiązań w najbliższej przestrzeni i wprowadzaniu lokalnych zmian, aby w końcu znaleźć rozwiązanie najlepsze w okolicy tzw. optimum lokalne. Jeśli wiemy, że istnieje tylko jedno takie optimum, możemy mieć pewność, że znalezione rozwiązanie jest najlepszym w całej przestrzeni rozwiązań. Przykładami optymalizacji lokalnych są:

– hill climbing

– przeszukiwanie tabu

Jeśli natomiast funkcja celu posiada wiele optimów lokalnych (tzw. funkcja wielomodalna) to optymalizację nazywamy optymalizacją globalną. Jeśli zadanie jest ciągłe, a więc niemożliwe jest przeszukanie całej przestrzeni rozwiązań, nigdy nie możemy być pewni, że zastosowany algorytm optymalizacji da nam rozwiązanie najlepsze - być może będzie to tylko minimum lokalne a nie globalne. Nie mając takiej pewności nie wiemy kiedy należy zatrzymać algorytm. Z tego powodu stosuje się parametr sterujący czasem trwania obliczeń, kosztem mniejszej pewności co do poprawności rozwiązania możemy otrzymać krótszy czas optymalizacji i odwrotnie.

### 2.4.1. Algorytm ewolucyjny

Algorytm ewolucyjny jest przykładem algorytmu optymalizacyjnego, przeszukującego przestrzeń rozwiązań w celu znalezienia najlepszego rozwiązania problemu. Algorytm ten oparty jest na obserwacjach środowiska i przystosowywania się organizmów do jego warunków. Wiele terminów zapożyczonych jest zatem z genetyki.

Podstawą całego algorytmu jest populacja osobników, z których każdy reprezentuje rozwiązanie problemu. Populacja ta zmienia się wraz z działaniem algorytmu. Ewolucja zakłada, że populacja będzie się składać z coraz

lepiej przystosowanych osobników. Przystosowanie to jest obliczane za pomocą wcześniej określonej funkcji oceniającej jakość danego osobnika, czyli wyznaczenie jak dobre jest rozwiązanie reprezentowane przez niego. Przystosowanie jest wartością liczbową obliczoną za pomocą tej funkcji przystosowania.
Funkcja przystosowania określa wartość przystosowania osobnika na podstawie jego fenotypu, który jest tworzony z genotypu. Genotyp określa zestaw cech danego osobnika i składa się z chromosomów (najczęściej z jednego). Natomiast każdy z chromosomów składa się z elementarnych jednostek - genów.

#### Schemat działa algorytmu ewolucyjnego

Algorytm ewolucyjny rozpoczyna się poprzez wygenerowanie populacji bazowej oraz obliczenie przystosowania jej osobników. Przeważnie osobniki te generowane są całkowicie losowo, ale można także wprowadzić konkretne osobniki np. o znanym dobrym przystosowaniu do środowiska.

Główna część algorytmu opiera się na powtarzaniu pętli, w której wykonywane są kolejno:

– reprodukcja

– operacje genetyczne

– ocena

– sukcesja

Często reprodukcję i sukcesję łączy się pod nazwą selekcja.

Reprodukcja powoduje powielenie losowo wybranych osobników z populacji. Prawdopodobieństwo wybrania osobnika do powielenia najczęściej jest proporcjonalne do jego przystosowania. Może się zdarzyć, że dany osobnik zostanie wybrany więcej niż raz, a także, że nie zostanie wybrany ani razu.
Następnie na tych kopiach przeprowadzane są operacje genetyczne powodujące zmiany w genotypie osobników. Wyróżniamy dwie podstawowe operacje:

– mutacja

– krzyżowanie

Zadaniem mutacji jest losowe zmodyfikowanie genów w genotypie.
Krzyżowanie, zwane także rekombinacją (ang. *crossover*), działa na co najmniej dwóch osobnikach i na podstawie ich genotypu tworzy jeden lub więcej osobników potomnych. Chromosomy rodzicielskie są mieszane w celu otrzymania nowych genotypów dla osobników potomnych.

W wyniku operacji genetycznych powstają nowe osobniki, które wchodzą w skład populacji potomnej. Każdy z tych osobników jest oceniany za pomocą funkcji przystosowania. Porównując jakość osobników z populacji bazowej oraz potomnej dokonuje się sukcesji, czyli wyboru osobników z tych populacji (czasem wyłącznie z populacji potomnej) i tworzy nową populację bazową.

Zakończenie działania algorytmu przeważnie opiera się na badaniu funkcji przystosowania całej populacji. Jeśli wartość przystosowania populacji nie jest zróżnicowana mówimy o stagnacji algorytmu i może być to wskazaniem do zakończenia działania algorytmu. Czasem jednak oczekuje się aż przystosowanie to będzie wystarczająco duże, żeby stwierdzić, że znalezione rozwiązanie jest bardzo dobre. Przeważnie jednak nie znamy nawet przybliżonej wartości jakości rozwiązania, więc nie możemy stwierdzić kiedy przystosowanie jest odpowiednie i czy nie może się jeszcze znacznie poprawić.

### Kodowanie osobników

W przypadku algorytmów genetycznych, będących szczególnym przypadkiem algorytmów ewolucyjnych, do kodowania osobników stosuje się kodowanie binarne chromosomów. Pojedynczy bit reprezentuje zatem gen należący do chromosomu.
W takim przypadku mutacja wykonywana jest na każdym genie osobno z pewnym prawdopodobieństwem, jeśli do niej dochodzi, zmienia się wartość bitu na przeciwną. W krzyżowaniu wybiera się dwa osobniki rodzicielskie, których chromosomy rozcinane są na dwie części i łączone ńa krzyż". Miejsce przecięcia jest losowane z rozkładem równomiernym.

W algorytmach ewolucyjnych porzuca się kodowanie binarne - chromosom składa się z jednej lub więcej liczb stanowiących cechy osobnika.
Mutacja takiego osobnika najczęściej odbywa się poprzez losową zmianę każdej z wartości genów chromosomu. Do krzyżowania wybiera się dwa osobniki, z których dla każdej pary odpowiadających genów wyciągana jest średnia i tak otrzymane wartości genów tworzą genotyp nowego osobnika.

### Typy algorytmów ewolucyjnych

Algorytmy ewolucyjne wywodzą się z kilku osobnych nurtów zajmujących się tą tematyką, więc istnieje wiele podobnych schematów. Najlepiej traktować algorytmy ewolucyjne jako metaheurystykę - określony jest pewien szkic algorytmu, który można dostosowywać do konkretnego rozwiązania. W tym podrozdziale opisane są podstawowe i najbardziej popularne schematy postępowania oparte o algorytmy ewolucyjne.

**Prosty algorytm genetyczny** Prosty algorytm genetyczny został zaproponowany w roku 1975 przez John'a Holland'a.

---

$t = 0$
$P^0 = createInitPop()$
**while** $stopCondition == false$ **do**
  $T^t = createTempPop(P^t)$
  $T^t = crossPop(T^t)$
  $O^t = mutatePop(T^t)$
  $P^{t+1} = O^t$
  $t = t + 1$
**end while**

---

Mając populację bazową $P^t$ dokonujemy reprodukcji tej populacji, tworząc populację tymczasową $T^t$ składającą się z takiej samej liczby osobników. Wybierani są oni z prawdopodobieństwem proporcjonalnym do ich przystosowania z populacji bazowej. Na populacji tymczasowej dokonujemy operacji genetycznych (mutacji i krzyżowania). Do krzyżowania wybierane są rozłączne pary osobników i z pewnym prawdopodobieństwem $p_c$ zachodzi ich skrzyżowanie. Jeśli doszło do powstania osobników potomnych zastępują one osobniki rodzicielskie. Następnie na tak otrzymanej populacji tymczasowej dochodzi do mutacji osobników i otrzymania populacji potomnej $O^t$. Ta populacja staje się w następnej iteracji algorytmu nową populacją bazową.
Zatrzymanie algorytmu może być dokonane jeśli np.:

- wykonano określoną z góry liczbę iteracji
- znaleziono osobnika o wystarczająco wysokiej wartości przystosowania

W tej wersji algorytmu często pętlę algorytmu nazywa się generacją, a każdą populację $P^t$ w chwili t pokoleniem.

**Strategia (1+1).** Strategia (1+1) jest podstawową strategią ewolucyjną. W algorytmie tym mamy do czynienia z populacją składającą się z tylko jednego osobnika posiadającego jeden chromosom. W każdej pętli algorytmu dokonuje się mutacji tego chromosomu, co powoduje powstanie nowego osobnika. Osobnik ten jest poddawany ocenie, a następnie dokonuje się wyboru lepszego z dwóch istniejących osobników i tego pozostawia w populacji. W mutacji dodaje się do każdego genu chromosomu losową modyfikację rozkładem normalnym:

$$Y_i^t = X_i^t + \sigma\xi_{N(0,1),i} \tag{2.3}$$

Wartość $\sigma$ będzie powodowała większe lub mniejsze zmiany w chromosomie. Jeśli chcemy przeszukać przestrzeń rozwiązań, powinniśmy zwiększać jej wartość, co jest pożądane zwłaszcza w początkowej fazie działania algorytmu. Natomiast, aby znaleźć jak najlepsze rozwiązanie, wiedząc że obecne rozwiązanie jest już bardzo bliskie najlepszemu, możemy zmniejszać wartość $\sigma$ przeszukując tylko najbliższą przestrzeń.
Do wyznaczania $\sigma$ powstał następujący algorytm zwany regułą 1/5 sukcesów:

1. Jeśli przez kolejnych k pętli algorytmu mutacja powoduje powstanie lepszego osobnika w więcej niż 1/5 wszystkich mutacji, to zwiększamy $\sigma$: $\sigma' = c_i\sigma$. Wartość $c_i$ wyznaczona empirycznie wynosi $\frac{1}{0.82}$
2. Gdy dokładnie 1/5 kończy się sukcesem, wartość $\sigma$ pozostaje bez zmian.
3. Jeśli nie zachodzi żadne z powyższych wartość $\sigma$ jest zmniejszana: $\sigma' = c_d\sigma$. Gdzie $c_d$ powinna wynosić $0.82$

**Strategia ($\mu$ + $\lambda$).** Strategia ($\mu$ + $\lambda$) jest rozwinięciem strategii (1+1). $\mu$ oznacza ilość osobników w populacji początkowej, a $\lambda$ ile osobników jest reprodukowanych i poddawanych operacjom genetycznym. Dodatkowo, zamiast reguły 1/5 sukcesów wprowadzono mechanizm samoczynnej adaptacji zasięgu mutacji, a także wprowadzono operator krzyżowania.

Oznaczenie $\mu + \lambda$ oznacza, że po wygenerowaniu populacji potomnej wybierane jest $\mu$ najlepszych osobników do nowej populacji bazowej - zarówno spośród populacji potomnej, jak i starej populacji bazowej zawierających łącznie $\mu$ + $\lambda$ osobników.

---

$t = 0$
$P^0 = createInitPop(\mu)$
**while** $stopCondition == false$ **do**
  $T^t = createTempPop(P^t, \lambda)$
  $T^t = crossPop(T^t)$
  $O^t = mutatePop(T^t)$
  $P^{t+1} = select(O^t \cup P^t, \mu)$
  $t = t + 1$
**end while**

---

W strategii tej ważne jest też kodowanie, do którego dodatkowo dołożono również chromosom przechowujący wektor $\sigma$ zawierający wartości odchyleń standardowych, które wykorzystuje się w trakcie mutacji.
Po wylosowaniu wartości zmiennej losowej o rozkładzie normalnym ($\xi_{N(0,1)}$) dla każdego elementu wektora $\sigma$ losujemy jeszcze jedną zmienną losową o rozkładzie normalnym ($\xi_{N(0,1),i}$) i obliczamy nowe wartości odchyleń z wektora $\sigma$:

$$\sigma_i' = \sigma_i e^{(\tau' \xi_{N(0,1)} + \tau \xi_{N(0,1),i})} \tag{2.4}$$

Gdzie $\tau$ oraz $\tau'$ są parametrami algorytmu, a ich wartości powinny wynosić:

$$\tau = \frac{K}{\sqrt{2n}} \tag{2.5}$$

$$\tau' = \frac{K}{\sqrt{2\sqrt{n}}} \tag{2.6}$$

Mając dane nowe wartości odchyleń standardowych możemy obliczyć nowe wartości genów korzystając ze wzoru:

$$X_i' = X_i + \sigma_i' \xi_{N(0,1),i} \tag{2.7}$$

gdzie $\xi_{N(0,1),i}$ jest nową losową wartością.

Algorytm ewolucyjny wybiera osobniki lepiej przystosowane, a więc te, które posiadają także lepsze wartości odchyleń standardowych. Powoduje to naturalną selekcję, doprowadzającą do samoczynnej adaptacji odchyleń standardowych stosowanych w trakcie mutacji.

Krzyżowanie występuje w tym algorytmie pod nazwą rekombinacja. Najczęściej sprowadza się do uśrednienia lub wymianie wartości wektorów, także wektora $\sigma$.

**Strategia ($\mu$, $\lambda$).** Strategia ($\mu$ + $\lambda$) posiada pewne wady, które postanowiono spróbować wyeliminować za pomocą nowej strategii ($\mu$, $\lambda$). Poprzedni algorytm sprawia problemy jeśli w populacji pojawia się osobnik o wysokiej wartości przystosowania, ale posiadający zbyt duże (albo zbyt małe) wartości odchyleń standardowych. Usunięcie takiego osobnika z populacji często nie jest procesem krótkotrwałym, gdyż wpływa on na powstające potomstwo, przekazując mu podobne do jego, nieodpowiednie wartości odchyleń.
W nowej strategii wprowadzono zmianę, która powoduje, że osobniki rodzicielskie nie są nigdy brane do kolejnej populacji bazowej. Podczas selekcji korzysta się zatem tylko z powstałej populacji potomnej, z niej wybierając osobniki do populacji bazowej w kolejnej iteracji.

---

$t = 0$
$P^0 = createInitPop(\mu)$
**while** $stopCondition == false$ **do**
  $T^t = createTempPop(P^t, \lambda)$
  $T^t = crossPop(T^t)$
  $O^t = mutatePop(T^t)$
  $P^{t+1} = select(O^t, \mu)$
  $t = t + 1$
**end while**

---

### 2.4.2. Hill climbing

Algorytm hill climbing jest jedną z metod przeszukiwania lokalnego. W każdej iteracji zmieniając wartość jednej ze zmiennych rozwiązania sprawdzana jest wartość funkcji celu dla nowego rozwiązania i jeśli wartość ta jest lepsza od dotychczas najlepszej znalezionej, zapamiętujemy zmienione rozwiązanie. Dopóki zmiany powodują poprawę rozwiązania, algorytm nie jest zatrzymywany. Na końcu wiemy, że znalezione rozwiązanie jest rozwiązaniem lokalnie optymalnym.

Przeszukiwanie przestrzeni dyskretnej sprowadza się do sprawdzenia rozwiązań najbliższych obecnemu i wybieranie tego rozwiązania, którego wartość obliczona za pomocą funkcji celu jest najlepsza. Jeśli wśród sąsiadów nie ma już lepszego rozwiązania, możemy zakończyć przeszukiwanie.

```
current = startPoint
foundBetter = true
while foundBetter == true do
  foundBetter = false
  neighbours = getNeighbours(current)
  for all neighbour in neighbours do
    if neighbour.getValue() < current.getValue() then
      current = neighbour
      foundBetter = true
    end if
  end for
end while
```

W przestrzeni ciągłej konieczne jest dobranie kroku, który wyznacza punkty przeszukiwane w okolicy w trakcie każdej iteracji. Dodatkowo wykorzystywane jest tzw. przyspieszenie (ang. *acceleration*), które wyznacza pięciu możliwych kandydatów na lepsze rozwiązania. Najczęściej przyspieszenie to wynosi 1.2, a wartość kroku jest osobna dla każdej zmiennej rozwiązania i często wynosi na początku 1. Zatem za każdym razem obliczane są następujące współczynniki: -acceleration, -1/acceleration, 0, 1/acceleration, acceleration. Następnie współczynniki mnożone są przez krok (step) i dodawane do obecnie analizowanej zmiennej i wybierane jest najlepsze z pięciu rozwiązań. Wartość kroku jest indywidualna dla każdej zmiennej. Po wybraniu najlepszego rozwiązania uaktualniana jest wartość tego kroku - krok mnożony jest przez odpowiedni współczynnik, ten który był dobrany wcześniej do znalezienia tego najlepszego rozwiązania. Algorytm zatrzymywany jest jeśli zmiana żadnej ze zmiennych nie przynosi już poprawy rozwiązania, czasem również jeśli ta zmiana jest już bardzo mała - wprowadzany jest parametr $\epsilon$ wyznaczający tę różnicę.

## 2.5. Uczenie maszynowe

Uczeniem się systemu jest każda autonomiczna zmiana w systemie zachodząca na podstawie doświadczeń, która prowadzi do poprawy jakości jego działania. (Cichosz)

Program się uczy z doświadczenia E dla zadań T i miary jakości P jeśli jego efektywność w zadaniach z T mierzona P wzrasta z doświadczeniem E. (Mitchell)

Istnieje wiele rodzajów uczenia maszynowego. Podstawowy podział wynika z rodzaju informacji trenującej na:

– uczenie z nadzorem

– uczenie bez nadzoru

W uczeniu się z nadzorem źródłem informacji trenującej jest nauczyciel. Od niego otrzymuje uczeń informację jakie zachowanie jest pożądane. Natomiast w przypadku uczenia bez nadzoru uczeń dowiaduje się o skuteczności swojego działania obserwując wyniki - nazywa się to czasem wbudowanym nauczycielem.

Istnieją jeszcze dwie grupy, które trudno zakwalifikować do powyższych:

$currentResult = startPoint$
$steps = initialSteps$ {for each dimension of the solution}
$candidates = [-acc, -\frac{1}{acc}, 0, \frac{1}{acc}, acc]$
$currentValue = currentResult.getValue()$
$beforeValue = MAX\_VALUE$
$\epsilon = EPSILON$
**while** $beforeValue - currentValue > \epsilon$ **do**
  $beforeValue = currentValue$
  **for** $i$ *in* $dimensions$ **do**
    $bestIndex = -1$
    $bestScore = MAX\_VALUE$
    **for** $j$ *in* $candidatesNrs$ **do**
      $currentResult[i] = currentResult[i] + stepSize[i] * candidates[j]$
      $tempValue = currentResult.getValue()$
      $currentResult[i] = currentResult[i] - stepSize[i] * candidates[j]$
      **if** $tempValue < bestScore$ **then**
        $bestScore = tempValue$
        $bestIndex = j$
      **end if**
    **end for**
    **if** $candidates[bestIndex]! = 0$ **then**
      $currentResult[i] = currentResult[i] + stepSize[i] * candidates[bestIndex]$
      $stepSize[i] = stepSize[i] * candidates[bestIndex]$ {accelerate}
    **end if**
  **end for**
  $currentValue = bestScore$
**end while**

– uczenie się na podstawie zapytań

– uczenie się przez eksperymentowanie

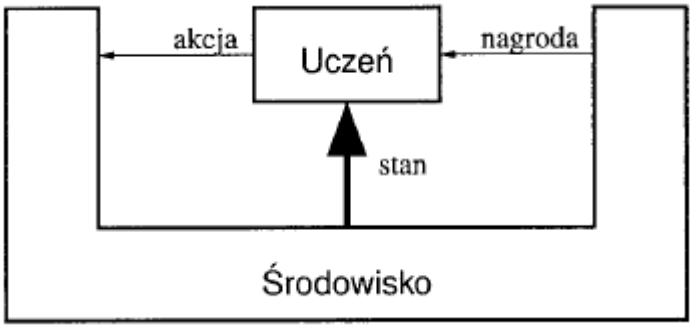
– uczenie się ze wzmocnieniem

Do pierwszej z nich należą algorytmy polegające na zadawaniu pytań przez ucznia nauczycielowi. Natomiast do drugiej te, w których uczeń gromadzi swoje doświadczenia obserwując konsekwencje swojego działania w środowisku. Uczenie się ze wzmocnieniem jest podobne do tej metody, ale dodatkowo istnieje krytyk, który służy jako dodatkowe źródło informacji trenującej. Jego zadaniem jest karanie bądź nagradzanie ucznia za jego zachowanie. Uczeń nie dowiaduje się co ma robić, ale jak wartościowe jest dane działanie.

Czasem granice pomiędzy tymi grupami są nieostre i przynależność algorytmu do jakiejś grupy może zależeć wyłącznie od punktu widzenia.

### 2.5.1. Uczenie się ze wzmocnieniem

W przypadku uczenia się ze wzmocnieniem zadaniem ucznia jest obserwacja stanów środowiska, wykonywanie akcji oraz obserwowanie efektów tych akcji poprzez wartość otrzymywanego wzmocnienia jako rzeczywistoliczbowej nagrody. Tak jak zostało to napisane wcześniej, w tym przypadku nie mówimy o nauczycielu, ale o krytyku, który wartościuje zachowanie poprzez dostarczanie wzmocnienia. Zadaniem ucznie jest odnalezienie takiego zachowania, które przyniesie mu jak największą nagrodę. Najczęściej uczeń nie ma pojęcia o tym jakie jest środowisko, często niedeterministyczne, dlatego musi wchodzić w interakcję z nim, aby je poznać.




W każdym kroku uczeń jest w określonym stanie środowiska. Decydując się na określoną akcję otrzymuje informację o nowym stanie, w którym znajduje się po wykonaniu tej akcji oraz o nagrodzie (wzmocnieniu) jaką otrzymuje za swoje działanie. Uczeń obserwując nagrody otrzymywane za swoje zachowanie może uczyć się jak postępować, aby były one jak najwyższe.

Schemat algorytmu przedstawia się następująco:
Dla kolejnych kroków czasowych t:

1. obserwujemy stan $x_t$
2. wybieramy akcję $a_t$ możliwą do wykonania w stanie $x_t$
3. wykonujemy akcję $a_t$
4. obserwujemy wzmocnienie $r_t$ i następny stan $x_{t+1}$
5. uczymy się na podstawie doświadczenia $(x_t, a_t, r_t, x_{t+1})$

Wybór akcji w kroku 2. dokonywany jest autonomicznie przez ucznia. Natomiast stan, do którego przechodzi po wykonaniu akcji jest określony przez środowisko na podstawie stanu poprzedniego oraz wykonanej akcji. Warto jednak zwrócić uwagę na fakt, że środowisko może być stochastyczne - wykonanie dwa razy tej samej akcji może dawać różne rezultaty. Poza tym, przeważnie środowisko jest nieznane uczniowi, stąd konieczność podejmowania prób i błędów poprzez wykonywanie różnych akcji. Jednocześnie, uczeń nie może wpływać na środowisko w żaden sposób.

#### Strategia maksymalizacji nagród

Nauka ucznia oparta jest na nagrodach, które otrzymuje za swoje działania. Musi znaleźć on najlepszą strategię wyboru akcji, aby uzyskiwać jak najlepsze nagrody. Najczęściej uczeń próbuje maksymalizować swoje nagrody długoterminowo. Strategia ta polega na tym, że nagrody za poprawne działanie mogą przyjść wiele kroków później niż wtedy gdy ono zostało wykonane. Strategia ta nazywana jest uczeniem się z opóźnionym wzmocnieniem. W uczeniu z natychmiastowym wzmocnieniem interesuje nas tylko maksymalizacja nagród tuż po danym zachowaniu. Nie jesteśmy wtedy w stanie brać pod uwagę tego, jakie w przyszłości mogą być jego skutki.
W przypadku opóźnionego wzmocnienia wprowadza się współczynnik dyskontowania $\gamma \in [0, 1]$. Zadaniem ucznia jest zmaksymalizowanie zdyskontowanej sumy nagród:

$$E[\sum_{t=0}^{\infty} \gamma^t r_t] \tag{2.8}$$

Im współczynnik $\gamma$ jest bliższy 0, tym bardziej maksymalizuje się tylko natychmiastowe nagrody. Jeśli $\gamma = 1$ to maksymalizowana jest suma wszystkich otrzymanych nagród.

#### Zadania epizodyczne

W niektórych przypadkach działania ucznia można łatwo wydzielić na niezależne epizody, z których każdy trwa najczęściej skończoną liczbę kroków. Rozdział ten jest kierowany tym, że każda z tych prób jest oceniania osobno. Zatem maksymalizujemy kryterium jakości w każdej próbie oddzielnie. Zatem w równaniu 2.8 musimy zastąpić sumę nieskończoną otrzymując:

$$E[\sum_{t=0}^{n-1} \gamma^t r_t] \tag{2.9}$$

Wartość $n$ to liczba kroków epizodu. Powyższe równanie można traktować tak naprawdę jako szczególny przypadek równania 2.8. Zakładając, że w ostatnim kroku wchodzimy do stanu, w którym jedyna możliwa akcja prowadzi z powrotem do niego, a nagroda w tym stanie wynosi 0, otrzymujemy dokładnie powyższe równanie. W równaniu tym zmienna $r_t$ powinna być dla uściślenia zastąpiona przez $r_{i,t}$, ponieważ wartości wzmocnienia mogą być różne w każdej próbie $i$.

Istnieją także dwa szczególnego rodzaju typy zadań epizodycznych, które nazywane są zadaniami do-sukcesu lub do-porażki. Zakończenie każdej próby kończy się odpowiednio w każdym typie sukcesem lub porażką. W przypadku zadań do-sukcesu chcemy, aby w każdym epizodzie w jak najmniejszej liczbie kroków osiągnąć pewien pożądany stan, co powoduje osiągnięcie sukcesu i zakończenie tej próby. Odpowiednio dla zadań do-porażki jak najbardziej chcemy odwlec moment przejście do niepożądanego stanu, który oznacza porażkę.

#### Algorytm uczenia się strategii

**Q-learning.** Algorytm Q-learning jest najczęściej stosowanym algorytmem do uczenia się optymalnej strategii. Uczenie się polega na oszacowaniu optymalnej funkcji wartości akcji. W każdy kroku czasowym obliczana jest wartość następującego wyrażenia, które nazywane jest błędem:

$$\Delta = r_t + \gamma \max_a Q_t(x_{t+1}, a) - Q_t(x_t, a_t); \tag{2.10}$$

$r_t$ to wartość wzmocnienia w tym kroku czasowym, następny człon określa maksymalną wartość funkcji $Q$ dla stanu, w którym znajdzie się uczeń po wykonaniu wybranej wcześniej przez siebie akcji. Tę wartość mnożymy przez współczynnik dyskontowania $\gamma$ i od tak obliczonej liczby odejmujemy obecną wartość funkcji dla obecnego stanu i wybranej akcji.
Aktualizacja wartości funkcji odbywa się w następujący sposób:

$$Q_{t+1}(x_t, a_t) = Q_t(x_t, a_t) + \alpha\Delta \tag{2.11}$$

Dwa warunki dotyczące wartości $\alpha$ powinny być spełnione:

$$\sum_{i=1}^{\infty} \frac{1}{\alpha_i(x, a)} = \infty \tag{2.12}$$

$$\sum_{i=1}^{\infty} \frac{1}{\alpha_i^2(x, a)} < \infty \tag{2.13}$$

Jednak najczęściej w praktyce rzadko stosuje się do tych warunków.

Warto zwrócić uwagę na fakt, że nie określono tu w jaki sposób uczeń rzeczywiście wybiera akcję. Nie musi to być wcale strategia, w której zawsze wybierana jest najlepsza z akcji.

**Algorytm Sarsa.** Algorytm Sarsa niewiele różni się od algorytmu Q-learning. Jedyną różnicą jest modyfikacja wyrażenia na błąd:

$$\Delta = r_t + \gamma Q_t(x_{t+1}, a_{t+1}) - Q_t(x_t, a_t); \tag{2.14}$$

Zamiast wykorzystywać maksymalną wartość funkcji w przyszłym stanie, korzystamy z wartości, która faktycznie zostanie wybrana. Ponieważ wybór ten dokonywany jest dopiero w następnym kroku, ogólny schemat musi być lekko zmodyfikowany. Różnica ta powoduje, że algorytm Sarsa nie posiada własności algorytmu Q-learning dotyczącego wyboru akcji. Maksymalizacja dokonywana jest z użyciem tej samej strategii, z której korzysta algorytm.

### Wybór akcji

Tak jak w przypadku algorytmów ewolucyjnych konieczna jest eksploracja środowiska, w którym znajduje się uczeń. Z drugiej jednak strony chcemy jak najszybciej znaleźć optymalne zachowanie, czyli dokonać eksploatacji wiedzy, którą już posiadamy. Pojawia się pytanie w jaki sposób wybierać akcję, aby dobrze poznać środowisko, a jednocześnie nie zmarnować zbyt dużo czasu na bezcelowe przeszukiwanie opcji, które nie dają korzystnego rozwiązania.
W przypadku algorytmów takich jak Q-learning, w których można użyć innej strategii wyboru akcji niż ta, która jest używana w trakcie maksymalizacji funkcji $Q$, zapewnienie eksploracji można zapewnić stosując jednostajnie losowy wybór akcji.

Najprostszym sposobem wyboru akcji jest wybranie najlepszej z możliwych akcji, nazywane strategią zachłanną, jednak metoda ta uniemożliwia eksplorację. Aby to umożliwić, wprowadza się modyfikację, która polega na tym, że co jakiś czas z prawdopodobieństwem $\epsilon$ stosuje się wspomniany wcześniej jednostajnie losowy wybór akcji. Strategia ta nazywana jest strategią $\epsilon$-zachłanną. Metoda ta jest znana jako dobrze równoważąca zadania eksploracji i eksploatacji.

Minusem strategii $\epsilon$-zachłannej jest to, że w trakcie eksploracji każda z akcji jest losowana z takim samym prawdopodobieństwem. Aby wyeliminować ten element, stosuje się metody selekcji nazywane *softmax*. Wszystkie akcje mogą zostać wylosowane, ale z prawdopodobieństwem odpowiadającym im dotychczasowej wartości. Najczęściej korzysta się z rozkładu Gibbs'a lub Boltzmann'a. Prawdopodobieństwo wybrania akcji $a$ w próbie $t$ wynosi:

$$\frac{\epsilon^{Q_t(a)/\tau}}{\sum_{b=1}^{n} \epsilon^{Q_t(b)/\tau'}} \tag{2.15}$$

Parameter $\tau$ nazywany jest temperaturą. Im jest ona wyższa, tym wybór dowolnej akcji jest bardziej losowy. W przypadku temperatur bliskich 0 wybór praktycznie jednoznacznie padnie na akcję o najwyższej wartości funkcji $Q$.

## 2.6. Volunteer Computing

Volunteer computing to nieformalny kontrakt w którym zwykli ludzie czy też organizacje, nazywani dalej ochotnikami, dobrowolnie udostępniają swoje zasoby obliczeniowe by uruchamiać na nich obliczenia związane z różnorakimi projektami. Projekty to, przeważnie projekty naukowe, których celem jest rozwiązanie problemów i zadań matematycznych czy też problemów dotykających ludzkość, lub dążących do lepszego poznania świata i wszechświata. Dzięki platformą umożliwiającym Volunteer computing, każdy człowiek może w niewielkim stopniu kontrybuować w rozwiązywaniu tych problemów.

Pprocesory w komputerach osobistych spędzają około 80 procent czasu nie robiąc nic. Równocześnie wiele naukowych problemów związanych na przykład z modelowanie zmian klimatu potrzebuje ogromnych zasobów obliczeniowych. Używanie do tego celu superkomputerów jest bardzo drogie. Połączenie tych faktów doprowadziło do stworzenia konceptu by używać do tych obliczeń zasobów zwykłych ludzi, którzy chcą się świadomie nimi dzielić. Obecne doświadczenia wskazują, że wydajność systemu dla pojedynczego prowadzonego projektu opartego o volunteer computing jest porównywalna do tego, gdyby projekt prowadzony był przy wsparciu jednego z topowych superkomputerów z listy TOP500.

Ochotnicy to osoby prywatne albo instytucje takie jak szkoły czy uniwersytety. Ochotnicy przeważnie pozostają anonimowi, choć w niektórych projektach wymagane jest dostarczenie podstawowych danych kontaktowych jak np. adresu email. W wypadku celowego dostarczania błędnych wyników przez ochotnika, utrudnione jest jego dyscyplinowanie czy też wyłączenie z projektu. Ochotnicy nie są wynagradzani finansowo za uczestnictwo w projekcie.

Organizacja czy osoba chcąca wykorzystać model Volunteer computing do swoich projektów, musi być jednostką zaufaną dla ochotników realizujących obliczenia. Wynika to z prostego faktu, że ochotnicy decydują się, według standardowego modelu computing, na zainstalowanie aplikacji dostarczanej przez dawcę zadań obliczeniowych. Osoba instalująca aplikację musi ufać, że nie uszkodzi ona jej komputera ani też nie będzie wykorzystywać jej zasobów w sposób niezgodny z zapewnieniami zleceniodawcy obliczeń. Zleceniobiorca obliczeń ma też prawo oczekiwać, że aplikacja, została napisana przestrzegając dobrych praktyk bezpieczeństwa, gdyż jako, że aplikacja ta łączy się z internetem i potencjalnie jest zainstalowana na dużej ilości maszyn więc jest atrakcyjnym celem ataków zmierzających do przejęcia tych maszyn do niezgodnych z prawem celów przez hakerów.

Przeważnie model komunikacyjny systemu Volunteer Computing uwzględnia tylko komunikacje poszczególnych klientów z centralnym serwerem i nie zakłada bezpośredniej komunikacji między klientami.

Volunteer Computing pierwotnie zakładał, że obliczenia są wykonywane na zwykłych PC-tach. Ilość komputerów tego typu jest nieporównywalnie większa do ilości wyspecjalizowanych komputerów o dużej mocy ob-

liczeniowej i jest szacowana na ponad miliard. Dodatkowa, z przyczyn ekonomicznych, na rozwój tych maszyn producenci sprzętu przeznaczają największe fundusze więc ich moc i zdolności obliczeniowe stale rosną.

Ważnym aspektem, który istotnie wpływa na stosowanie modelu w praktyce jest koszt prowadzenia obliczeń. Model zakłada, że dołączanie się do obliczeń jest ochotnicze i nie dostaje się za uczestnictwo w projekcie wynagrodzenia. Dzięki temu, projekty, które mają poparcie i akceptację społeczną mogą liczyć na darmowe moce obliczeniowe udostępnione przez zwykłych ludzi.

Na ten model można patrzyć także w kategoriach edukacyjnych. Podczas gdy ochotnik przystępuje do projektu i udostępnia swoje moce obliczeniowe, można wykorzystać jego potencjalne zainteresowanie rozwiązywanym problemem i za pomocą przystępnych wizualizacji przedstawić mu sedno rozwiązywanego zadania, nakreślić mu problem z różnych perspektyw i pokazać mu do czego potencjalnie zmierzają obliczenia. Połączenie atrakcyjnej formy tłumaczenia rozwiązywanych problemów z potencjałem portali społecznościowych i wiralności ciekawego materiału można uzyskać daleko idący efekt propagacji i podłączaniu się do obliczeń coraz większej ilości osób.

## 2.7. Web Workers

Przeprowadzanie intensywnych obliczeń w przeglądarkach internetowych nie było możliwe do czasu wprowadzenia przez grupę WHATWG (Web Hypertext Application Technology Working Group) specyfikacji Web Worker. Ograniczenie wynikało z faktu, że język w którym wykonywane są skrypty poprzez silniki przeglądarki to Java Script. Java Script to środowisko jednowątkowe, więc nic nie może być wykonywane równolegle. Zlecając więc skryptowi intensywne obliczenia, na ich czas cały UI strony byłby nieresponsywny, co jest nie do przyjęcia dla człowieka obsługującego stronę internetową. Przeglądarki bronią użytkownika przed takim zachowaniem skryptów na stronie i czasami zdarza się jeszcze zobaczyć okno z ostrzeżeniem, że skrypt przestał odpowiadać i możliwością manualnego zatrzymania skryptu.

Web Workers definiuje API do tworzenia osobnych procesów w tle. Workery wykorzystują do komunikacji z wątkiem głównym klasyczny model przekazywania wiadomości. Nowoczesne przeglądarki umożliwiają przekazywanie zarówno tekstu jak i obiektów zserializowanych jako JSONy. Należy zwrócić uwagę, że obiekty te nie są współdzielone ale w pełni kopiowane.

Web Workery nie ma ją dostępu do struktury DOM, obiektu *window* ani *document*. Zewnętrzne skrypty wykorzystywane przez workera muszą być serwowane z tej samej domeny co kod workera.

Według specyfikacji, stworzonej przez WHATWG, Web Workery powinny być używane do zadań trwających dłuższy czas, mających duży narzut startowy i spory narzut pamięciowy. Nie są więc odpowiednie tworzenie bardzo wielu workerów zajmujących się obliczeniami trwającymi marginalny czas, gdyż sam narzut na stworzenie przez przeglądarkę osobnego procesu może być zbyt duży by uzasadnić jego użycie.

# 3. Istniejące rozwiązania

W rozdziale tym przedstawimy istniejące rozwiązania zarówno architektoniczne, które umożliwiają rozproszone obliczenia w modelu Volunteer Computing, jak i rozwiązania dotykające problemu związanego z poszukiwaniem optymalnej trasy narciarza na slalomie.

## 3.1. Platformy do Volunteer Computing

W klasycznym modelu, architektura umożliwiająca prowadzenie obliczeń w modelu Volunteer Computing składała się z aplikacji klienckich, które muszą być pobrane oraz zainstalowane na komputerze typu PC, oraz serwera, który zarządza wysyłaniem zadań i odbieraniem rozwiązań.

Konieczność pobierania i instalowania aplikacji na urządzeniu implikuje wymóg tworzenia i utrzymywania wersji aplikacji pod każdy znaczący system operacyjny. Proces wyboru odpowiedniej kompilacji programu klienckiego oraz jego instalacji jest pierwszą barierą jaką napotyka przeciętna osoba chcąca kontrybuować do naukowego projektu.

Zleceniodawcy obliczeń, zauważyli, że na świecie popularność zdobywają inne od komputerów osobistych urządzenia, które również dysponują nie wykorzystywaną mocą obliczeniową. Konsola Play Station 3 była pierwszym urządzeniem tego typu, włączonych do projektów wykorzystujących Volunteer Computing. W sierpniu 2013 udostępniono natomiast aplikację dzięki do naukowych projektów można kontrybuować poświęcając zasoby obliczeniowe swojego urządzenia z systemem Android.

W ostatnich latach, można zaobserwować trend przenoszenia softwaru, który dotychczas zainstalowany był na komputerach jako natywna aplikacja, w model SAAS (Software As A Service). Programy pocztowe, funkcjonalne systemy CRM, czy odtwarzacze muzyczne coraz częściej dostępne są za pośrednictwem przeglądarek internetowych. Taka zmiana możliwa była dzięki rozwojowi silników Java Script we współczesnych przeglądarkach, oraz implementacja w przeglądarkach zaawansowanych elementów ze specyfikacji HTML5 oraz Web Workers.

W tym rozdziale, zaprezentujemy przekrój obecnie najbardziej znaczących projektów korzystających z modelu Voluneteer Computing, oraz platform które te projekty wykorzystują.

### 3.1.1. Great Internet Mersenne Prime Searchy

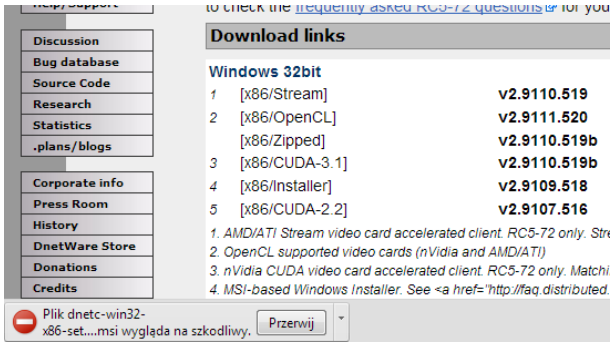
Pierwszym projektem wykorzystującym Volunteer Computing jest rozpoczęty na początku 1996 roku a trwający do dziś *GIMPS* (Great Internet Mersenne Prime Search), którego celem jest znalezienie jak największej ilości specyficznych liczb pierwszych - tzn. Liczb Pierwszych Marsenne'a. Do tej pory kolaboracyjne wysiłki doprowadziły do odnalezienia 14 takich liczb. Moc systemu dała by teoretycznie, wg. danych na listopad 2012 roku - 330-ste miejsce w rankingu TOP500 - rankingu najmocniejszych komputerów na świecie.

System składa się z aplikacji klienckich stworzonych dla 9 platform i konfiguracji systemów. Twórcy zapewniają, że kod programu jest wysoce zoptymalizowanym kodem asemblerowym Intela. Po uruchomieniu, program nawiązuje kontakt z serwerem nazwanym *PrimeNet* aby otrzymać część pracy.

*GIMPS* jest ciekawym projektem, który ma swoje sukcesy i skale, co potwierdzają publikowane regularnie informacje o nowych znaleziskach oraz wielkości sieci komputerów biorących udział w obliczeniach. Interfejs i User Experience które oferuje platforma jest jednak bardzo słaby i archaiczny i zupełnie nie przystosowany do użytkowników współczesnego internetu.

### 3.1.2. Distributed.net

Kolejną znaczącą platformą do Volunteer Computing jest stworzony w 1997 roku *distributed.net*. Platforma to powstała oryginalnie w celu brania udziału w konkursach organizowanych przez RSA Laboratory, tzw. RSA Secret-Key Challenge. Konkursy te polegały na uhonorowaniu pierwszej osoby, która znalazła klucz użyty do szyfrowania tekstu oraz odszyfrowała za jego pomocą tekst zaszyfrowany jednym ze znanych algorytmów szyfrujących. Konkursy miały na celu zademonstrowanie stopnia bezpieczeństwa algorytmu szyfrującego używającego kluczy o różnej długości. Nagrodą było 10000 dolarów. Sieć ochotników podpiętych do systemu *distributed.net* złamała 56 bitowy klucz użyty przez szyfr blokowy RC5 do zaszyfrowania zdania: "The unknown message is: It's time to move to a longer key length". Metodą przyjętą do złamania tego szyfru był najprostszy *brute-force*, która to dzięki metodologi Volunteer Computing przyniosła pozytywne skutki po 250 dniach poszukiwań całej przestrzeni rozwiązań.

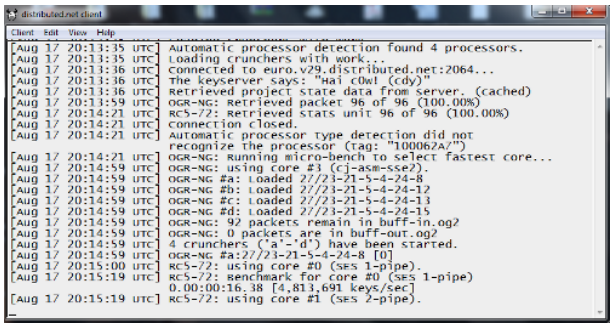



Rysunek 3.1: Pobranie aplikacji klienckiej powoduje wygenerowanie ostrzeżenia dotyczącego bezpieczeństwa.

Program *distributed.net* trwa dalej mimo, że RSA Laboratory wycofało się już z fundowania nagród za łamanie kolejnych coraz to dłuższych kluczy. Osoba chcąca dołączyć do ochotników nie ma jednak bardzo łatwego zadania i nie dołączanie do programu nie należy do przyjemnych, Należy wejść na stronę *distributed.net* i wybrać odpowiedni dla swojego systemu operacyjnego program kliencki. Wybór jest spory i może przysporzyć problemy mniej obeznanym w komputerach osobą gdyż wyróżnione są nie tylko nazwy systemów ale i architektury (np. Windows x86/CUDA-2.2, x86/CUDA-3.1). Po zdecydowaniu się już na któryś z programów i po zakończeniu pobierania, może dostać groźnie wyglądający komunikat w którym jego system operacyjny ostrzeże go że program jest potencjalnie szkodliwy dla komputera, jak przedstawiono na rycinie 3.1 na stronie 24.

Ufając dostawcy, po zdecydowaniu się na zainstalowanie programu, otwiera się bardzo mało przyjazny interfejs konsolowy, który widać na rycinie 3.2 na stronie 25.

Nasz wniosek jest taki, że przestarzałe i nieatrakcyjne systemy do Volunteer Computing nie mają racji bytu w dzisiejszym świecie i prędzej czy później wyginą na rzecz bardziej przyjaznych rozwiązań.




Rysunek 3.2: Mało przyjazny interfejs konsolowy programu *distributed.net*

### 3.1.3. Berkeley Open Infrastructure for Network Computing

Do stworzenia platformy BOINC (Berkeley Open Infrastructure for Network Computing), obecnie najbardziej znaczącego open-sourcowego middlewaru do obliczeń typu volunteer i grid computing, przyczynił się projekt SETI@home.

SETI@home to program rozpoczęty w maju 1999 roku na uniwersytecie Berkeley, mający na celu analizę kosmicznego szumu radiowego i odnalezienie w nim sygnałów pochodzących od pozaziemskich cywilizacji. Przed migracją do BOINC, projekt obsługiwał system posiadający wiele luk bezpieczeństwa. Kwestie, które na początku nie było nacisku czyli wiarygodność zwracanych przez 'wolontariuszy' wyników, wraz z rozwojem projektu zaczęły odgrywać coraz większe znaczenie, gdyż zauważono, że niektóre rozwiązanie były w całości fałszowane.

Obecnie, platforma BOINC umożliwia weryfikację wyników. Opiera się ona przede wszystkim wysyłaniu tych samych zadań do wielu klientów i porównywanie wyników. W programie jest obecnie ponad 40 projektów. Instalując klienta na swojej maszynie można kontrybuować do wybranych przez siebie projektów.

Serwer BOINC może być uruchomiony na jednej lub wielu maszynach z systemem operacyjnym Linux, a oparty jest na technologiach Apache, PHP i MySQL.

Struktura klienta składa się z

– programu *boinc*, który zajmuje się komunikacją z serwerem oraz zarządzaniem aplikacjami naukowymi, które wykonują już konkretne obliczenia.

– jednej lub wielu aplikacji naukowych. Aplikacja naukowa związana jest z konkretnym projektem w którym klient bierze udział i zajmuje się obliczeniami dla dostarczonych jej części obliczeń. Uaktualnianiem aplikacji zajmuje się *boinc*.

– programu *boincmgr* - GUI, które komunikuje się z *boinc* przy użyciu zdalnego wywołania procedur. GUI napisane jest w cross-platformowym toolkicie *WxWidgets*. Poprzez GUI , użytkownik może wybrać nowe aplikacje naukowe, monitorować postęp prowadzonych przez siebie obliczeń oraz przeglądać logi systemu BOINC.

– BOINC'owego wygaszacza ekranu. Jest to framework dzięki któremu aplikacje naukowe mogą wyświetlać w atrakcyjnie graficzny sposób np. animacje czy też wykresy wyjaśniające prowadzone obliczenia. wykresy

– wymagany promień skrętu narty

Obecnie prawie trzy miliony ludzi udostępnia zasoby swoich PC-tów do obliczeń związanych tyko z projektem *SETI@home*. Opiekunom projektu BOINC, zależy na rozwoju kultury udostępniania swoich zasobów na rzecz naukowych obliczeń. Interfejs systemu jest przyjazny, a sam system łatwy w instalacji.

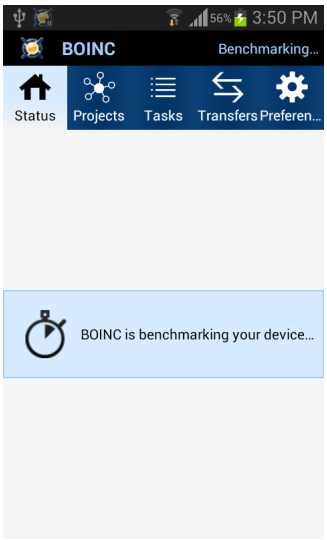
### 3.1.4. Urządzenia mobilne w rozproszonych obliczeniach naukowych

Przez dziesięciolecia, miliony ludzi udostępniały zasoby swoich maszyn aby wspomóc naukę. Jednak komputery typu PC są coraz mniej popularne, a ich sprzedaż spada o prawie 8 procent rocznie. Prognozy jasno wskazują na to, że ten trend się utrzyma. Z drugiej strony, bardzo dynamicznie rozwija się rynek urządzeń mobilnych w tym smartphonów. Do końca roku 2013, na rynek zostanie dostarczonych 919 milionów nowych urządzeń, co jest 27 procentowym wzrostem w stosunku do roku 2012.

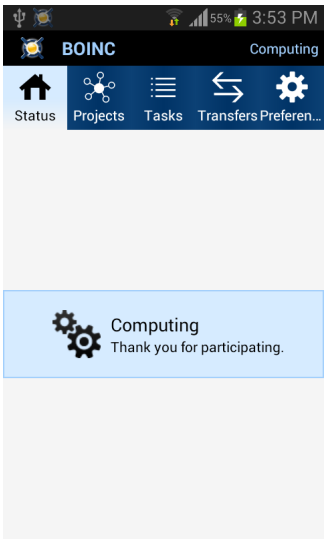
Do niedawna, telefony tego typu nie miały wystarczających zasobów by prowadzić naukowe obliczenia. Obecnie urządzenia tego typu mają nawet 4 rdzennie i mogą wykonywać półtorej miliarda operacji numerycznych na sekundę, co odpowiada około jednej piątej tego co potrafią współczesne komputery osobiste. David Anderson, naukowiec z Berkeley, który współtworzy mobilna aplikacją która jest furtką do naukowych obliczeń na smartphonach, twierdzi, że jako, że hardware mobilnych urządzeń jest obecnie najbardziej rozwijaną gałęzią urządzeń elektronicznych, już wkrótce, moc obliczeniowa tych urządzeń może wyrównać obecne osiągi PC-tów. Między innymi z tego powodu, warto zacząć uwzględniać urządzenia mobilne w roli potencjalnego wykonawcy intensywnych obliczeń.

Ciekawym konceptem jest obliczeniowa sieć stworzona z ogromnej liczby urządzeń, która mogła by rywalizować z takimi dostawcami usług jak Amazon Web Services. W tym momencie wartość rynku komercyjnych obliczeń w chmurze osiąga 131 miliarda dolarów. Sieć stworzona z telefonów może być tanim odpowiednikiem i umożliwić na przykład firmie farmaceutycznej płacić po kilkadziesiąt centów miesięcznie za udostępnienie swojego urządzania np. przez noc do potrzebnych jej obliczeń. Wizję tą roztacza Bernie Meyerson, vice president innowacji w firmie IBM.

W sierpniu 2013, na głównym sklepie z aplikacjami na system Android: Google Play, pojawiła się oficjalna aplikacja stworzona przez zespół BOINC z Berkeley. Aplikacja umożliwia posiadaczowi smartphona w bardzo przyjazny sposób na dołączenie do obliczeń. Rozważając użycie urządzeń mobilnych do obliczeń wywołuje pytanie o zużycie baterii. Aplikacja zaprojektowana jest tak, by dać użytkownikowi pełną kontrolę nad tym jakie warunki muszą zajść aby prowadzone były obliczenia (implikujące szybsze zużycie baterii). Warunkami tymi może być np. podłączenie do źródła zasilania oraz poziom zużycia baterii powyżej zadanego progu.



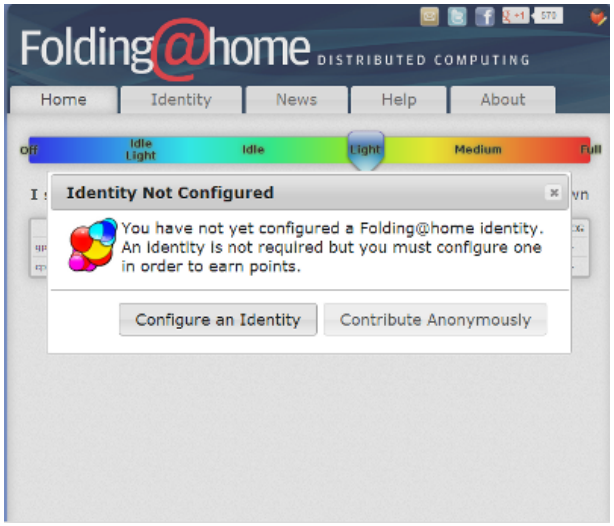

Rysunek 3.3: Proces benchmarkowania urządzenia, po pierwszym uruchomieniu aplikacji BOINC na system Android




Rysunek 3.4: Aplikacja BOINC w trakcie obliczeń dokonywanych na platformie z systemem Android

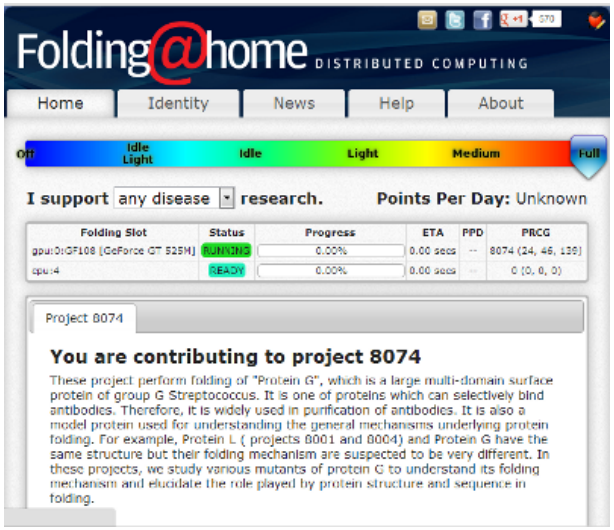
### 3.1.5. Folding@home

Projekt wywodzący się z Amerykańskiego Uniwersytetu Stanford, był pierwszym, który zauważył że potencjał Volunteer Computing nie zamyka się tylko na komputery osobiste i wykorzystanie jednostek CPU ale także coraz bardziej rozwijanych jednostek GPU, procesorów na PlayStation 3. Jako pierwszy duży projekt wykorzystał też model Message Passing Interface. Pojedynczy klienci dostają z serwera część symulacji, i po jej wykonaniu odsyłają ją z powrotem do serwera, w którym części są łączone i stwarzają całościową symulację. Co ciekawe, ochotnicy biorący udział w symulacjach, mogą śledzić swój wkład w projekt Folding@home poprzez stronę internetową. Obserwujemy, że techniki grywalizacyjne zaczęły być coraz szerzej wprowadzane do projektów Volunteer Computing w celu utrzymania zainteresowania ochotników w braniu udziału w projekcie.




Rysunek 3.5: Przyjazny użytkownikowi ekran powitalny interfejsu aplikacji *Folding@home*

Folding@home jest jednym z największych systemów komputerowych na świecie. Jego szybkość szacuje się na około 14 petaFLOPSów na sekundę, więcej niż wszystkich projektów, które są przetwarzane na wywodzącej się z Berkeley platformie BOINC. W 2007 został wpisany do księgi rekordów Guinessa jako system rozproszonych

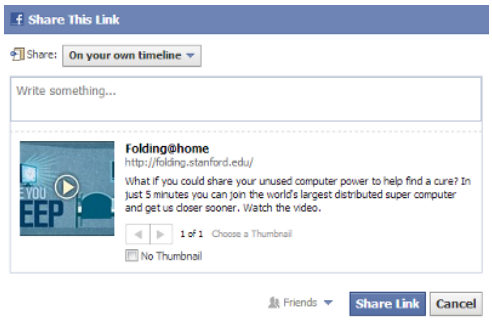



Rysunek 3.6: Monitorowanie postępu obliczeń poprzez interfejs aplikacji*Folding@home*

obliczeń o największej mocy obliczeniowej. Stan na Czerwiec 2013 wskazuje prawie pół miliona aktywnych CPU oraz prawie 25 tysięcy aktywnych GPU.

Folding@home wprowadza elementy grywalizacyjne poprzez wprowadzenie systemu punktowego. Osoby udostępniające swoje zasoby dostają punkty za każde wykonane zadanie, dodatkowe punkty można zdobywać za szybkie i pewne wykonanie pewnych zadań które są szczególnie wymagające obliczeniowo albo mają większy priorytet ze względu na wartość naukową. Gromadzić więcej punków można dzięki włączeniu do programu, pod jednym loginem, wielu swoich maszyn. Ogólnym założeniem systemu punktowego jest motywowanie ochotników do coraz większego zaangażowania i włączania znajomych do dających korzyści nauce rywalizacji. Statystyki, zarówno poszczególnych drużyn jak i indywidualnych osób są widoczne na stronie głównej projektu.

Oprogramowanie Folding@Home jest przyjazne. Ściągnięcie i instalacja nie powoduje żadnych niepokojących ostrzeżeń ze strony systemu operacyjnego. Interfejs kliencki jest interfejsem webowym, łączącym się z nadającą na lokalnej maszynie aplikacją kliencką wykonującą obliczenia. Twórcy platformy starają się ułatwić rozprzestrzenianie się informacji o możliwości kontrybuowania do naukowego projektu dając możliwość udostępniania informacji o programie poprzez portale społecznościowe takie jak Facebook czy Twitter.




Rysunek 3.7: Zachęcanie do udostępniania informacji o programie *Folding@home* w sieciach społecznościowych

# 4. Proponowane rozwiązanie

W rozdziale tym przedstawiono informacje .

## 4.1. Model narciarza i środowiska

W zaproponowanym rozwiązaniu trzeba była podjąć pewne decyzje odnośnie reprezentacji środowiska oraz narciarza. Zostało przyjęte, że stok traktowany jest jako płaszczyzna, która jest nachylona pod określonym przez stałą $\alpha$ kątem do powierzchni ziemi. Założenie co do płaskiej powierzchni jest tylko ograniczeniem przyjętym do testów. Umożliwia to łatwiejszą analizę wyników niezaburzonych zmianami nachylenia terenu. Jednak stworzony program może zostać zmodyfikowany tak, aby zamodelować bardziej skomplikowaną powierzchnię. Narciarz traktowany jest jako punkt materialny o masie m.

## 4.2. Opis matematyczny modelu

m - masa
g - współczynnik grawitacji
$\alpha$ - kąt nachylenia powierzchni stoku do powierzchni ziemi

Siła grawitacji działająca na obiekt o masie m:

$$Q = mg \tag{4.1}$$

Siła ta może zostać rozłożona na dwie siły składowe względem powierzchni stoku. Narciarz poruszający się w dół stoku przypomina klasyczny przykład punktu materialnego staczającego się po równi pochyłej.
Składowa siły grawitacji równoległa do powierzchni stoku:

$$Q_a = mg \sin \alpha \tag{4.2}$$

jest to siła ściągająca narciarza w dół stoku.
Składowa siły grawitacji prostopadła do powierzchni stoku:

$$F_n = mg \cos \alpha \tag{4.3}$$

Wartość tej siły wpływa na wartość siły tarcia działającej na narciarza:

$$F_f = \mu F_n = \mu mg \cos \alpha \tag{4.4}$$

Oprócz siły tarcia uwzględniamy również inną siłę oporu jaką jest siła oporu powietrza. Zależy ona od prędkości poruszania się obiektu. Prędkość będziemy wyrażać jako pierwszą pochodną położenia - $\dot{x}$:

$$F_d = k_1\dot{x} + k_2\dot{x}^2 \tag{4.5}$$

Założenia, które narzucane są na stałe $k_1$ oraz $k_2$:

$$\begin{cases} k_2 = 0 & v \leq B \\ k_1 = 0 & v \geq B \end{cases} \tag{4.6}$$

gdzie $B$:

$$B = 4\frac{m}{s} \tag{4.7}$$

Ograniczenie to zostało zaczerpnięte z pracy ???? (Aerodynamic-Drag) i opisane w rozdziale ????

Współczynnik $k_1$ ....

$$k_1 = ... \tag{4.8}$$

Współczynnik $k_2$ został opisany w ????

$$k_2 = \frac{1}{2}C\rho A \tag{4.9}$$

gdzie:

$C$ - współczynnik oporu

$\rho$ - gęstość powietrza

$A$ - powierzchnia przednia narciarza prostopadła do kierunku przepływu, a zatem prostopadła do wektora prędkości narciarza

## 4.3. Numeryczne rozwiązanie problemu

Rozpatrując wszystkie siły działające na narciarza równoległe do powierzchni stoku, a więc powodujące jego ruch w dół, otrzymujemy następujące równanie:

$$ma = mgsin\alpha - \mu mgcos\alpha - k_1\dot{x} - k_2\dot{x}^2 \tag{4.10}$$

Wyraźmy teraz przyspieszenie jako drugą pochodną położenia:

$$a = \ddot{x} \tag{4.11}$$

Podstawiając do równania:

$$m\ddot{x} = mgsin\alpha - \mu mgcos\alpha - k_1\dot{x} - k_2\dot{x}^2 \tag{4.12}$$

Zatem po podzieleniu przez m:

$$\ddot{x} = gsin\alpha - \mu gcos\alpha - \frac{k_1}{m}\dot{x} - \frac{k_2}{m}\dot{x}^2 \tag{4.13}$$

Aby rozwiązać to równanie numerycznie, wprowadzamy nową zmienną v (odpowiadającą prędkości):

$$v = \dot{x} \tag{4.14}$$

Otrzymujemy następujące równania:

$$\begin{cases} \dot{v} = gsin\alpha - \mu gcos\alpha - \frac{k_1}{m}v - \frac{k_2}{m}v^2 \\ v = \dot{x} \end{cases} \tag{4.15}$$

Pamiętamy przy tym, że:

$$\begin{cases} k_2 = 0 \quad v \leq B \\ k_1 = 0 \quad v \geq B \end{cases} \tag{4.16}$$

Powyższy układ równań wystarczy wykorzystać w dostępnych w wielu językach funkcjach bibliotecznych do rozwiązywania równań różniczkowych zwyczajnych. W naszym przypadku wykorzystałyśmy funkcję dopri (Numerical integration of ODE using Dormand-Prince RK method) z biblioteki Numeric Javascript.

### 4.3.1. Rozwiązanie w 3D

Powyższy układ równań opisuje poruszanie się punktu materialnego po równi pochyłej. Jednak w przypadku narciarza przemieszczającego się po stoku musimy uwzględnić również możliwość poruszania się w poprzek stoku, a nie tylko w dół.
Załóżmy, że narciarz porusza się tylko po liniach prostych.
Rozpatrzmy teraz jak będzie wyglądał układ sił działających na narciarza:

Na naszego narciarza działają siły: grawitacji - w kierunku pionowym oraz siły oporu - w kierunku linii jazdy narciarza. Jeśli rozpatrzymy teraz układ współrzędnych zorientowany względem kierunku jazdy, musimy najpierw zrzutować siłę grawitacji otrzymując siłę ściągająca narciarza:

$$g \sin \alpha sinus \tag{4.17}$$

gdzie sinus to:
Zatem nasze równanie ruchu będzie wyglądało następująco:

$$\ddot{x} = g \sin \alpha * sinus - (\mu F_n + \frac{k_1}{m}\dot{x} + \frac{k_2}{m}\dot{x}^2) \tag{4.18}$$

gdzie $F_n$ to siła nacisku narciarza, która pozostaje taka sama jak w przypadku 2D:

$$F_n = gcos\alpha \tag{4.19}$$

Transformując powyższe równanie na układ współrzędnych zorientowany wzdłuż i wszerz stoku otrzymamy następujące równania:

$$\begin{cases} \ddot{x_x} = (g * \sin \alpha * sinus - (\mu * F_n + \frac{k_1}{m}\dot{x} + \frac{k_2}{m}\dot{x}^2)) * sinus \\ \ddot{x_y} = (g * \sin \alpha * sinus - (\mu * F_n + \frac{k_1}{m}\dot{x} + \frac{k_2}{m}\dot{x}^2)) * cosinus \end{cases} \tag{4.20}$$

Po wprowadzeniu jak poprzednio dodatkowych zmiennych, w tym wypadku prędkości $v_x$ i $v_y$ oraz pamiętaniu o zależności między nimi a pochodną przemieszczenia:

$$\begin{cases} v_x = \dot{x_x} \\ v_y = \dot{x_y} \\ \dot{x} = \sqrt{v_x^2 + v_y^2} \end{cases} \tag{4.21}$$

otrzymujemy:

$$\begin{cases} v_x = \dot{x_x} \\ v_y = \dot{x_y} \\ \dot{v_x} = (g * \sin\alpha * sinus - (\mu * N + \frac{k_1}{m}\dot{x} + \frac{k_2}{m}\dot{x}^2)) * sinus \\ \dot{v_y} = (g * \sin\alpha * sinus - (\mu * N + \frac{k_1}{m}\dot{x} + \frac{k_2}{m}\dot{x}^2)) * cosinus \end{cases} \tag{4.22}$$

$$\begin{cases} v_x = \dot{x_x} \\ v_y = \dot{x_y} \\ \dot{v_x} = gsin\alpha - (\dot{x_x}^2 + \dot{x_y}^2)^{\frac{1}{2}}\kappa\dot{x_y}sgn(\dot{\varphi}) - gsin\alpha\frac{\dot{x_y}^2}{\dot{x_x}^2+\dot{x_y}^2} - \frac{(F_f+F_d)}{m}\frac{\dot{x_x}}{(\dot{x_x}^2+\dot{x_y}^2)^{\frac{1}{2}}} \\ \dot{v_y} = (\dot{x_x}^2 + \dot{x_y}^2)^{\frac{1}{2}}\kappa\dot{x_x}sgn(\dot{\varphi}) + gsin\alpha\frac{\dot{x_x}}{\dot{x_x}^2+\dot{x_y}^2} - \frac{(F_f+F_d)}{m}\frac{\dot{x_y}}{(\dot{x_x}^2+\dot{x_y}^2)^{\frac{1}{2}}} \end{cases}$$

## 4.4. Optymalizacja toru przejazdu

Aby znaleźć rozwiązanie problemu, należy przyjąć jakiś sposób reprezentacji każdego z rozwiązań. Tor przejazdu narciarza w rzeczywistości to ślad, który pozostawiają narty na śniegu w trakcie przemieszczania się po stoku. Jak opisano w rozdziale 4.2 ????, w celu uproszczenia sposobu przemieszczania się narciarza, zostało zdecydowane, że jako przybliżenie można przyjąć poruszanie się po łamanej. Zatem do reprezentacji rozwiązania można przyjąć zbiór punktów, przez które kolejno przejeżdża narciarz, poruszając się między tymi punktami wyłącznie po linii prostej.

Jednak wciąż takie podejście jest niewystarczające, ponieważ w algorytmach optymalizacyjnych potrzebujemy ściślejszego opisu, aby wiedzieć jak skutecznie przeszukiwać przestrzeń rozwiązań. Zatem w zaproponowanym rozwiązaniu narzucamy z góry co ile metrów w pionie stoku ma pojawić się punkt. Można traktować to jako poziome linie, z której każda wyznacza możliwe położenie pojedynczego punktu. Oprócz tego narzucone zostało, że narciarz musi przejechać jak najbliżej każdej wewnętrznej bramki, co indukuje dołożenie punktów w miejscu każdej wewnętrznej bramki. Warunek ten jest spowodowany tym, że w przeciwnym przypadku algorytm miałby do przeszukania dużo więcej rozwiązań. Opierając się na doświadczeniu z rzeczywistych slalomów, wiadomo, że przejeżdżanie tuż przy bramkach jest najkorzystniejsze.

Zatem pozostaje określić w jaki sposób możemy stwierdzić, że dane rozwiązanie jest najlepsze. W przypadku algorytmów optymalizacyjnych zawsze należy określić funkcję celu. W naszym przypadku interesuje nas, aby narciarz w jak najkrótszym czasie dotarł do mety. Mając dane rozwiązanie w postaci punktów wyznaczających łamaną, obliczamy ile czasu zajmie narciarzowi przejechanie po tej trasie. Im mniejsza wartość tym rozwiązanie jest lepsze, gdyż tym mniej czasu potrzebuje narciarz na prawidłowe pokonanie slalomu.

### 4.4.1. Algorytm ewolucyjny

Opisana powyżej reprezentacja rozwiązania to w zastosowanym algorytmie ewolucyjnym pojedynczy osobnik, a punkty składające się na to rozwiązanie, a ściślej, ich położenie w pozycji poziomej określają genotyp każdego osobnika. Nie wprowadzamy tu typowego dla algorytmów genetycznych kodowania binarnego, pozycja każdego punktu jest zapamiętana jako wartość rzeczywista.

Dodatkowo do genotypu wchodzi także parametr $\sigma$, który w strategiach ewolucyjnych używany jest podczas mu-

tacji tak jak opisano w rozdziale 2 w części o strategiach. Każdemu punktowi przypisana jest osobna wartość $\sigma$ - odchylenie standardowe.

Zastosowany algorytm opiera się na strategii ($\mu$ + $\lambda$) opisanej w rozdziale 2. Jako początkową populację wybieramy losowe osobniki - poziome wartości punktów są ograniczone jedynie przez wartości poziome położenia dwóch najbliższych bramek. Jest to kierowane koniecznością zadbania o szybsze znalezienie rozwiązania - zbyt duże odległości można z góry odrzucić opierając się na doświadczeniach z rzeczywistej jazdy narciarza po slalomie. Wielkość populacji bazowej $\mu$ jest jednym z parametrów programu, ale najczęściej wartość ta wynosi 30. Wartość parametru $\lambda$ także jest parametrem, jednak w testach przeważnie użyto wielkości 100.
Szkielet algorytmu zgodny jest z zastosowaną strategią, po wylosowaniu z istniejącej populacji populacji tymczasowej o wielkości $\lambda$, dokonuje się na jej osobnikach operacji genetycznych, najpierw krzyżowania, a następnie mutacji na osobnikach otrzymanych z krzyżowania. Kolejnym krokiem jest ocenienie nowych osobników i wybranie spośród nich oraz populacji początkowej osobników o najlepszym przystosowaniu i to one stanowią nową populację bazową.

#### Krzyżowanie

Aby dokonać krzyżowania potrzebne są pary rodziców dla każdego nowego osobnika. Aby utrzymać wielkość populacji tymczasowej, losujemy (ze zwracaniem) $\lambda$ par spośród populacji tymczasowej. Krzyżowanie rodziców sprowadza się do obliczenia średniej wartości położenia odpowiadających sobie punktów oraz parametrów $\sigma$.

#### Mutacja

Po krzyżowaniu mamy znowu w populacji tymczasowej $\lambda$ osobników. Mutacja osobników przeprowadzana jest zgodnie ze strategią - wykorzystywane są wartości odchyleń standardowych odpowiadających kolejnym punktom. Jedynie punkty, które są przy bramkach nie podlegają mutacji. Wynika to z wcześniejszego założenia, że wtedy otrzymamy rozwiązanie najlepsze.

#### Warunek zakończenia

Warunek zakończenia algorytmu zawsze sprawia wiele problemów. Nie jest łatwo zdecydować na jakiej podstawie zatrzymywać jego działanie. Często korzysta się z informacji o rozrzucie przystosowania w populacji - obliczamy go na podstawie różnicy pomiędzy najlepszym i najgorszym osobnikiem. Jeśli rozrzut ten jest niewielki może oznaczać stagnację algorytmu. Niekoniecznie świadczy to o znalezieniu rozwiązania, ale w połączeniu z dodatkowymi mechanizmami może być skuteczną metodą na decyzję o zakończeniu optymalizacji.
W naszym rozwiązaniu bierzemy zatem również pod uwagę taki wskaźnik jak poprawa najlepszego obecnego rozwiązania. Jeśli przez określoną liczbę iteracji, najczęściej kilka lub kilkanaście najlepsze rozwiązanie nie poprawia się w ogóle, a populacja jest bardzo mało zróżnicowana to jest to znak, że znalezione rozwiązanie powinno być wystarczająco bliskie najlepszemu. Oczywiście sterując liczbą iteracji przez które sprawdzamy zmiany oraz wielkością rozrzutu populacji możemy znajdować lepsze lub gorsze rozwiązania kosztem wydłużenia lub skrócenia czasu działania algorytmu.

### 4.4.2. Hill climbing

Zastosowanie algorytmu genetycznego sprawdziło się w przypadku problemu narciarza, jednak problemem był długi czas wykonywania się programu. Zwłaszcza słaba poprawa wyników występowała w końcowej fazie działania. Widoczne były niepotrzebne próby przeszukiwania zbyt odległych rozwiązań, a jednak wciąż znalezione rozwiązanie nie było tak dobre jak można by tego oczekiwać. Wiedząc, że rozwiązanie jest już dosyć bliskie najlepszemu można z dużym prawdopodobieństwem założyć, że wystarczy znaleźć rozwiązanie lokalnie optymalne, aby było ono satysfakcjonujące. Oczywiście nie mamy pewności, że będzie globalnie optymalne, ale takiej pew-

ności nie możemy mieć nigdy.
Zatem zastosowanie algorytmu lokalnej optymalizacji powinno pomóc w końcowej fazie poszukiwań. Z tego powodu użyty został algorytm Hill climbing opisany w rozdziale 2. W każdym kroku algorytmu sprawdzane jest czy zmiana pojedynczej zmiennej - w tym wypadku poziomej pozycji punktu przejazdu, daje poprawę wyniku. Jeśli zmiany te nie przynoszą rezultatów, są mniejsze niż narzucony parametr $\epsilon$ algorytm zatrzymuje swoje działanie. Wartość $\epsilon$ wynosi przeważnie 0.00001. W przypadku parametrów typowych dla tego algorytmu postanowiono wybrać wartości: dla przyspieszenia standardowa - 1.2, natomiast dla kroku, mniejszą niż zwykle, bo wynoszącą 0.5. Zmiana ta wynika z założenia, że rozwiązanie nie powinno potrzebować większych zmian, aby znaleźć rozwiązanie jak najlepsze.

## 4.5. Modelowanie karania

## 4.6. Uczenie maszynowe

Oprócz użycia algorytmu genetycznego oraz algorytmu Hill climbing warto byłoby sprawdzić czy inne podejście do problemu może dawać lepsze rozwiązanie. Zamiast analizować trasę jako całość, można spróbować tak jak w rzeczywistości pozwolić narciarzowi decydować o tym jak pojechać. Dobry narciarz korzysta z wiedzy o położeniu dwóch lub trzech kolejnych bramek i na tej podstawie podejmuje decyzję o torze jazdy. W takim podejściu można spróbować zastosować uczenie maszynowe.

Jeśli chodzi o reprezentację problemu to zachowane zostało dotychczasowe założenie o poziomych liniach wyznaczających kolejne położenia w poziomie toru jazdy. Jednak w tym przypadku konieczne jest dołożenie również linii pionowych, a więc utworzenie siatki na stoku, której węzły wyznaczają możliwe punkty tworzące tor przejazdu. Wciąż obowiązuje zasada, że kolejne punkty muszą znajdować się na kolejnych poziomych liniach, tak samo jak konieczność przejazdu przez punkty, które reprezentują bramki. Oczywiście można powiedzieć, że takie rozwiązanie jest mniej dokładne, ale wszystko zależy od gęstości siatki.

### 4.6.1. Uczenie ze wzmocnieniem

Załóżmy na razie, że nasz narciarz zna położenie tylko jednej bramki wprzód, a dokładniej, w każdym kroku wie w jakiej odległości w poziomie i pionie znajduje się kolejna bramka. Jego zadaniem jest nauczenie się jak na podstawie tej wiedzy zachować się, aby jak najszybciej dotrzeć do celu.
Niech slalom składa się tylko z jednej bramki. Opierając się na informacjach z rozdziału **??**, bardzo łatwo przedstawić nasz problem w postaci uczenia się ze wzmocnieniem. Narciarz jest uczniem, który poprzez serię prób i błędów, stwierdza jakie zachowanie daje najlepsze efekty. Stanem w naszym środowisku będzie odległość narciarza (w poziomie i pionie) od najbliższej bramki. Akcja to wybór zmiany położenia poziomego na stoku, poruszanie się w dół stoku jest określone poprzez parametr wejściowy. Po wykonaniu akcji, zmienia się położenie narciarza a zatem i jego stan. Ponieważ poziome linie wyznaczają ile punktów znajdzie się pomiędzy każdymi dwiema bramkami, więc zawsze, bez względu na ilość bramek, dla konkretnego ich ułożenia nasze rozwiązanie będzie składało się z określonej liczby punktów. Widzimy zatem, że nasz algorytm będzie polegał na wielokrotnym powtarzaniu przejazdu slalomu, a ilość kroków w każdym powtórzeniu będzie stała. Oznacza to, że naturalnym jest nazwanie tego problemu zadaniem epizodycznym opisanym w rozdziale **??**, jednak nie jest to zadanie do-sukcesu lub do-porażki, gdyż narzucone jest, że liczba kroków jest zawsze stała w epizodzie. Zatem w każdym epizodzie narciarz musi otrzymywać wzmocnienie, które będzie wskazywało jak dobry jest tor, którym właśnie się poruszał. Najważniejszy w tej kwestii jest oczywiście czas, w jakim narciarz jest w stanie pokonać dany slalom. Zatem nagroda powinna zawsze być funkcją tego czasu. Przykładowo, może być to funkcja liniowa: $a - t$, gdzie $a$ to pewna stała,

a $t$ to czas przejazdu jaki uzyskał uczeń. Im krótszy czas, tym wyższa nagroda i odwrotnie. Prawidłowe dobranie funkcji nagrody, a także jej parametrów, tak jak $a$, może niestety wpływać na skuteczność rozwiązania. Pozostałe przykłady funkcji oraz ich wpływ na jakość rozwiązania opisane są w rozdziale z eksperymentami (5). Gdyby slalom składał się z większej ilości bramek, nagrodę narciarz otrzymałby przy każdej bramce, a czas brany do obliczania byłby czasem potrzebnym na przejazd od poprzedniej bramki - w przypadku znajomości odległości do większej ilości bramek, brano by pod uwagę czas przejazdu takiej liczby bramek.

### 4.6.2. Q-learning

W zastosowanym rozwiązaniu użyty został algorytm Q-learning opisany w rozdziale 2.5.1. W algorytmie tym stosowane są dwa parametry: $\alpha$, $\gamma$, których wartość trzeba dostosować, aby otrzymać jak najlepsze rozwiązanie. Testy przeprowadzone do znalezienia tych wartości są opisane w rozdziale 5.

### 4.6.3. Akcja

Przyjrzyjmy się dokładniej akcjom, które podejmować będzie narciarz.
Za każdym razem otrzymywana jest lista akcji, które można wykonać będąc w danym stanie. Aby ograniczyć liczbę możliwości, a jednocześnie nie odrzucać najlepszego rozwiązania, trzeba było zdecydować jakie i ile akcji powinno być dostępnych. Opierając się na doświadczeniach rzeczywistych, wiadomo że narciarz najczęściej kieruje się w stronę najbliższej bramki i nie wykonuje bardzo gwałtownych skrętów. Dlatego zakładając, że bramka jest po prawej stronie, narciarz ma dokładnie cztery możliwości: przesunąć się o krok w lewo, nie przemieszczać się w poziomie w ogóle, o krok w prawo, lub dwa kroki w prawo. Symetryczna sytuacja jest dla bramki umieszczonej z lewej strony. Oczywiście krok oznacza tu szerokość narzuconej siatki. W przypadku gęstej siatki, ilość możliwości powinna być zwiększona, jednak zwiększa to także złożoność algorytmu.
Z powodu nałożenia decyzji o przejeżdżaniu przez punkty reprezentujące bramki, będąc tuż przed bramką, jedyną możliwością narciarza jest pojechanie prosto do tej bramki - zamiast czterech opcji dostępna jest tylko jedna, która wymusza przejazd do punktu-bramki.

Kolejną ważną decyzją jest wybór sposobu w jaki narciarz decyduje o wybraniu akcji spośród możliwych do wykonania w danym stanie. W wykonanych testach wypróbowane zostały dwie możliwości:

– strategia zachłanna - wybór pada zawsze na najlepszą z możliwości

– ...

## 4.7. Architektura systemu

Wybór docelowej architektury dla naszego systemu został poprzedzony eksperymentowaniem z dwoma różnymi, zgoła odmiennymi podejściami. Dzięki temu, zostały przeanalizowane mocne i słabe punkty każdej z nich a ostatecznie wybrane rozwiązanie spełnia wszystie potrzeby.

### 4.7.1. Architektura prototypowa

Pierwszym wybranym przez nas środowiskiem tworzenia systemu był język Python i dostępne dla niego moduły: VPython umożliwiający wizualizację w 2D i 3D oraz biblioteki numeryczne NumPy i SciPy wykorzystywane do rozwiązywania równań różniczkowych.

Dobrze znany nam język pozwolił nam na szybkie prototypowanie i testowanie pierwszego fizycznego modelu narciarza. Wizualizacja była prosta w implementacji, a biblioteki numeryczne wystarczająco dobrze udo-

kumentowane i powszechnie używane, co zapewniało ich stabilność i jakość. Szybko ujawniły się jednak wady wybranego środowiska - uruchomienie naszego programu wymagało od użytkownika instalacji złożonego środowiska do obsługi języka Python oraz graficznych i numerycznych bibliotek co było istotną barierą. Bariera ta była istotnym problemem, biorąc pod uwagę potencjalną możliwość rozpraszania obliczeń. Zdałyśmy sobie sprawę, że rozwiązywany przez nas problem i otrzymywane przez nas wyniki są atrakcyjne wizualnie i ludzie związani z narciarstwem z chęcią mogą wziąć udział w obliczeniach i udostępnić nam zasoby obliczeniowe swoich komputerów. Warunkiem koniecznym by było to możliwe jest łatwość w dołączaniu do obliczeń, wizualizacji wyników i bezproblemowa konfiguracja. Biorąc to pod uwagę, zdecydowałyśmy że potrzebujemy innego środowiska.

### 4.7.2. Architektura docelowa

Potrzeba łatwości dołączania do obliczeń skłoniła nas do stworzenia systemu w którym klientami obliczeniowymi są przeglądarki internetowe. Klienci chcący podłączyć się do obliczeń wchodzą na dobrze znany adres internetowy skąd serwowana jest strona główna i skrypty które dokonują obliczeń. Na żądanie klienta, dostaje on ze zdalnego serwisu instancję problemu, która jest następnie lokalnie przetwarzana i wizualizowana klientowi na bieżąco. Po skończonym obliczeniu, rozwiązanie jest umieszczane na serwerze a klient może przejść do obliczania kolejnego problemu, lub jeszcze raz tego samego problemu.

Architektura naszego systemu składa się z dwóch głównych, niezależnych komponentów. Pierwszym jest aplikacja kliencka przeprowadzająca obliczenia i wizualizację oraz komunikująca się z serwerem zarządzającym obliczeniami. Drugim jest aplikacja lekkiego serwera http, jako warstwa wystawiająca RESTowe API. Warstwą persystentną jest dokumentowa, nierelacyjna baza danych CouchDB.

### 4.7.3. Aplikacja kliencka

Aplikacja kliencka, to aplikacja webową stworzoną przy użyciu frameworku Chaplin, wprowadzającego dobre praktyki w strukturyzowaniu aplikacji Java Scryptowych opartych na bibliotekę Backbone.js. Aplikacja w całości jest pisana w języku Coffee Script. Coffee Script to przyjazdy w składni język programowania, kompilujący się do Java Scriptu. Powstały po kompilacji kod Java Script jest czytelny i przyjazny do debugowania. Programując w Coffee Script można używać wszystkich bibliotek i modułów napisanych w Java Script, bo koniec końców wszystko zamieni się w Java Script. Zdecydowałyśmy się używać tego języka przede wszystkim z uwagi na składnie przypominającą dobrze znany nam język Python, większą przejrzystość i czytelność kodu.

Aplikacja ma zasadniczo trzy warstwy:

– warstwę prezentacyjną i wizualizacyjną dla rozwiązywanego problemu

– warstwę komunikacji z serwerem

– warstwę obliczeniową

Sekwencję komunikacji pomiędzy poszczególnymi warstwami aplikacji klienckiej, przedstawia Diagram 4.1 na stronie 39

#### Warstwa prezentacji

Wizualizacja zaimplementowana jest na elemencie Canvas ze specyfikacji HTML5. Wykresy wizualizujące przebieg obliczeń stworzone są przy użyciu biblioteki Highcharts. Warstwa spełnia następujące funkcję:

– zaznajamiająca użytkownika wchodzącego na stronę internetową z tym czemu strona służy

– umożliwienie rozpoczęcia wzięcia udziału w obliczeniu

– prezentowanie trasy narciarskiej, popranej z serwera do obliczeń

– wizualizacja na bieżąco wyników obliczeń w postaci bieżąco rozważanego toru przejazdu

– wizualizacja parametrów algorytmu obliczającego w postaci wykresów

**Warstwa komunikacyjna**

Komunikacja z serwerem odbywa się poprzez REST-owe API, wykorzystujące JSONowy format do wymiany danych. Warstwa spełnia następujące funkcję:

– pobieranie instancji problemu do obliczenia

– wysyłanie na serwer rozwiazania zadanego problemu

– pobierania z serwera informacji o obecnie najlepszym rozwiązaniu tego problemu

**Warstwa obliczeniowa**

Zastosowanie w naszej architekturze Web Workerów do wykonywania obliczeń jest dobrze uzasadnione, biorąc pod uwagę ich specyfikację, opisaną szerzej w rozdziale 2.7. Obliczenia trasy przejazdu nawet dla toru z kilkoma bramkami zajmuje co najmniej kilka sekund. Worker na bieżąco, w trakcie trwania obliczeń, przekazuje do głównego procesu obsługującego UI strony, kolejne rozważane trasy. Proces UI rysuje na elemencie Canvas na bieżąco otrzymywaną trasę.

### 4.7.4. Aplikacja serwerowa

Aplikacja jest lekkim serwerem Http napisanym w środowisku Node.js. Użytym językiem programowania podobnie jak w aplikacji klienckiej, jest Coffee Script. Serwer wystawia REST-owe API, używając do wymiany danych JSON-owego formatu serializacji. Serwer komunikuje się z dokumentową, nierelacyjną bazą danych CouchDB. Podstawowymi funkcjami serwera są:

– umożliwienie dodania instancji problemu obliczeniowego do bazy danych

– zwracanie na żądanie problemu obliczeniowego do rozwiązania

– zwracania informacji o obecnych dostępnych rozwiązaniach każdego z problemów obliczeniowych

Diagram 4.2 na stronie 40 pokazuje sekwencję komunikacji pomiędzy komponentami systemu oraz warstwą persystencji.

### 4.7.5. Efekt końcowy

Efektem widocznym dla końcowego użytkownika powstałej na potrzeby tego projektu aplikacji jest strona www, dostępna pod dobrze znanym adresem URL - http://giant-client.herokuapp.com/. Na stronie jest krótko wytłumaczone o co chodzi w projekcie i w jaki sposób każdy odwiedzający może pomóc w obliczeniach. Na stronie znajduje się jedno wyraźne tzn. ćall to action- duży przycisk zachęcający do rozpoczęcia obliczeń.

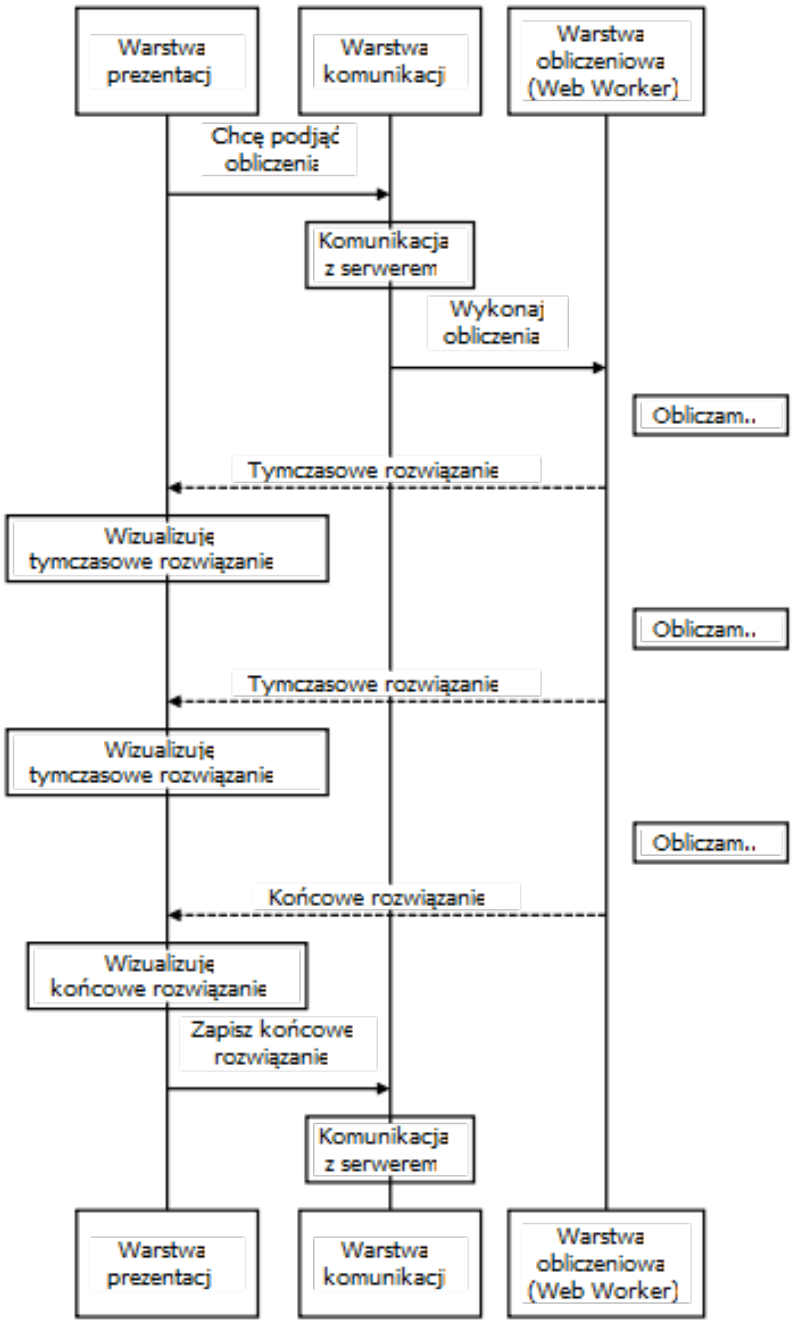
Po uruchomieniu obliczeń, osoba odwiedzająca stronę dostaje na bieżąco informacje zwrotne dotyczące efektów jej współpracy. Na stronie:

– rysowana jest cały czas aktualnie rozważana trasa przejazdu

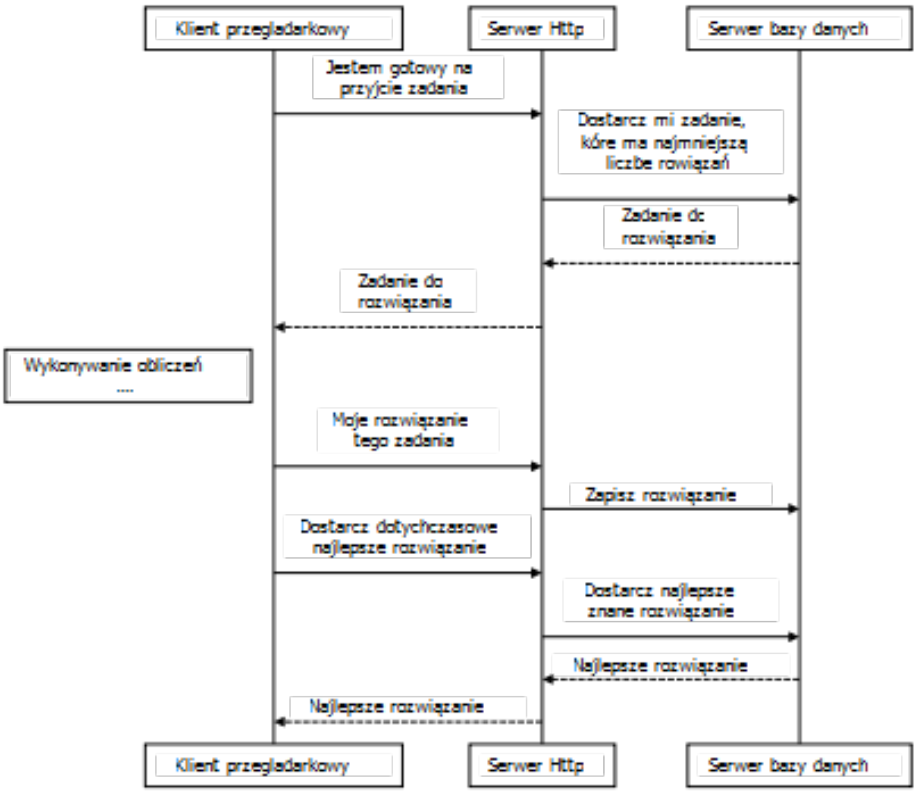
– wyświetla się obliczony czas przejazdu po aktualnie rozważanym torze przejazdu

– wyświetla się najlepszy, dotychczas znaleziony, czas przejazdu dla aktualnie rozważanej konfiguracji trasy

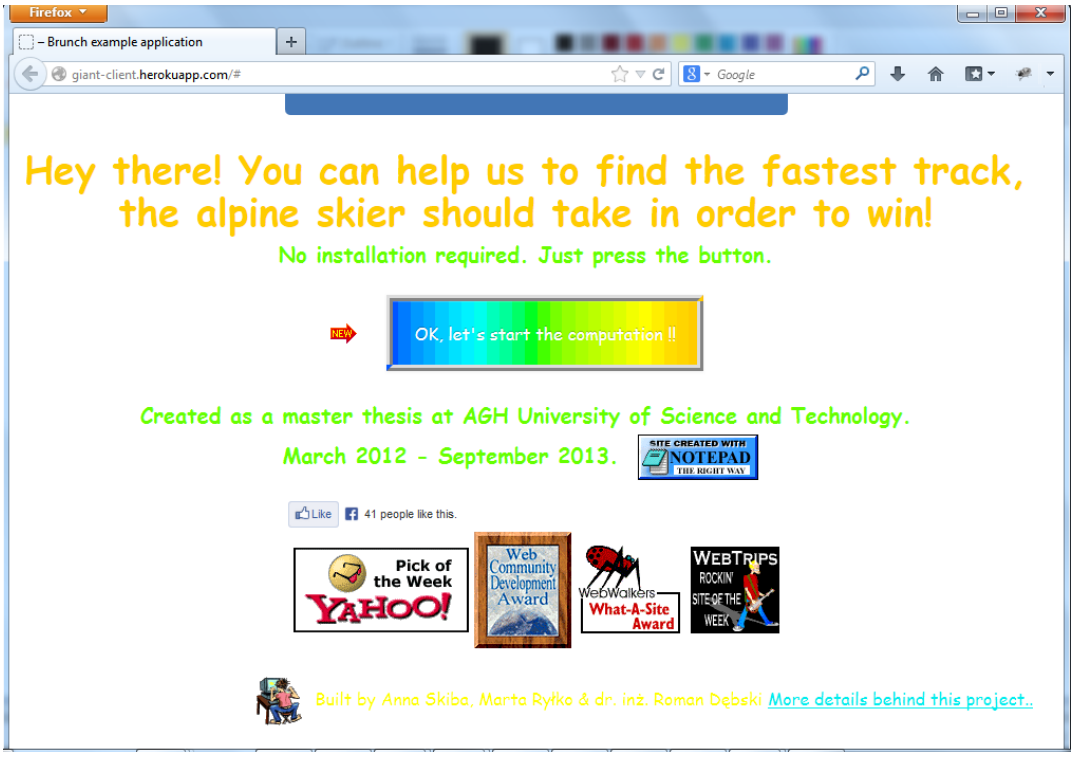
– rysowane są wykresy przedstawiające parametry algorytmu genetycznego (aktualny Fitness populacji)




Rysunek 4.1: Sekwencję komunikacji pomiędzy poszczególnymi warstwami aplikacji klienckiej




Rysunek 4.2: Sekwencja komunikacji pomiędzy poszczególnymi komponentami systemu oraz warstwą persystencji




Rysunek 4.3: Nasza strona zachęcającą do uruchomienia obliczeń


Rysunek 4.4: Nasza strona, podczas prowadzonych obliczeń

# 5. Wyniki

W rozdziale tym przedstawiono informacje .

Jeżeli nie jest wskazane inaczej w eksperymencie, przyjmowane są następujące wartości stałe:

- $\mu = 0.05$ - współczynnik tarcia, typowa wartość dla nasmarowanych nart
- $\rho = 1.17 \frac{kg}{m^3}$ - przybliżona wartość gęstości powietrza dla temperatury $0°C$ i wilgotności 20% na wysokości 800 m n.p.m
- $C = 0.6$ - współczynnik oporu powietrza, typowe wartości to 0.4 - 1
- $A = 0.2m^2$ - frontalna powierzchnia narciarza w projekcji prostopadłej do wektora prędkości narciarza
- $k_2 = 0.5 * C\rho A$ - współczynnik oporu powietrza
- $k_1 = 0.05$ - współczynnik oporu powietrza

W eksperymentach wszystkie współrzędne punktów są w układzie kartezjańskim, dwuwymiarowym, zorientowanym na płaszczyźnie stoku. Oś x skierowana jest wzdłuż stoku, natomiast oś y w poprzek.

## 5.1. Weryfikacja modelu

W tej sekcji chciałyśmy pokazać, że model, który został przyjęty jest prawidłowy. W poniższych eksperymentach zostanie pokazane, że zależności pomiędzy zmianą poszczególnych zmiennych modelu, takich jak masa, nachylenie stoku oraz opory ruchu, a otrzymywanym czasem przejazdu, jest zgodne z prawami fizyki. Kolejne eksperymenty ukażą również, że czas przejazdu po teoretycznie najszybciej trasie jest praktycznie identyczny z tym, który można wyliczyć ze wzorów przedstawionych w artykułach naukowych. Na koniec sprawdzimy czy możliwe jest przybliżanie jazdy narciarza jazdą po łamanej i jak gęsto będą musiały być rozmieszczane punkty przegięcia, żeby nie stracić na dokładności.

### 5.1.1. Masa i nachylenie stoku

Na początek zostały przeprowadzane proste eksperymenty, które dowodzą, że przyjęty model jest poprawny. Dlatego dla celów pierwszego testu najłatwiej było sprawdzić jak zmienia się czas przejazdu narciarza poruszającego się po prostej w dół stoku przy różnych masach narciarza oraz zmianach nachylenia stoku, dodatkowo sprawdzając jaki wpływ na wyniki ma również tarcie oraz opór powietrza.

Analizując równanie 4.13, które opisuje model poruszania się narciarza zauważamy, że jego masa wpływa jedynie na wartość przyspieszenia wynikającą z siły oporu powietrza. Natomiast kąt nachylenia $\alpha$ wpływa zarówno na siłę ściągającą, jak i na siłę tarcia. W opisanym eksperymencie zostanie pokazane, że:

– im większy kąt nachylenia stoku tym szybciej porusza się narciarz, bez względu na rodzaj uwzględnionych sił oporu

– jeśli uwzględnimy tylko siłę ściągającą, bez sił oporu czas przejazdu narciarzy o różnych masach na tej samej trasie jest identyczny

– taki sam wynik otrzymamy jeśli uwzględnimy dodatkowo siłę tarcia

– przy uwzględnieniu siły oporu powietrza otrzymujemy następującą zależność: im większa jest masa narciarza, tym mniejsza jest siła oporu powietrza co skutkuje szybszym przejazdem narciarza (przy założeniu, że frontalna powierzchnia narciarza $A$ nie zmienia się)

**Opis eksperymentu**

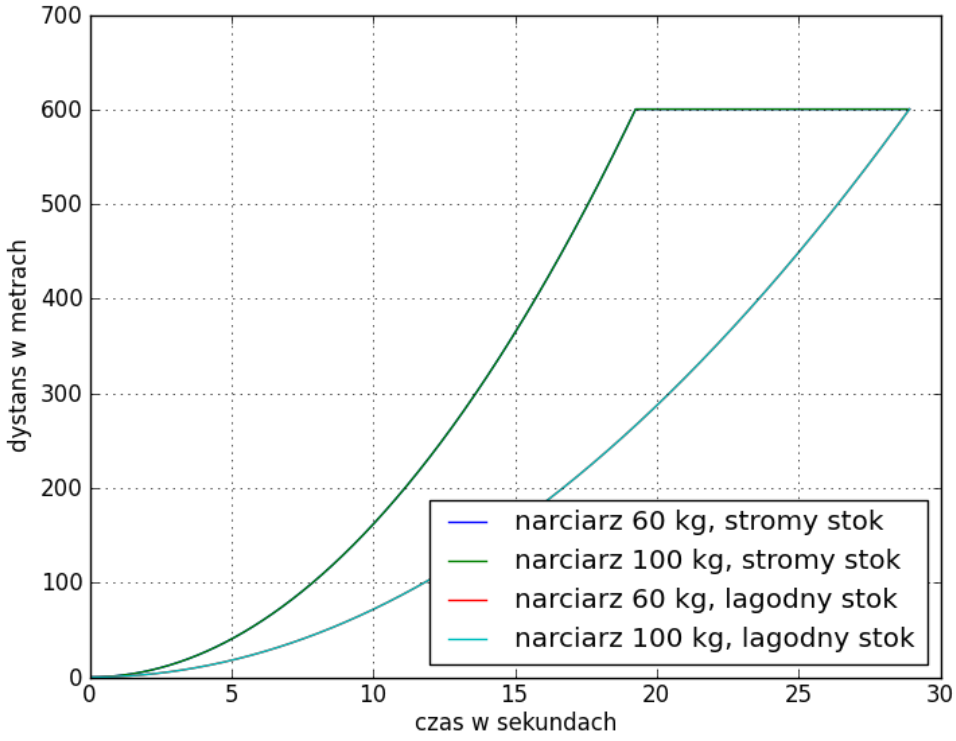
Eksperyment został przeprowadzony na dwóch modelach stoków narciarskich. Oba mają 600 m długości co odpowiada rzeczywistej długości stoku Harenda w Zakopanem. Stoki modelujemy jako równie pochyłe o stałym nachyleniu. Aby zweryfikować prawidłowość modelu, w eksperymencie zostały wprowadzone rzeczywiste wartości nachyleń stoków Harenda w Zakopanem oraz Kotelnica w Białce Tatrzańskiej.

– Harenda - ok. $20^\circ$ (0.3367 radianów)

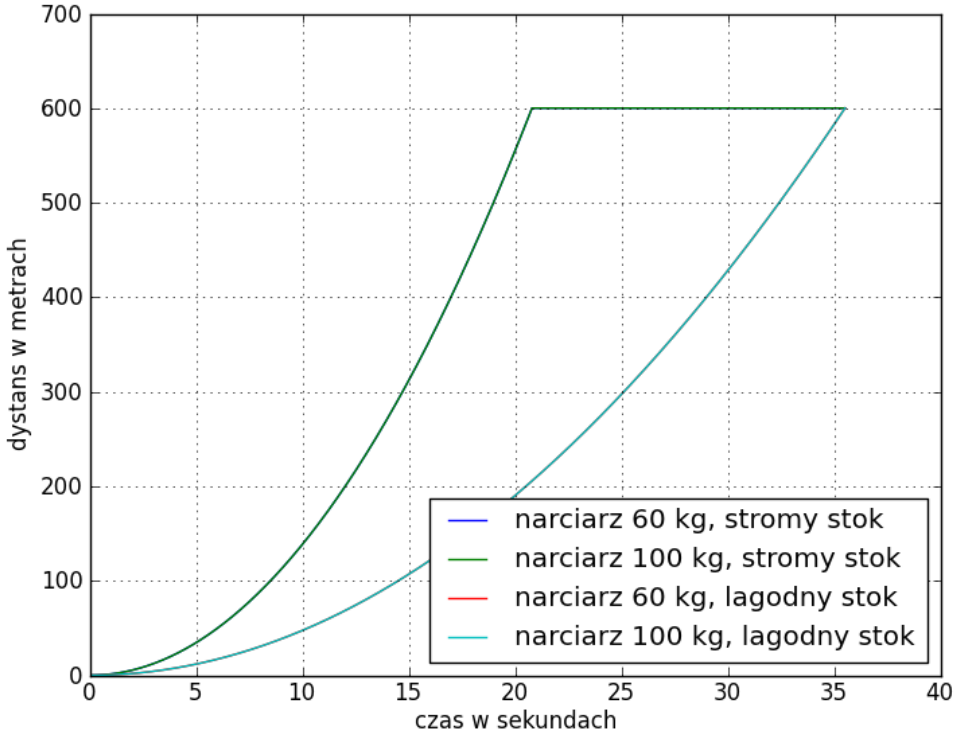
– Kotelnica - ok. $8^\circ$ (0.1470 radianów)

Zostały przeprowadzone trzy próby, których wyniki można zobaczyć odpowiednio na kolejnych wykresach 5.1.1, 5.1.1, 5.1.1. W pierwszej części eksperymentu uwzględniamy tylko i wyłącznie siłę grawitacji, w drugiej dokładamy wyłącznie siłę tarcia , natomiast w trzeciej uwzględniamy wszystkie siły oporu, a więc także siłę oporu powietrza.

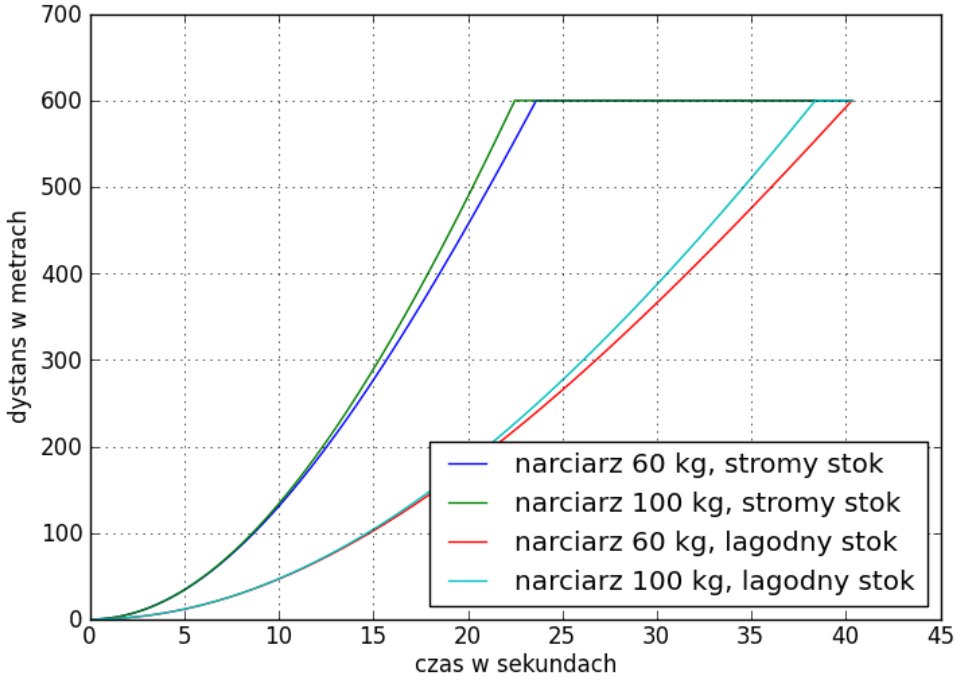
**Rezultaty eksperymentu**




Rysunek 5.1: masa i stok




Rysunek 5.2: masa i stok




Rysunek 5.3: masa i stok

Obserwujemy, że w każdym z trzech przypadków na stromym stoku narciarze przejeżdżają ten sam dystans w krótszym czasie. W pierwszej i drugiej części eksperymentu nie ma znaczenia masa narciarza - czas przejazdu jest taki sam. Jednak na wykresie 5.1.1 czas przejazdu wynosi odpowiednio ok. 19 i 28 s., natomiast jeśli uwzględnimy siłę tarcia, czasy te wydłużają się do ok. 21 i 35 s.
Analizując wykres 5.1.1 zauważamy, że na bardziej stromym stoku różnica w czasie przejazdu pomiędzy cięższym a lżejszym narciarzem wynosi 1.14 s na korzyść cięższego, a na łagodniejszym 1.94 s również na korzyść cięższego. Zmierzone czasy przejazdu slalomu w rzeczywistości na stoku Harenda wynoszą ok. 40 s, a więc wydaje się rozsądne, że przejazd na wprost może zajmować ok. 22 s, które zostały otrzymane w eksperymencie. W porównaniu z poprzednimi częściami eksperymentu łatwo spostrzec, że czas przejazdu jest jeszcze dłuższy - 22-23 oraz 38-40 s.

### 5.1.2. Czas przejazdu w zależności od toru jazdy

Optymalizacja trasy narciarza polega na znalezieniu takiej linii jazdy, która spowoduje najszybszy przejazd. Warto tu zauważyć, że wbrew pozorom, nie oznacza to zawsze najkrótszej linii. W poniższym eksperymencie zostanie pokazane jak zmienia się czas przejazdu w zależności od linii poruszania się.

Jeśli rozważymy przykład bez bramek, gdzie najkrótsza linia przejazdu nie powoduje jazdy w skos stoku, a jedynie w dół, to w takim przypadku, jest to jednocześnie najszybsza trasa przejazdu. Jednak w pozostałych przypadkach, jeśli rozważamy układ bez działania oporów, linie te będą różne.

#### Opis eksperymentu

Chcemy sprawdzić ile czasu zajmie narciarzowi przejechanie z punktu (0,0) do punktu (10,10). Aby uprościć przykład, sprawdzimy wyniki pomijając wkład wszystkich oporów. Gdyby narciarz poruszał się po linii prostej, bezpośrednio do punktu końcowego, czas przejazdu można wyliczyć przekształcając wzór:

$$s = \frac{at^2}{2} \tag{5.1}$$

i otrzymując:

$$t = \sqrt{\frac{2s}{a}} \tag{5.2}$$

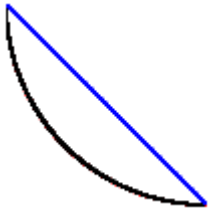
gdzie:

- $s = 10\sqrt{2}$ - odległość między punktem początkowym i końcowym
- $a = g\sin(\frac{\pi}{12})\sin(\frac{\pi}{4})$ - przyspieszenie narciarza

Przyspieszenie narciarza jest składową przyspieszenia ziemskiego uwzględniającą zarówno nachylenie stoku ($\frac{\pi}{12}$), jak i jazdę w skos stoku ($\frac{\pi}{4}$, czyli $45°$). Zatem czas przejazdu po linii prostej powinien wynosić około:

$$t = \frac{2 * 10\sqrt{2}}{9.81 * \sin(\frac{\pi}{12})\sin(\frac{\pi}{4})} \simeq 3.969 \tag{5.3}$$

Sprawdzimy teraz jakie czasy otrzymamy wykorzystując program, sprawdzając poruszanie się po linii prostej oraz po ćwiartce okręgu (a dokładniej, po łamanej, której kolejne punkty załamania znajdują się na okręgu w niewielkich odległościach od siebie). Poniżej widzimy jak wygląda trasa przejazdu w dwóch przypadkach.



**Rezultaty eksperymentu**

W tym przypadku otrzymaliśmy wyniki odpowiednio dla narciarza jadącego po prostej 3.965, a dla poruszającego się po czarnej linii 3.671.

Po pierwsze, warto zwrócić uwagę, że czas jazdy po prostej minimalnie różni się od czasu teoretycznego, obliczonego ze wzoru - różnica ta wynosi ok. 0.1%, co jest dopuszczalne i możliwe do zaniedbania, gdyż nie ma znaczącego wpływu na wyniki eksperymentów. Różnica ta może wynikać z niedokładności obliczeń numerycznych, a także z trudności trafienia przez narciarza idealnie w punkt końcowy, w którym powinien się znaleźć.

Wyraźnie widać również, że czas przejazdu po linii czarnej jest krótszy niż czas przejazdu po linii niebieskiej. Trasę po prostej narciarz przebywa w ok. 7% dłuższym czasie niż trasę dłuższą. Na tak krótkim odcinku narciarz był w stanie zdobyć przewagę ok. 0.3 s. Ważne jest to, że w prawdziwych przejazdach slalomowych nawet najmniejsza różnica czasu może decydować o wygranej, oczywiście zależy to od dokładności pomiaru, który zawsze jest sprawdzany do setnych części sekundy. Dlatego ogromne znaczenie ma tor po jakim porusza się narciarz i nawet najmniejsza zmiana może powodować kluczową poprawę czasu na całej trasie. Czasami warto pojechać dłuższą trasą, ale taką, na której nabierzemy większej prędkości i później zyskamy na kolejnym odcinku.

Pozostaje jeszcze pytanie jaka jest najlepsza linia poruszania się pomiędzy dwoma punktami. Problem ten został już rozwiązany - linia najszybsza jest fragmentem cykloidy. Dokładniejsze informacje można znaleźć m.in. w http://www.hep.caltech.edu/ fcp/math/variationalCalculus/variationalCalculus.pdf oraz http://www.math.utk.edu/ freire/teaching/m231f08/m231f08brachistochrone.pdf. W http://www.math.utk.edu/ freire/teaching/m231f08/m231f08brachistochrone.pdf znajduje się wzór na czas przejazdu pomiędzy dwoma punktami. Korzystając z tego wzoru podstawiając odpowiednio wyliczone parametry otrzymujemy, że najkrótszy możliwy czas przejazdu z punktu (0,0) do punktu (10,10) to ok. 3.623. Jak widać, wynik ten jest lepszy niż otrzymane w naszym eksperymencie: od czasu jazdy po ćwiartce koła o ok. 0.05 s., co nie jest różnicą bardzo dużą, ale znaczącą dla zwycięstwa.

### 5.1.3. Jazda po łamanej

Kolejnym krokiem było sprawdzenie, że rzeczywiście możemy przybliżać jazdę narciarza jako poruszanie się po łamanej. Ważne jest stwierdzenie jakie różnice w czasie będą występować pomiędzy jazdą po łuku, a jazdą po łamanej. Różnice te oczywiście będą zależeć od tego jak dokładne przybliżenie zrobimy.

**Opis eksperymentu**

W eksperymencie sprawdzać będziemy jak zmienia się czas przejazdu w zależności od dokładności przybliżenia łamanej. Jako tor przejazdu przyjęta została najszybsza linia przejazdu do punktu (10,10), po której przejechanie zajmuje 3.623 s (jak wspomniano w poprzednim eksperymencie 5.1.2).

Wybierając określoną liczbę punktów znajdujących się na linii łuku sprawdzimy ile czasu zajmuje przejechanie po łamanej utworzonej z tych punktów w porównaniu z jazdą po łuku. Należy również określić jak często muszą znajdować się punkty, aby różnica w czasie była zaniedbywalnie mała. Sterowanie odbywa się poprzez ustalenie jak gęsto w pionie mają znajdować się punkty - w ten sposób również możemy sterować ilością punktów.

**Rezultaty eksperymentu**

Poniżej znajdują się obrazki przedstawiające poszczególne próby przejazdu po łamanej składającej się odpowiednio z 2-6 punktów pośrednich (linia szara) oraz pierwotną linię najszybszego przejazdu (linia niebieska). Ilość punktów na łamanej nie bierze pod uwagę punktu startowego, a jedynie kolejne punkty włącznie z punktem na mecie.

Oto uzyskane wyniki w zależności od ilości punktów pośrednich oraz różnica czasu w porównaniu do najkrótszego czasu przejazdu - 3.623.

| ilość punktów pośrednich | czas przejazdu [s] | różnica w stosunku do czasu najszybszego [s] |
|---|---|---|
| 2 | 3.687 | 0.064 |
| 3 | 3.647 | 0.023 |
| 4 | 3.635 | 0.012 |
| 5 | 3.629 | 0.006 |
| 6 | 3.626 | 0.003 |

Różnice w czasie wahają się od 0.003 do 0.064 s, jest to odpowiednio od 0.08% do ok. 1.8% różnicy. Wydaje się, że wystarczającym powinno być, aby na tym odcinku określić trzy punkty pośrednie, gdyż różnica dla tego przypadku wynosi ok. 0.023 s co stanowi ok. 100.6% czasu najszybszego, a więc różnica nie przekracza 1%. Zresztą to samo można stwierdzić z rysunków - już dla trzech punktów praktycznie nie widać różnicy pomiędzy liniami. Zatem na takiej trasie, której odległość w pionie wynosi 10 m można określić 3-4 punktów pośrednich, które z przybliżeniem nie wnoszącym zbyt dużych różnic do wyniku będą wyznaczały trasę końcową. Oznacza to, że punkty te powinny znajdować się w odległościach 2-4 m. Wiedząc, że pomiary przejazdu na zawodach są z dokładnością do setnych części sekundy, można stwierdzić, że dopiero 5 punktów pośrednich daje nam aż taką precyzję obliczeń. Oczywiście jeśli trasa nie będzie zawierać zbyt ostrych kątów załamania, można będzie zmniejszyć gęstość punktów.

## 5.2. Optymalizacja

W tej części zostaną pokazane wyniki jakie zostały otrzymane jako efekt działania algorytmu ewolucyjnego w celu znalezienia optymalnej trasy przejazdu. Sprawdzimy jak poszczególne parametry algorytmu wpływają na czas jego działania oraz uzyskiwane wyniki. Spróbujemy także zwiększyć ilość punktów, które stanowią rozwiązanie oraz wykonać algorytm na dłuższej trasie.

### 5.2.1. Algorytm ewolucyjny

Wiedząc, że założenia dotyczące modelu oraz sposób poruszania się narciarza można uznać za poprawne i zbliżone do tego co rzeczywiście dzieje się podczas jazdy na stoku, zostanie przetestowane to, jak sprawdza się algorytm genetyczny do celów optymalizacji trasy slalomu.

W poniższych eksperymentach wykorzystane zostanie następujące ustawienie bramek: (5,13), (0,26), (5,39), (4,44), (11,57), (0,70). Ustawienie to jest podobne do tych, które są wybierane w rzeczywistości. Posiada także wertikal, figurę slalomową, szerzej opisaną w sekcji 2.1, często stosowany w slalomie, który ma wymuszać zmianę rytmu, a więc będzie dobrym sprawdzianem dla zastosowanego algorytmu.

Zakładamy, że narciarz startując ma prędkość początkową $1\frac{m}{s}$. Jest to kierowane tym, że przy starcie zawodnicy zaczynają od odepchnięcia się kijami od śniegu nadając sobie prędkość początkową. Najważniejsze jest to, że wartość ta będzie stała dla wszystkich eksperymentów.

W eksperymencie 5.1.3 zostało stwierdzone, że odległości w pionie pomiędzy kolejnymi punktami wyznaczającymi trasę powinna wynosić 2-4 m. Zatem przyjmijmy, że będą to 3 m, co dla wyżej podanego slalomu daje nam niecałe 30 punktów do optymalizacji.

Wartości parametrów algorytmu genetycznego są takie jak wymienione wcześniej w rozdziale 4.4.1. Korzystamy z algorytmu $(\mu + \lambda)$, gdzie $\mu = 30$, a $\lambda = 100$. W każdej iteracji najpierw krzyżujemy osobniki z populacji tymczasowej, losując pary rodzicielskie, a w wyniku krzyżowania powstaje znów 100 osobników. Następnie mutacji podlegają wszystkie z tych osobników. Jedynym zastrzeżeniem jest nie zmienianie pozycji punktów przejazdu przez bramki - chcemy, aby narciarz przejechał zawsze jak najbliżej bramki. Jeśli któraś z wartości lub zastosowanych algorytmów zmienia się, jest to zaznaczone w eksperymencie.

#### Warunek zakończenia

Warunkiem stopu w zastosowanym algorytmie genetycznym jest sprawdzenie czy populacja, na której działamy przestała być już zróżnicowana. Dodatkowo, jeśli różnica pomiędzy najlepszym, a najgorszym osobnikiem jest mała, czekamy przez określoną liczbę iteracji, aby sprawdzić czy uzyskujemy jeszcze jakąś poprawę najlepszego osobnika, ale większą niż zadana granica. Jeśli taka sytuacja nie zachodzi, oznacza to, że szanse na poprawę wyniku są niewielkie i można zakończyć działanie algorytmu. W rozwiązaniu tym istnieją jednak trzy parametry, których wpływ na ostateczny wynik zbadamy w tym eksperymencie.

**Eksperyment 1.** W pierwszym eksperymencie parametry zakończenia algorytmu wynoszą:

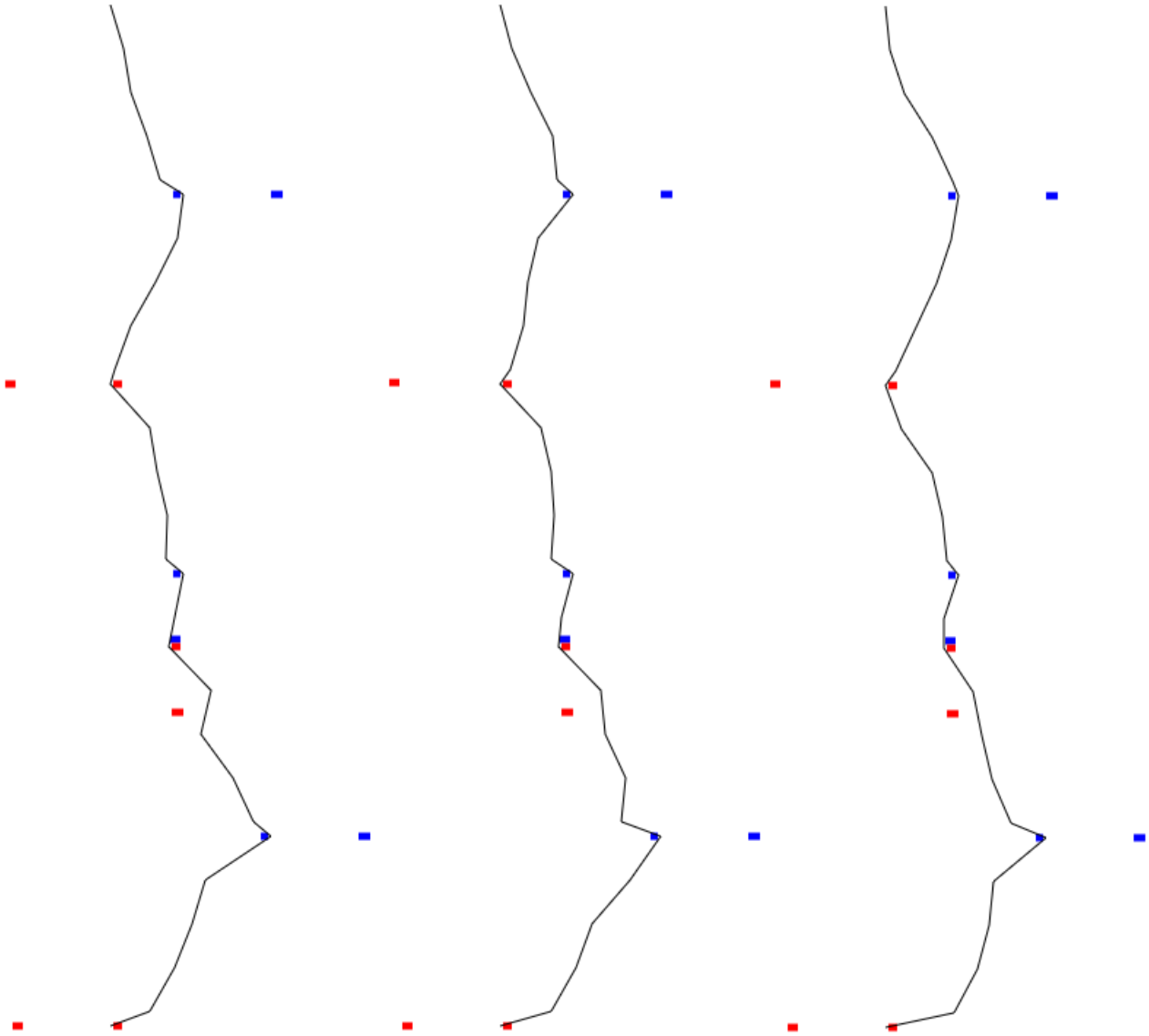
- dla zróżnicowania populacji przyjmujemy na początek wartość 0.1, czyli 10%, gdzie wartość ta obliczana jest na podstawie najlepszej i najgorszej wartości fitness w populacji, a następnie dzielona przez wartość najlepszą (aby uzyskać wartość w % mnożymy przez 100%)

- zmiana najlepszego osobnika jest uznawana za wystarczająco dużą jeśli wyniesie przynajmniej 0.3 s

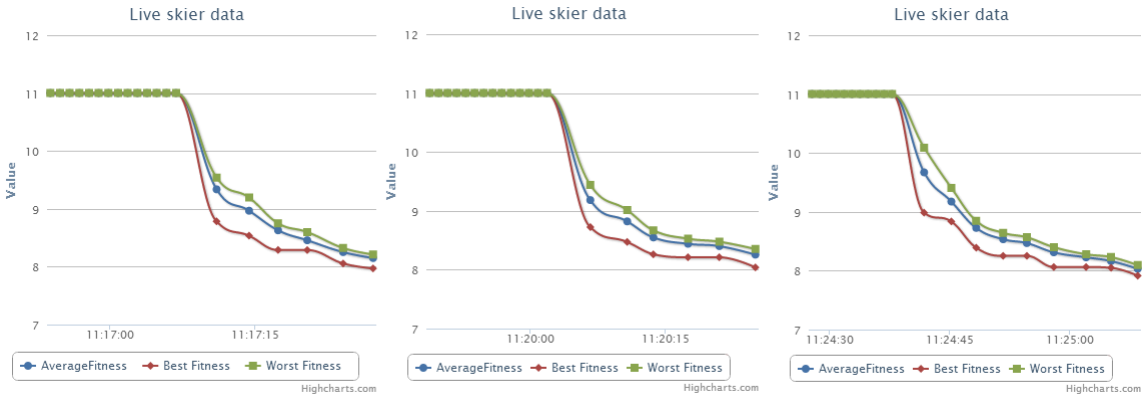
- ilość iteracji przez które czekamy na zmianę najlepszego osobnika to na początek 7

W związku z tym, że zachowanie algorytmu jest niedeterministyczne, dla każdego zestawu parametrów zostały przeprowadzone po trzy eksperymenty, aby można było uśrednić wyniki i porównać eksperymenty między sobą.

Poniżej znajdują się wyniki wizualne przeprowadzonego eksperymentu. Na kolejnych obrazkach widoczne są odpowiednio wszystkie trzy znalezione najlepsze trasy oraz wykresy zmian wartości fitness odpowiednio: najlepszego osobnika w każdej iteracji (kolor czerwony), najgorszego osobnika (kolor zielony) oraz średnia wartość fitness całej populacji. Kolejność slalomów odpowiada kolejności wykresów.



Rysunek 5.4: Wyniki eksperymentu dla parametrów zakończenia: 10%, 0.3 s i 7




Rysunek 5.5: Wyniki eksperymentu dla parametrów zakończenia: 10%, 0.3 s i 7

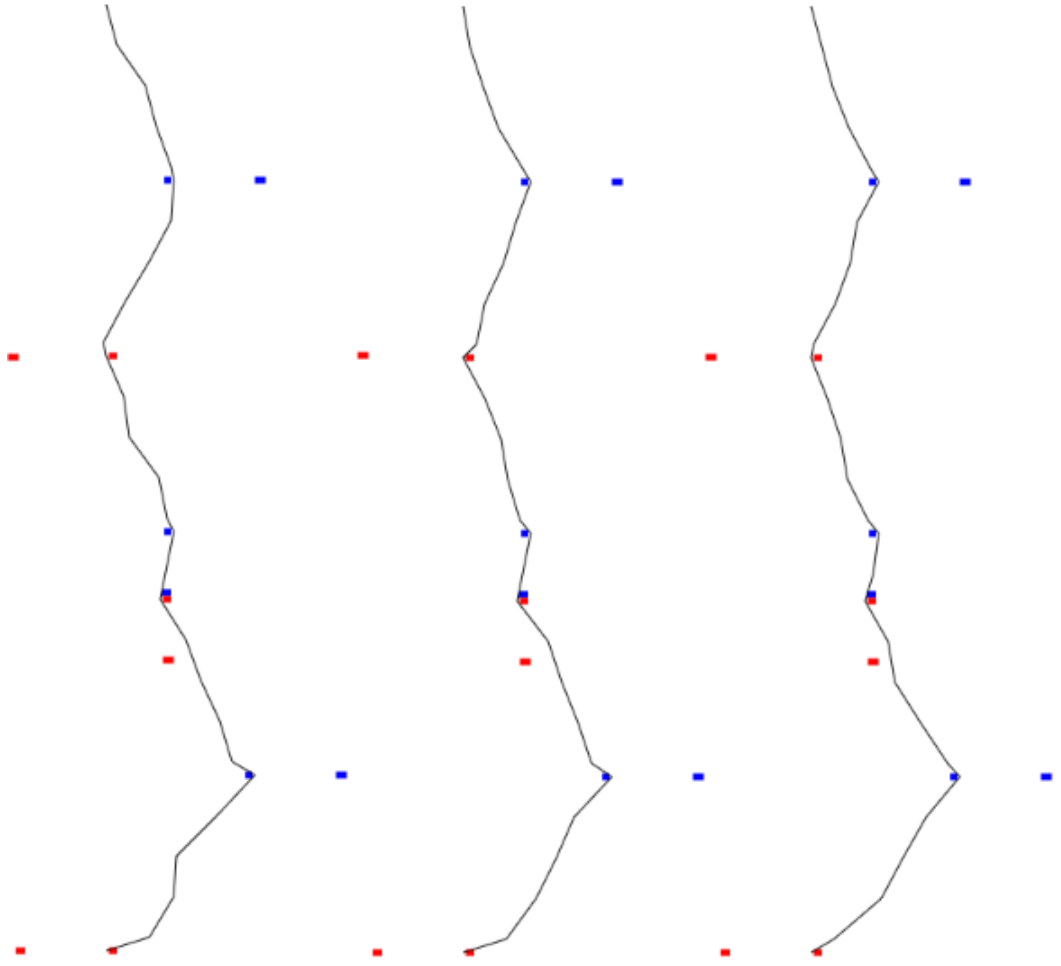
Oto otrzymane czasy przejazdu dla kolejnych prób:

| numer próby | czas przejazdu [s] | ilość iteracji | czas działania |
|---|---|---|---|
| 1 | 7.968 | 7 | 18 |
| 2 | 8.039 | 7 | 19 |
| 3 | 7.917 | 9 | 21 |

Zatem średni czas przejazdu w tych trzech próbach wyniósł 7.975.
Ilość iteracji w przypadku tych wyników wyniosła 7-9. Algorytm bardzo szybko dochodzi do rozwiązania, a czas działania to tylko ok. 20 s. Trzeba jednak pamiętać, że algorytm musi działać wystarczająco długo, aby miało sens przesyłanie problemu z serwera - najlepiej około kilku minut. Wtedy czas obliczeń będzie znacząco dłuższy w stosunku do czasu przesyłu danych.

Jednocześnie, jak widać wyraźnie na obrazkach, wyniki nie są satysfakcjonujące - trasa przejazdu nie jest gładka i z pewnością istnieje lepsze rozwiązanie. Jednak skoro zostało wykonanych tylko kilka iteracji algorytmu, oznacza to, że można spróbować uzyskać lepsze wyniki poprzez wydłużenie czasu działania algorytmu.

**Eksperyment 2.** Wprowadzamy teraz zmianę w wartości parametru oznaczającego różnicę w fitness najlepszego osobnika. Zamiast wartości 0.3 s teraz będzie to 0.1 s. Sprawdzimy zarówno czy czas działania algorytmu się wydłuży jak i czy uda się przez to otrzymać lepszy wynik.



Rysunek 5.6: Wyniki eksperymentu dla parametrów zakończenia: 10%, 0.1 s i 7

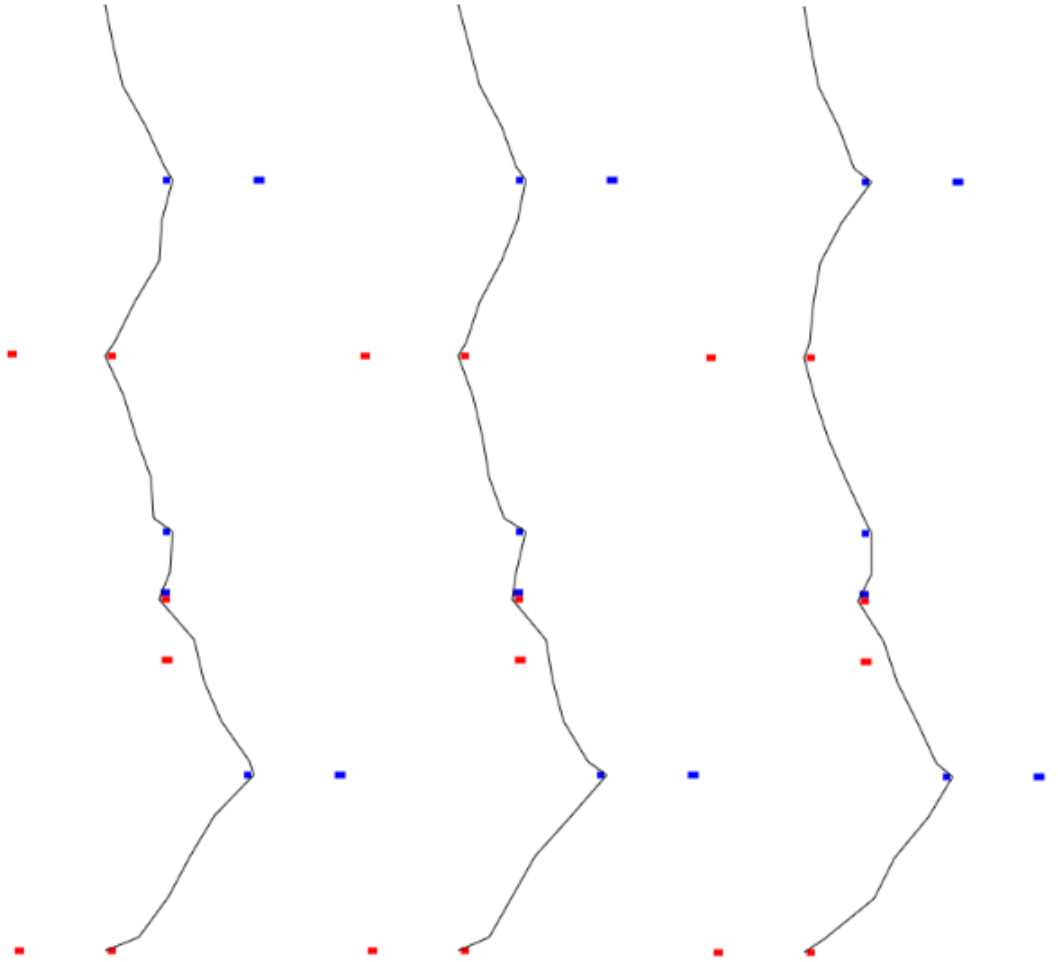

Rysunek 5.7: Wyniki eksperymentu dla parametrów zakończenia: 10%, 0.1 s i 7

| numer próby | czas przejazdu [s] | ilość iteracji | czas działania [s] |
|---|---|---|---|
| 1 | 7.916 | 15 | 49 |
| 2 | 7.809 | 14 | 46 |
| 3 | 7.777 | 16 | 51 |

Średnia uzyskanych czasów wynosi: 7.834.

Wyniki poprawiły się, linia przejazdu jest dużo gładsza, a czasy poprawiły się o ok. 0.15 s, czyli o ok. kilka procent. Jednak czas działania algorytmu wydłużył się do 45-50 s, a ilość iteracji wyniosła 14-16. Oczywiście czas ten jest wciąż na tyle mały, że możemy wprowadzać dalsze zmiany, które wydłużą czas działania i jednocześnie umożliwią znalezienie lepszego rozwiązania.

**Eksperyment 3.** Spróbujmy w takim razie zmienić również parametr dotyczący zróżnicowania populacji - niech będzie to teraz nie 10%, a 1%.



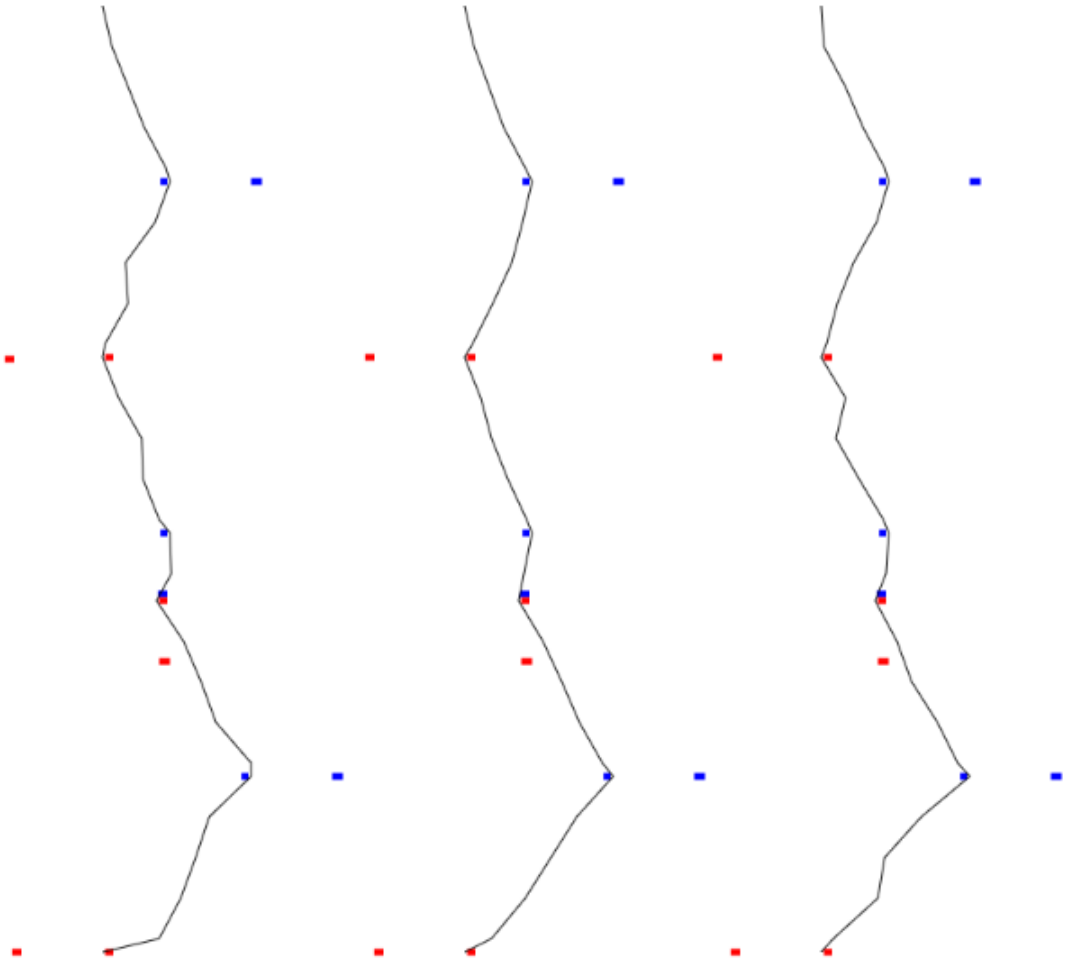
Rysunek 5.8: Wyniki eksperymentu dla parametrów zakończenia: 1%, 0.1 s i 7


Rysunek 5.9: Wyniki eksperymentu dla parametrów zakończenia: 1%, 0.1 s i 7

| numer próby | czas przejazdu [s] | ilość iteracji | czas działania [s] |
|---|---|---|---|
| 1 | 7.818 | 17 | 55 |
| 2 | 7.822 | 14 | 45 |
| 3 | 7.817 | 14 | 47 |

Średnia zmierzonych wyników wynosi: 7.819.
Jak widać wyniki są niewiele lepsze niż te poprzednie. Zatem ten warunek praktycznie nie zmienił zachowania. Pozostał natomiast jeszcze warunek ilości iteracji przez które sprawdzamy czy poprawia się fitness najlepszego osobnika.

**Eksperyment 4.** Zmieniamy teraz wartość ilości iteracji z 7 na 10.



Rysunek 5.10: Wyniki eksperymentu dla parametrów zakończenia: 1%, 0.1 s i 10


Rysunek 5.11: Wyniki eksperymentu dla parametrów zakończenia: 1%, 0.1 s i 10

| numer próby | czas przejazdu [s] | ilość iteracji | czas działania [s] |
|---|---|---|---|
| 1 | 7.892 | 13 | 41 |
| 2 | 7.743 | 19 | 60 |
| 3 | 7.837 | 13 | 41 |

Średnia wyników wynosi: 7.824. Jak widać, również zmiana tego parametru nie wnosi poprawy, choć udało się otrzymać dotychczas najlepszy wynik - 7.743. Ponieważ algorytm ten jest niedeterministyczny trudno ocenić jakie wartości parametrów są optymalne. Oczywiście można dalej zmniejszać ich wartości w poszukiwaniu lepszego wyniku.

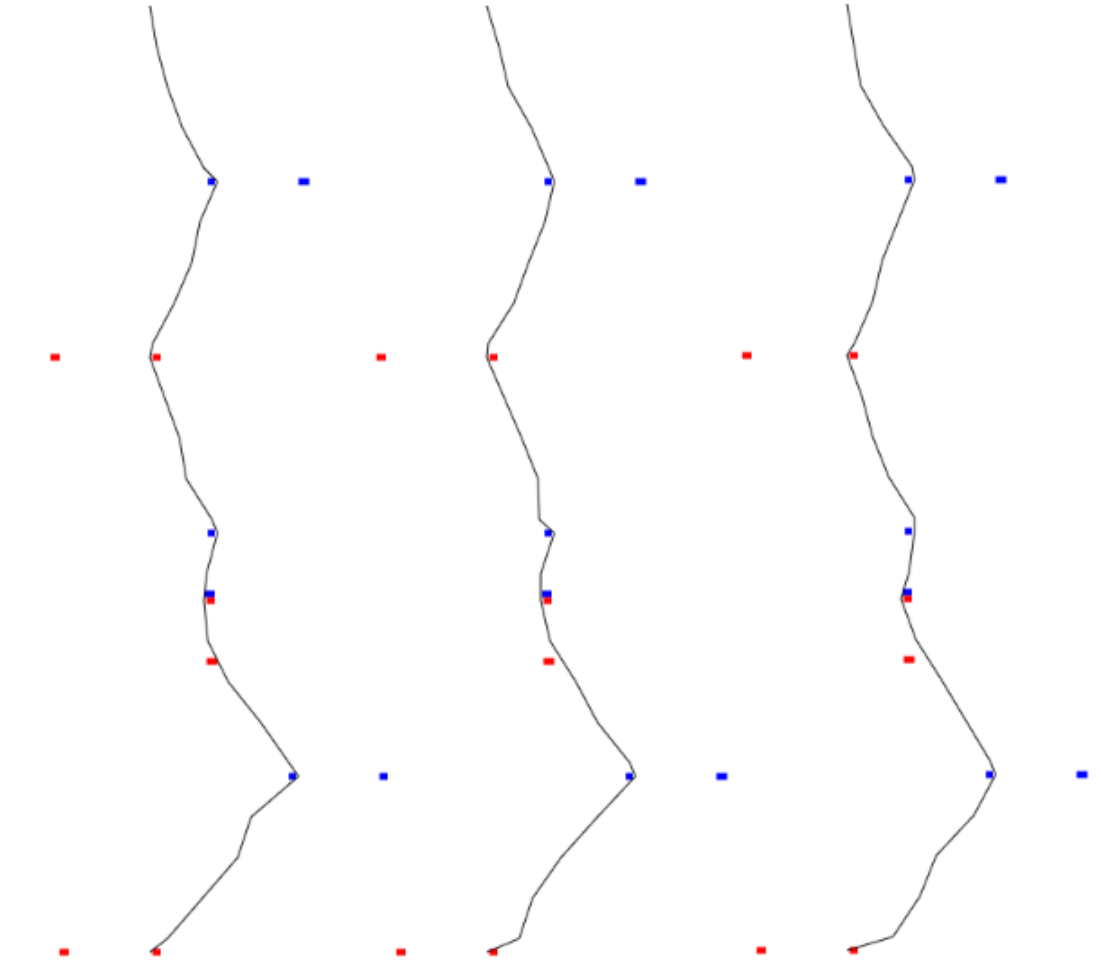
Drugą opcją, która pomogłaby rozwiązać problem zbyt szybkiej zbieżności populacji byłoby wprowadzenie w chwili pojawienia się takiej sytuacji nowych osobników, aby wprowadzić większą różnorodność i umożliwić dalsze poszukiwania. Jednak w trakcie szczegółowej analizy wyników z kolejnych iteracji (widocznych na wykresach), łatwo zauważyć, że na początku poprawa jest znacząca z każdą iteracją, a pod koniec zmiany są niewielkie. Można zatem pomyśleć o wprowadzeniu innego algorytmu kiedy zmiany są już słabo widoczne, który skuteczniej znajdzie lepsze rozwiązanie. Wprowadzeniem takiego rozwiązania jest zastosowanie lokalnej optymalizacji. Wyniki tego podejścia opisane są w sekcji 5.2.2.

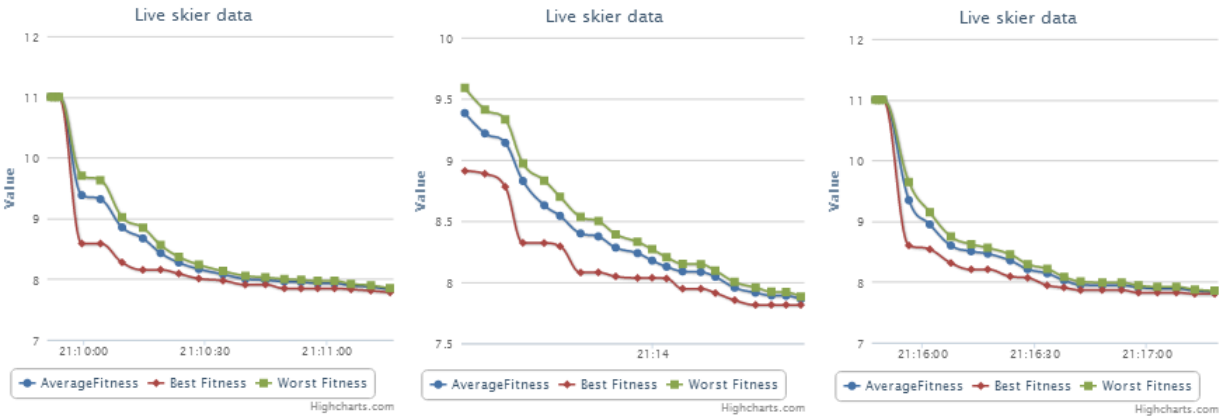
Zanim przejdziemy jednak do wyników wprowadzenia lokalnej optymalizacji pozostaje jeszcze sprawdzić ostatnie parametry algorytmu genetycznego: $\mu$ i $\lambda$.

#### Parametry $\mu$ i $\lambda$

Do tej pory początkowa populacja liczyła 30 osobników, a populacja tymczasowa - 100.

**Eksperyment 1.** Sprawdzimy teraz jak wpływa na wyniki zmiana tych parametrów, na początek spróbujemy dla wartości odpowiednio 50 i 150.



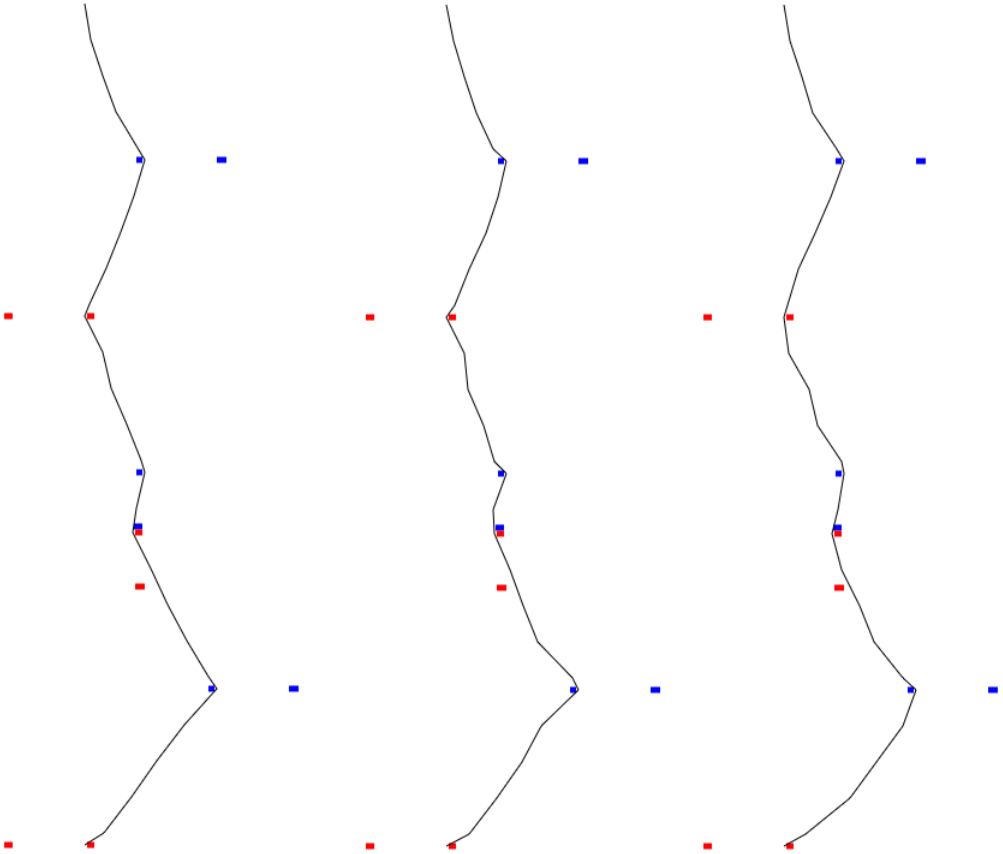


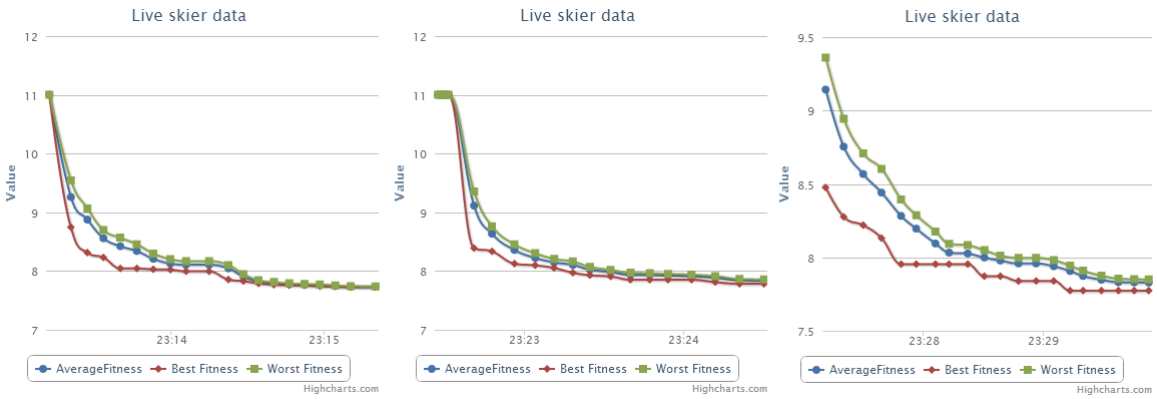

| numer próby | czas przejazdu [s] | ilość iteracji | czas działania [s] |
|---|---|---|---|
| 1 | 7.780 | 17 | 81 |
| 2 | 7.814 | 21 | 109 |
| 3 | 7.804 | 17 | 88 |

Średnia wyników wynosi: 7.802. Można się było spodziewać poprawy wyników, jako, że rozpatrujemy naraz więcej osobników przeszukując większą przestrzeń rozwiązań, a więc zwiększając prawdopodobieństwo znalezienia lepszego rozwiązania. Jednak poprawa jest niewielka, ale ważne jest to, że do tej pory nie udało nam się uzyskać tak dobrego średniego wyniku. Pokazuje to, że udało nam się doprowadzić do sytuacji, w której algorytm

daje rzeczywiście częściej lepsze wyniki, a uzyskane czasy są zbliżone do siebie. Takie wyniki zostały poniesione kosztem czasu wykonania algorytmu, który wzrósł do ponad 80 s, a w jednym przypadku nawet 110.

**Eksperyment 2.** Zobaczmy jak zmienią się wyniki jeśli wartości $\mu$ i $\lambda$ wyniosą odpowiednio 60 i 200.



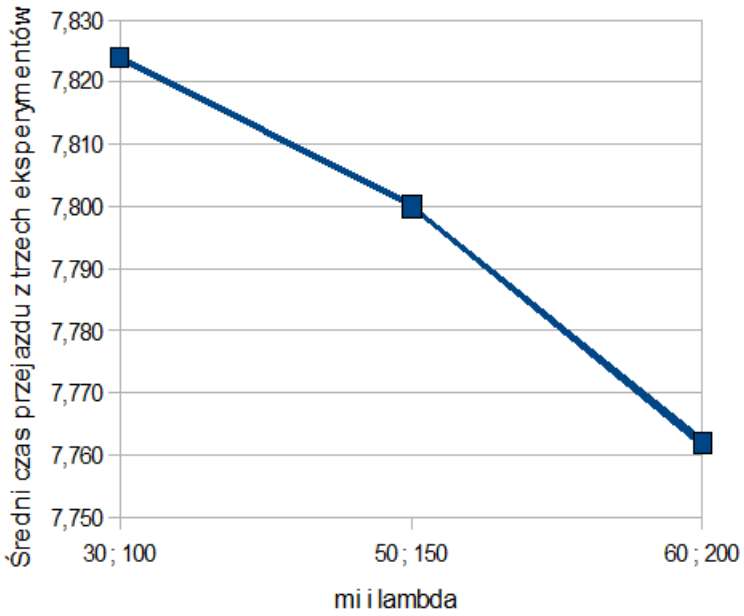



| numer próby | czas przejazdu [s] | ilość iteracji | czas działania [s] |
|---|---|---|---|
| 1 | 7.729 | 19 | 127 |
| 2 | 7.783 | 15 | 118 |
| 3 | 7.774 | 20 | 170 |

Średnia wyników wynosi: 7.762. W tym przypadku wszystkie otrzymane czasy są mniejsze niż 7,8 s, a otrzymane linie przejazdu wyglądają bardzo dobrze. Warto również zauważyć, że w pierwszym przypadku otrzymali-

śmy najlepszy czas - 7,729 s. Widzimy również, że zgodnie z oczekiwaniami, wydłużył się znów czas działania algorytmu. Minimalny czas to 118 s, czyli około 2 minuty, a maksymalnie wykonanie trwało prawie 3 minuty.

Dla porównania przeanalizujmy jak wpłynęły zmiany parametrów $\mu$ i $\lambda$ na uzyskiwane wyniki.




Powyższy wykres pokazuje jak zmienił się średni czas uzyskanych wyników odpowiednio dla kolejnych wartości parametrów $\mu$ i $\lambda$. Widać, że choć poprawa jest niewielka, bo o około 0,05 s, to jednak widoczny jest trend wskazujący, że zwiększenie populacji wpływa pozytywnie na znajdowane rozwiązanie.

**Gęstość siatki**

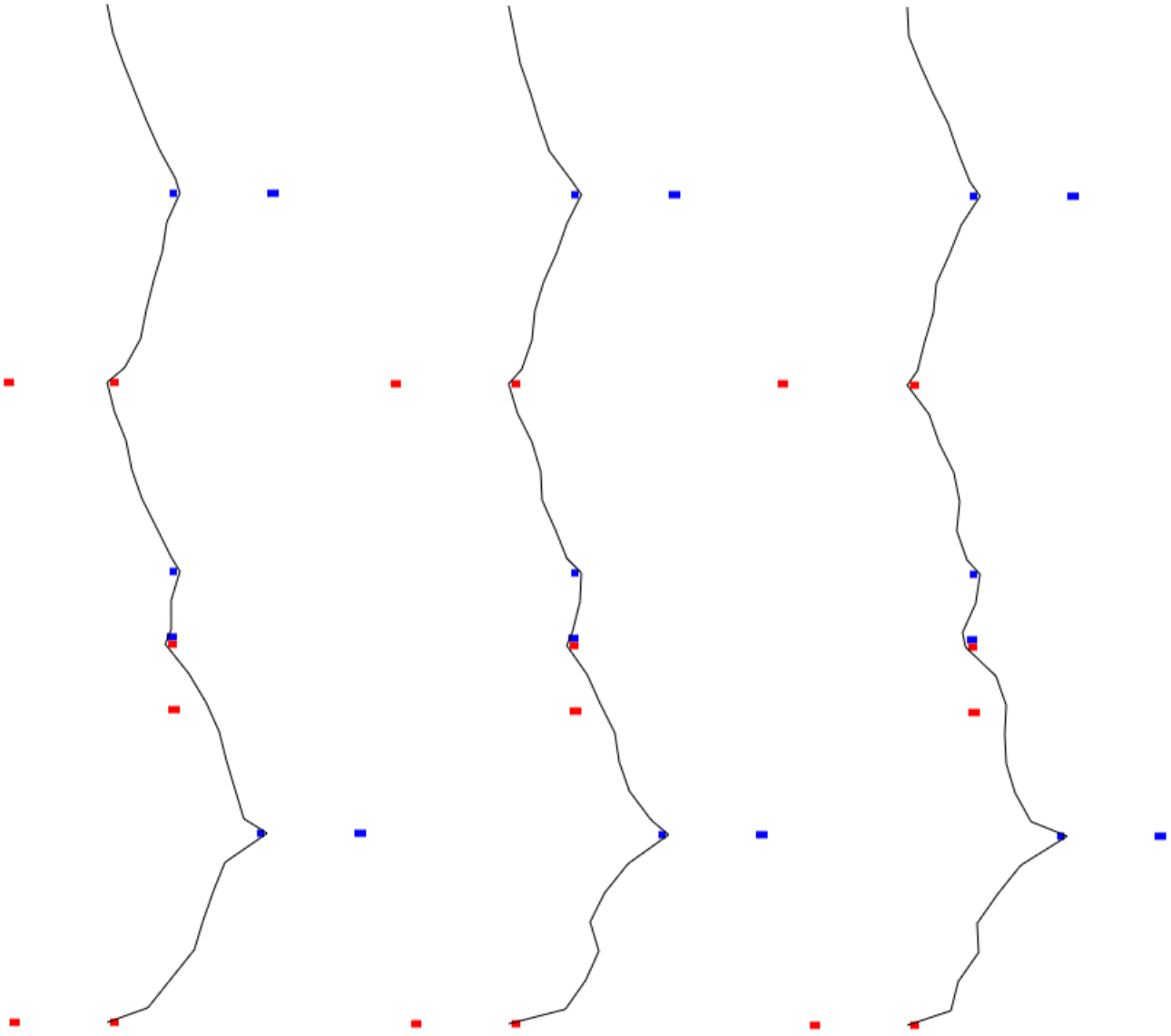
Aby uzyskać dokładniejsze rozwiązania zawsze można spróbować zagęścić poziome linie wyznaczające ilość punktów stanowiących reprezentację trasy. Do tej pory linie te były w odległości 3 m. Po zmianie na 2 m czas działania algorytmu powinien się znacznie wydłużyć ze względu na większą ilość rozpatrywanych punktów. W eksperymencie wartości parametrów wynoszą dla warunku zakończenia:

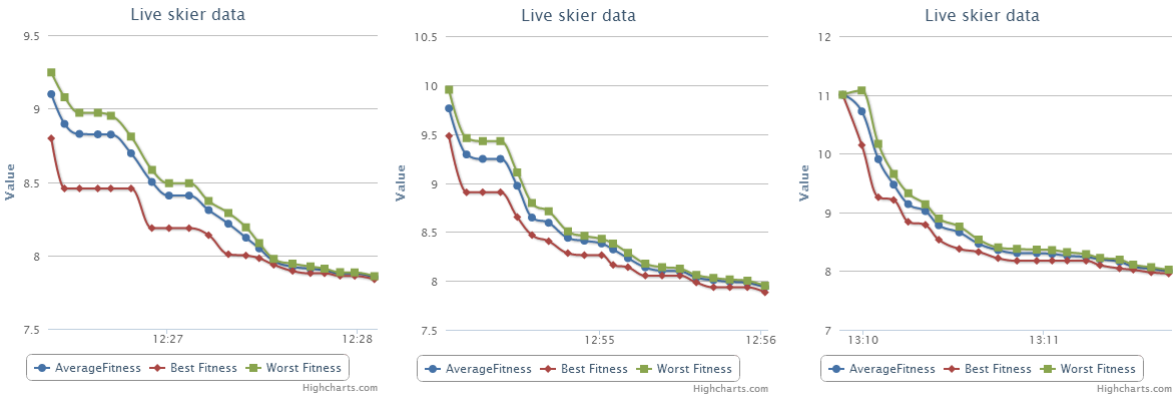
– dla zróżnicowania populacji przyjmujemy na wartość 0.01, czyli 1%

– zmiana najlepszego osobnika jest uznawana za wystarczająco dużą jeśli wyniesie przynajmniej 0.05 s

– ilość iteracji przez które czekamy na zmianę najlepszego osobnika - 10

Natomiast parametry $\mu$ i $\lambda$ przyjmijmy 30 i 100.






Czas wykonania wydłużył się do 100-150 s., a ilość iteracji zwiększyła się do ok. 20. Mimo zwiększenia dokładności, wciąż istnieją na trasie zbyt ostre zmiany kierunku, które w rzeczywistości są niemożliwe do wykonania bez dodatkowego czasu oraz zmniejszenia prędkości. Z tego powodu obliczona trasa jest miejscami zbyt kanciasta. Wychodzi zatem konieczność wprowadzenia mechanizmu, które eliminuje zbyt ostre zmiany kierunku np.

poprzez uwzględnienie konieczności zmniejszenia prędkości w takich miejscach. Szczegółowo, mechanizm ten opisany jest w sekcji 4.5 na stronie 34.

### Długa trasa

Dotychczas rozpatrywany slalom był jedynie fragmentem standardowego slalomu, czas przejazdu wynosił mniej niż 8 s. Pozostaje pytanie, czy dla większej ilości bramek algorytm nadal będzie w stanie znaleźć rozwiązanie zbliżone do optymalnego oraz jak dużo czasu będzie potrzebne do jego znalezienia. Możliwe też, że algorytm będzie potrzebował zmiany parametrów, aby znaleźć rozwiązanie.

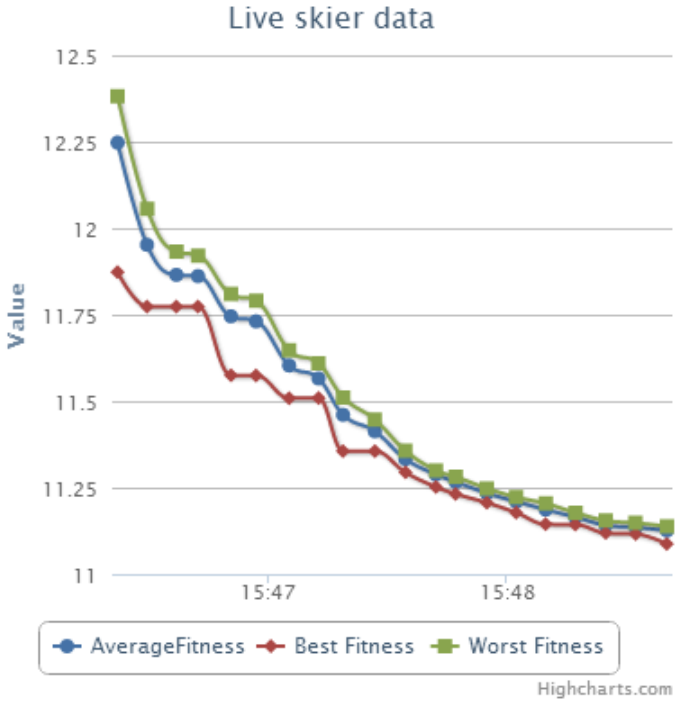
Sprawdzone zostanie teraz jakie wyniki otrzymamy dla dłuższego slalomu: (19,10),(11,30),(20,50),(12,65),(0,80),(10,100),(3,120),(6,135)). Parametry $\mu$ i $\lambda$ pozostają ustawione na 30 i 100. Parametry zatrzymania algorytmu zostawiamy, aby algorytm działał dłużej:

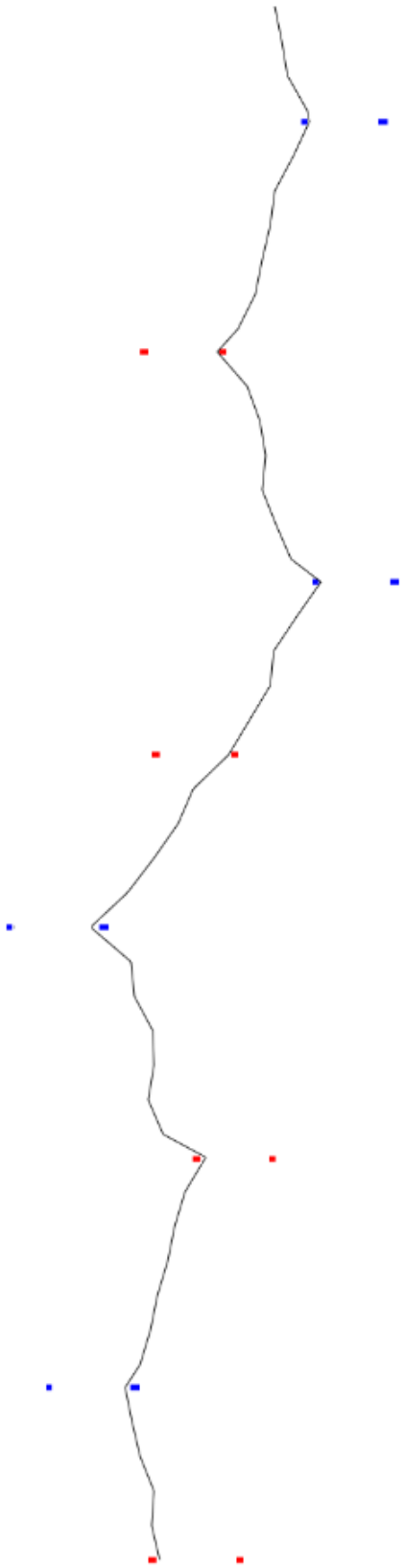
– dla zróżnicowania populacji przyjmujemy na wartość 0.01, czyli 1%

– zmiana najlepszego osobnika jest uznawana za wystarczająco dużą jeśli wyniesie przynajmniej 0.05 s

– ilość iteracji przez które czekamy na zmianę najlepszego osobnika - 10

Kolejne punkty rozwiązania znajdują się w odległości 3m w pionie, co dla tej trasy daje 48 punktów do optymalizacji.



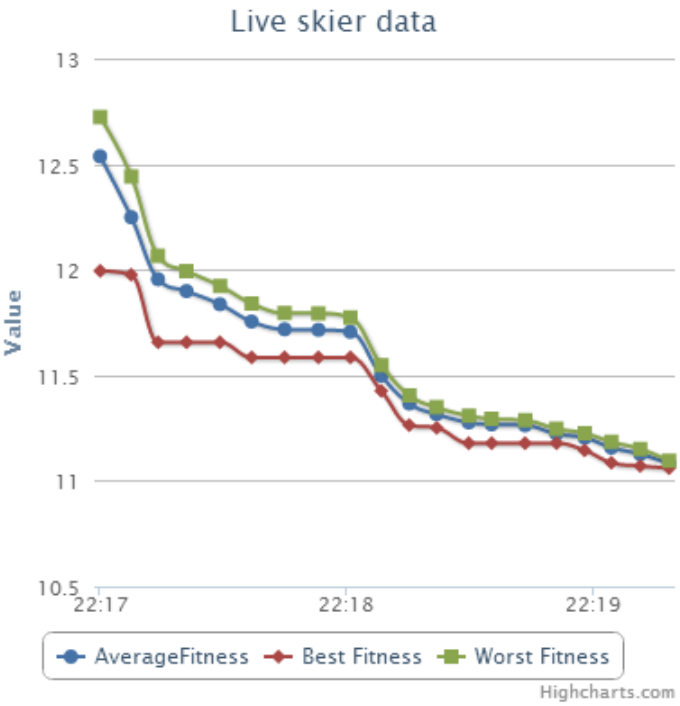



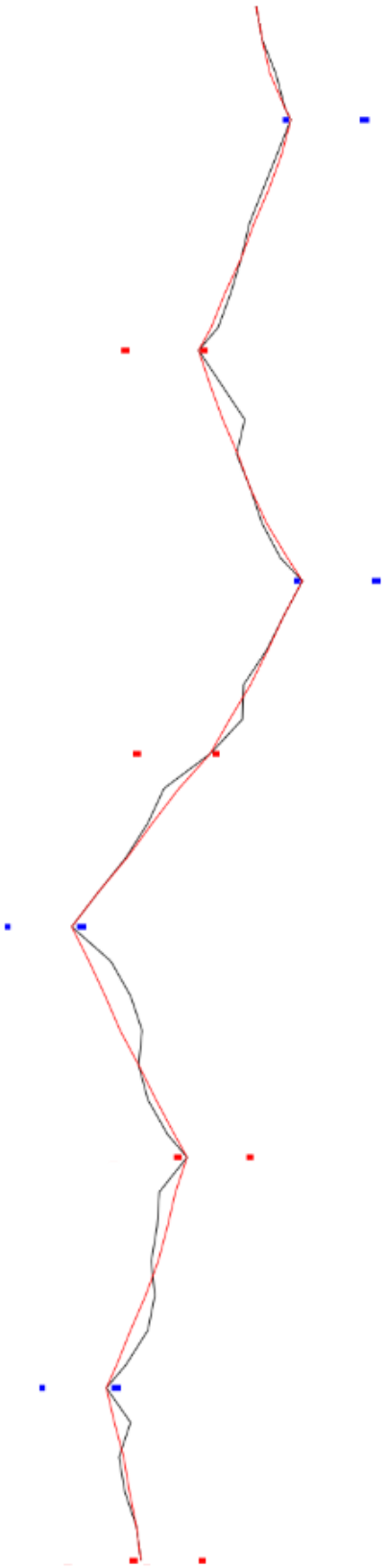
Otrzymany wynik to 11.087 s. Obliczenia trwały 170 s., a algorytm potrzebował 23 iteracje do zakończenia obliczeń. Znów można powiedzieć, że wynik nie jest wynikiem złym, jednak pozostawia wiele do życzenia - istnieje wiele ostrych zakrętów, zwłaszcza w okolicach bramek, rozwiązanie nie jest idealne.

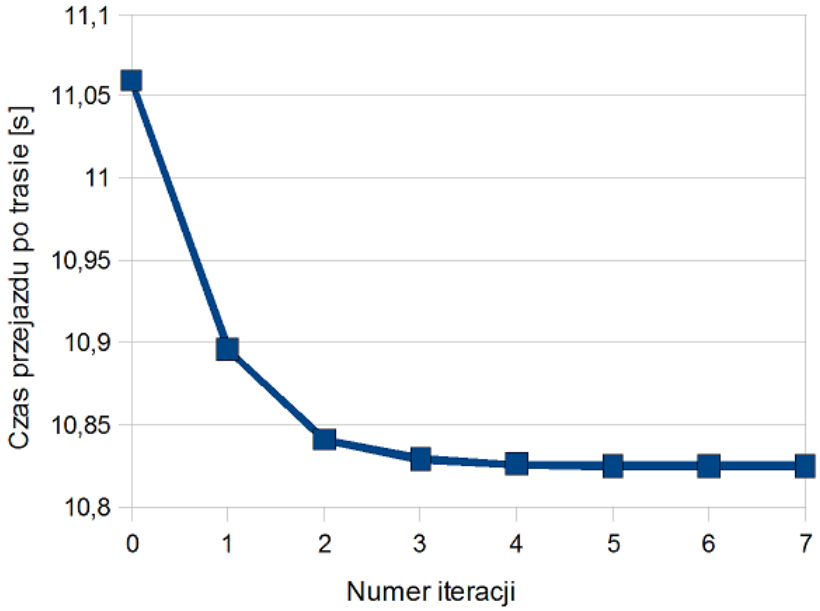
### 5.2.2. Lokalna optymalizacja

Ponieważ algorytm genetyczny miał problem ze znalezieniem lepszych rozwiązań zwłaszcza kiedy dotychczas znalezione rozwiązanie dawało już dosyć dobry wynik, ale wciąż istniały miejsca, które gołym okiem można było zobaczyć jako wpływające negatywnie na jakość wyniku. Skoro algorytm nie radził sobie będąc blisko rozwiązania, można spróbować dodać jakiś rodzaj algorytmu optymalizacji lokalnej, bazując na rozwiązaniu znalezionym przez algorytm genetyczny. Mamy wtedy pewność, że znajdziemy rozwiązanie nie gorsze od tego, na którym bazujemy i dodatkowo, że będzie ono najlepszym lokalnym rozwiązaniem. Możemy mieć także nadzieję, że będzie to również rozwiązanie globalnie optymalne, ale pewności takiej mieć nie możemy.

**Eksperyment 1.** Spróbujmy zatem, bazując na eksperymencie "Długa trasa"w sekcji 5.2.1 porównać jak poprawi się otrzymany wynik dodając na koniec obliczeń optymalizację lokalną algorytmem Hill climbing. Wszystkie parametry oraz trasa przejazdu pozostają takie same jak w eksperymencie, do którego się odwołujemy.


Live skier data
13
12.5
12
11.5
11
10.5
Value
22:17
22:18
22:19
AverageFitness
Best Fitness
Worst Fitness
Highcharts.com






Na wykresie 5.2.2 wciąż znajdują się zmiany wyłącznie dotyczące algorytmu genetycznego. Czarna linia trasy jest rezultatem otrzymanym po działaniu samego algorytmu genetycznego, czas przejazdu po takiej trasie wynosi 11.060 s., a więc jest to czas zbliżony do otrzymanego w eksperymencie bazowym. Algorytm genetyczny działał przez 21 iteracji. Czerwona linia stanowi trasę uzyskaną po zaaplikowaniu algorytmu optymalizacji lokalnej na trasie znalezionej przez algorytm genetyczny. Na wykresie 5.2.2 możemy zobaczyć jak dokładnie zmieniał się czas przejazdu w każdej iteracji algorytmu lokalnego. Iteracja numer 0 to wynik otrzymany przez algorytm genetyczny. Po 7 iteracjach otrzymaliśmy wynik 10.825 s., co stanowi poprawę o 0.235 s., czyli ok. 2%. Poprawa ta jest zdecydowanie zauważalna także w kształcie slalomu i na pewno na tyle znacząca, że decyduje o zwycięstwie zawodnika.

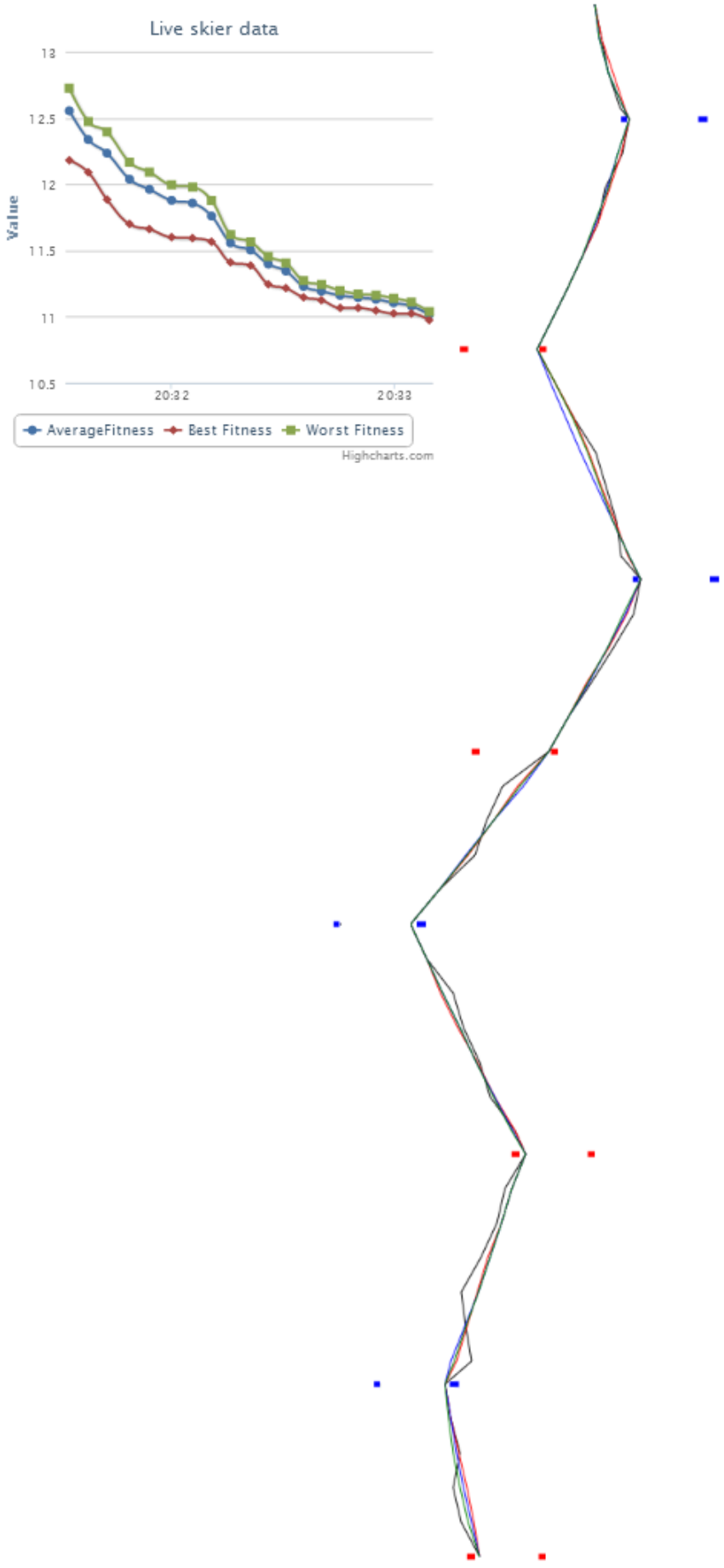
**Eksperyment 2.** W eksperymencie pierwszym sprawdziliśmy działanie algorytmu lokalnego dla domyślnie ustawionych parametrów. W zastosowanym algorytmie Hill Climbing mamy możliwość zmiany dwóch parametrów:

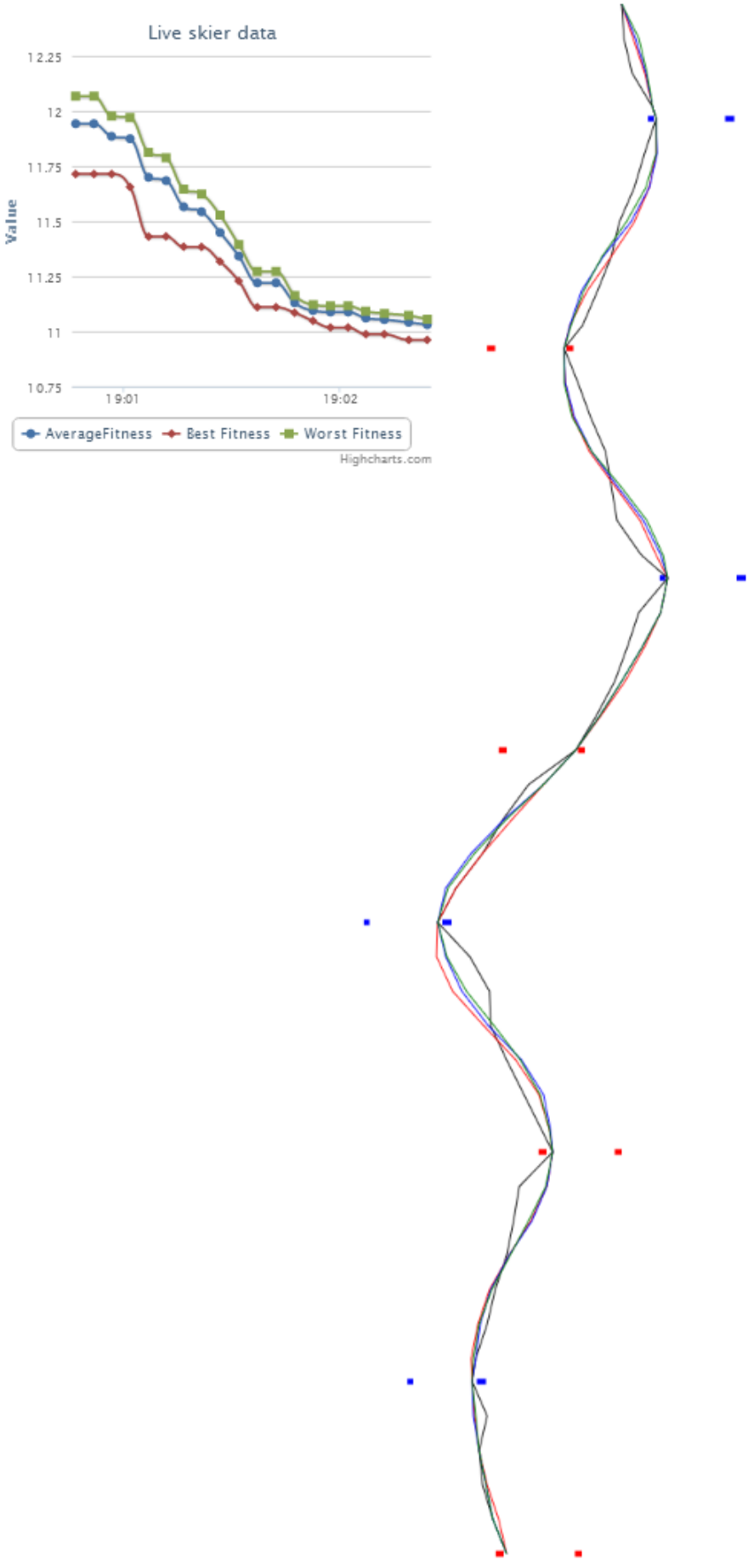
– rozmiar kroku

– tzw. przyspieszenie (ang. *acceleration*)

Szczegółowy opis parametrów można znaleźć w podrozdziale 2.4.2. W domyślnym ustawieniu wartości tych parametrów wynoszą odpowiednio 0.5 (odpowiadająca 0.5 metrom na rzeczywistej trasie) oraz 1.2 dla przyspieszenia. W przeprowadzonym eksperymencie pokażemy jak te parametry wpływają na ostateczne rozwiązanie. Na tym samym rezultacie działania algorytmu genetycznego uruchomimy trzy razy algorytm lokalnej optymalizacji, ale z następującymi zestawami parametrów:

| nr próby | wielkość kroku [m] | przyspieszenie |
|---|---|---|
| 1 | 0.5 | 1.2 |
| 2 | 0.3 | 1.2 |
| 3 | 0.3 | 1.1 |


Live skier data
13
12.5
12
11.5
11
10.5
Value
20:32
20:33
AverageFitness
Best Fitness
Worst Fitness
Highcharts.com


Live skier data
12.25
12
11.75
11.5
11.25
11
10.75
Value
19:01
19:02
AverageFitness
Best Fitness
Worst Fitness
Highcharts.com

Powyższe wykresy pokazują otrzymane trasy dla poszczególnych instancji algorytmów:

– trasa czarna - trasa uzyskana w wyniki działania algorytmu genetycznego

– trasa czerwona - optymalizacja lokalna z parametrami domyślnymi

– trasa niebieska - optymalizacja lokalna z krokiem 0.3 m

– trasa zielona - optymalizacja lokalna z krokiem 0.3 m oraz przyspieszeniem 1.1

Trasy otrzymane w wyniku działania algorytmu optymalizacji lokalnej różnią się miejscami od siebie, choć w niewielkim stopniu. Niewielkie zróżnicowanie wynika z tego, że początkowa trasa, którą należało zoptymalizować była w każdym przypadku taka sama. Algorytm Hill Climbing jest algorytmem deterministycznym, ale zmiana parametrów tego algorytmu wpływa na otrzymywany wynik końcowy. Jednak w przypadku pierwszym zmiany są minimalne, a otrzymywany wynik tak naprawdę można sprowadzić do jazdy od bramki do bramki po liniach prostych. W drugim teście wyniki różnią się zwłaszcza w okolicach bramek, gdzie podejmowane są decyzje o wcześniejszym lub późniejszym skręcie - to tam można najwięcej zyskać lub stracić.

Wyniki:

eksperyment 1. bez karania

Czas działania programu: 181 s.

Czas uzyskany w wyniku działania algorytmu genetycznego: 10.974

| nr | otrzymany czas przejazdu [s] | ilość iteracji |
|---|---|---|
| 1 | 10.829 | 6 |
| 2 | 10.819 | 8 |
| 3 | 10.824 | 8 |

eksperyment 2. z karaniem

Czas działania programu: 253 s.

Czas uzyskany w wyniku działania algorytmu genetycznego: 10.895

| nr | otrzymany czas przejazdu [s] | ilość iteracji |
|---|---|---|
| 1 | 11.036 | 11 |
| 2 | 11.050 | 14 |
| 3 | 11.024 | 10 |

W pierwszej części eksperymentu najlepszy czas otrzymany został dla drugiego przypadku, a więc przy zmniejszeniu wyłącznie kroku. Poprawa w stosunku do pozostałych przypadków to 0.005-0.01 s czyli mniej niż 0.1%, ale widzimy, że nawet tak niewielkie zmiany na trasie przejazdu mogą wpływać na końcowy wynik.

Drugi przypadek jest trudniejszy do analizy, ponieważ wprowadzenie karania powoduje, że nie jesteśmy w stanie przenieść uzyskiwanych ocen do rzeczywistości. Według nich, znów najlepszy okazuje się być zestaw parametrów z drugiego przypadku, kiedy jednak sprawdzimy uzyskiwane czasy, widzimy, że są one gorsze niż najlepszy uzyskany wynik z algorytmu genetycznego, a w dodatku kolejność rezultatów nie pokrywa się z kolejnością ocen - najlepszy czas uzyskany został w ostatnim przypadku. Uzasadnieniem tego zachowania jest fakt uwzględniania w karaniu zbyt ostrych zakrętów, które fizycznie nie są możliwe do wykonania lub musiałyby zabrać dodatkowy czas. W rzeczywistości wyniki z pierwszego eksperymentu są niemożliwe do otrzymania - przy każdej bramce musielibyśmy stracić dodatkowy czas na przestawienie nart poprzez konieczność przyhamowania.

## 5.3. Architektura systemu

Aby zweryfikować funkcjonalność zaproponowanej architektury zaplanowany został szereg eksperymentów, mających na celu zbadać:

– wydajność poszczególnych przeglądarek dla intensywnych obliczeń (Chrome, Firefox, Internet Explorer) oraz znalezienie maksymalnej prędkości obliczeń na jednej maszynie, poprzez znalezienie optymalnej konfiguracji - ilości uruchomionych na raz kart przeglądarki

– skuteczność rekrutowania ochotników do obliczeń typu Volunteer Computing w przeglądarce, poprzez internetowe media społecznościowe

### 5.3.1. Wydajność przeglądarek dla intensywnych obliczeń

W celu zbadania wydajności silników *JavaScript* dla rozwiązywania naszego problemu został przeprowadzony następujący eksperyment.

**Niezmienne warunki przeprowadzanego eksperymentu**

– ta sama maszyna - Procesor Intel Core i5-2410M, 2.3 GHz, 8 GB pamięci RAM, System Windows 7 Home Premium

– bardzo zbliżone i zminimalizowane obciążenie procesora przez inne procesy

– obliczana jest ta sama instancja problemu (ta sama konfiguracja slalomu)

– dana przeglądarka ma uruchomioną tylko jedną instancję

– instancja uruchomionej przeglądarki ma otwartą tylko określoną liczbę zakładek

– wszystkie zakładki prowadzą obliczenia

**Zmienne warunki przeprowadzanego eksperymentu**

– model przeglądarki
  – Internet Explorer v.10
  – Chrome v.29
  – Firefox v.23

– docelowa liczba kart, w których równocześnie mają być prowadzone obliczenia (1-15)
  – 1
  – 3
  – 6
  – 9
  – 15

**Sposób przeprowadzania eksperymentu**

- manualnie, więc z około 1 sekundowym opóźnieniem na każdą zakładkę przeglądarki, uruchomione są obliczenia
- na konsoli przeglądarki wypisywany jest czas poświęcony na obliczenie kolejno znajdowanego rozwiązania
- notowane są wyniki, gdy sumaryczna liczba rozwiązań problemu ($A$) wynosi co najmniej 10
- sumowany jest czas poświęcony przez każdą z zakładek na obliczenia
- od wyniku każdej kolejnej zakładki odejmowana jest jedna sekunda, razy numer zakładki. Zakładki są numerowane od 0. Ma to na celu zminimalizowanie błędu wynikającego z opóźnienia uruchomienia obliczenia wynikającego z manualnego uruchamiania
- z powyższej sumy, wybierane jest maksimum - $T_{MAX}$. $T_{MAX}$ jest czasem w którym sumarycznie, przeglądarki uzyskały *A* wyników
- obliczane jest tempo obliczeń:
  $P = \frac{T_{MAX}}{A}[\frac{s}{rozwiazanie}]$

Tablica 5.1: Niezależnie uruchomione obliczenia na każdej z przeglądarek. W każdej z przeglądarek obliczenia prowadzone w jednej zakładce.

| | **Chrome** | **Firefox** | **IE** |
|---|---|---|---|
| Numer zakładki | 0 | 0 | 0 |
| **Czas rozwiązania kolejnego problemu w sekundach** | 50.67 | 351.528 | 125.749 |
| | 59.671 | 272.213 | 137.551 |
| | 48.118 | 209.443 | 153.626 |
| | 47.877 | 160.335 | 163.051 |
| | 64.928 | 208.956 | 178.417 |
| | 50.003 | 239.561 | 152.229 |
| | 44.629 | 189.09 | 124.884 |
| | 53.924 | 214.901 | 123.323 |
| | 80.939 | 193.573 | 105.364 |
| | 50.685 | 195.714 | 90.609 |
| **Suma czasów** | 551.444 | 2235.314 | 1354.803 |
| **Suma czasów z uwzględnieniem opóźnienia uruchomienia i zaokrągleniem** | 551 | 2235 | 1355 |
| $T_{max}[s]$ | 551 | 2235 | 1355 |
| **A - Ilość obliczonych rozwiązań** | 10 | 10 | 10 |
| **P - Tempo [s/rozwiązanie]** | 55.1 | 223.5 | 135.5 |

**Wnioski** Eksperyment wykazał, że wydajność obliczeń drastycznie różni się w zależności od modelu przeglądarki, a więc od użytego silnika JavaScript. Całościowe wyniki eksperymentu dobrze obrazuje wykres 5.12 na stronie 71. Najlepiej wypadła przeglądarka Chrome, ponad dwa razy wolniejsza okazała się najnowsza przeglądarka Internet Explorer, a ponad 4 razy gorsza przeglądarka Firefox. Ciekawe wyniki dotyczą wydajności dla

Tablica 5.2: Uruchomienie obliczeń w 6 zakładkach przeglądarki Chrome równocześnie.

| **Chrome** | | | | | | |
|---|---|---|---|---|---|---|
| Numer zakładki | 0 | 1 | 2 | 3 | 4 | 5 |
| **Czas rozwiązania kolejnego problemu w sekundach** | 122.138 | 131.19 | 211.114 | 106.006 | 227.966 | 133.511 |
| | 178.092 | 167.305 | 161.464 | 132.485 | 210.441 | 144.768 |
| | 128.211 | 108.657 | 128.156 | 96.603 | 190.033 | 204.7 |
| | 188.534 | 107.546 | 170.106 | 155.116 | 148.481 | 184.593 |
| **Suma czasów** | 616.975 | 514.698 | 670.84 | 670.84 | 490.21 | 667.572 |
| **Suma czasów z uwzględnieniem opóźnienia uruchomienia i zaokrągleniem** | 617 | 514 | 669 | 668 | 486 | 663 |
| $T_{max}[s]$ | 669 | | | | | |
| **A - Ilość obliczonych rozwiązań** | 24 | | | | | |
| **P - Tempo [s/rozwiązanie]** | 27.9 | | | | | |

Tablica 5.3: Uruchomienie obliczeń w 6 zakładkach przeglądarki Firefox równocześnie.

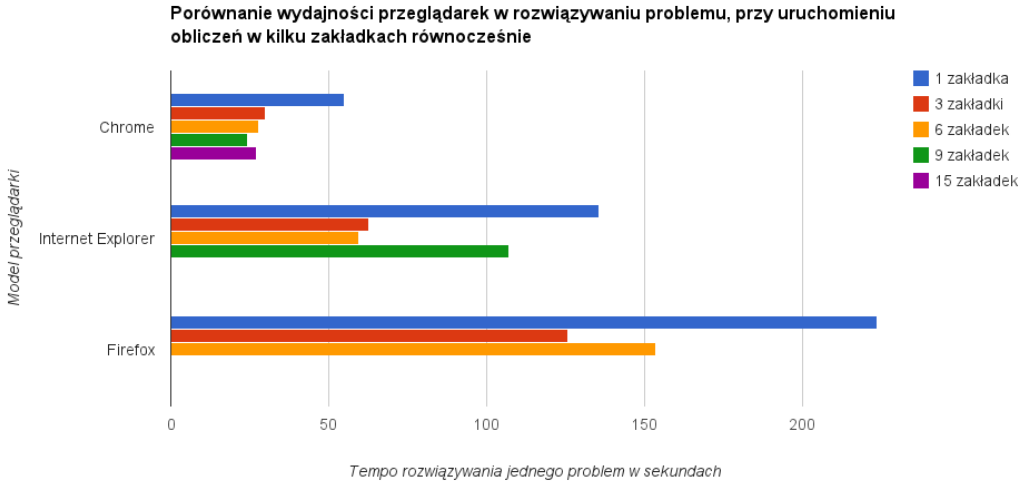
| **Firefox** | | | | | | |
|---|---|---|---|---|---|---|
| Numer zakładki | 0 | 1 | 2 | 3 | 4 | 5 |
| **Czas rozwiązania kolejnego problemu w sekundach** | 971.73 | 876.713 | 870.914 | 987.587 | 622.23 | 959.31 |
| | 707.105 | 1181.745 | 1167.395 | 987.172 | 787.954 | 716.807 |
| | 471.037 | | | | 608.404 | |
| **Suma czasów** | 2149.872 | 2058.458 | 2038.309 | 1974.759 | 2018.588 | 1676.117 |
| **Suma czasów z uwzględnieniem opóźnienia uruchomienia i zaokrągleniem** | 2150 | 2057 | 2036 | 1972 | 2015 | 1671 |
| $T_{max}[s]$ | 2150 | | | | | |
| **A - Ilość obliczonych rozwiązań** | 14 | | | | | |
| **P - Tempo [s/rozwiązanie]** | 153.6 | | | | | |

obliczeń prowadzonych w wielu kartach przeglądarki jednocześnie. Dla każdej z przeglądarek, zdecydowanie wydajniej jest uruchamiać obliczenia w dwóch zamiast w jednej zakładce jednocześnie. Równoległe obliczenia w dwóch oknach, prawie że nie wpływają na spowolnienie obliczeń w którymś z nich, przez co uzyskuje się prawie dwukrotny skok w tempie rozwiązywania problemów. Replikowanie obliczeń na kolejne zakładki nie przynosi już jednak istotnego przyspieszenia. W przypadku przeglądarki Chrome, najlepsze tempo uzyskuje się przy 9-14 otwartych równolegle zakładkach. Przy 15 otwartych instancjach prowadzących obliczenia, wyniki zaczynają się już pogarszać. W przypadku Internet Explorer'a, optymalna ilość zakładek, to między 6 a 8. Przy 9 zakładkach, tempo istotnie spadło - bo o prawie 80 procent, w stosunku do 8 okien. Najgorzej poradził sobie z równoległym przetwarzaniem Firefox. Najbardziej optymalną dla tego silnika konfiguracją są 4 zakładki dokonujące równolegle obliczeń. Już przy 5, 6 zakładkach całościowe tempo spada. Uruchomienie obliczeń w 9 zakładkach równocześnie okazało się na testowej maszynie niemożliwe, gdyż powodowało awaryjne zakończenie działania programy Firefox.

Ciekawym wnioskiem jest też fakt, że przeglądarki przydzielają podobne zasoby do obliczeń w tle, prowadzonych przez wiele kart równocześnie, i nie ma istotnego znaczenia, która z kart jest kartą otwartą. Obserwację tą ilustrują zrzuty ekranu 5.13 i 5.14 na stronie 72 z managera zadań systemu podczas prowadzenia obliczeń oraz

Tablica 5.4: Uruchomienie obliczeń w 6 zakładkach przeglądarki Internet Explorer równocześnie.

| **Internet Explorer** | | | | | | |
|---|---|---|---|---|---|---|
| Numer zakładki | 0 | 1 | 2 | 3 | 4 | 5 |
| **Czas rozwiązania kolejnego problemu w sekundach** | 465.152 | 186.834 | 154.918 | 247.763 | 486.031 | 147.32 |
| | 267.013 | 194.181 | 212.844 | 248.019 | 295.73 | 200.593 |
| | 260.614 | 207.847 | | 415.193 | 280.427 | 172.52 |
| | 260.386 | 245.773 | | 248.019 | | 231.295 |
| **Suma czasów** | 1253.165 | 834.635 | 367.762 | 1158.994 | 1062.188 | 751.728 |
| **Suma czasów z uwzględnieniem opóźnienia uruchomienia i zaokrągleniem** | 1253 | 834 | 366 | 1156 | 1058 | 747 |
| $T_{max}[s]$ | 1253 | | | | | |
| **A - Ilość obliczonych rozwiązań** | 21 | | | | | |
| **P - Tempo [s/rozwiązanie]** | 59.7 | | | | | |




Rysunek 5.12: Porównanie wydajności różnych modeli przeglądarek w rozwiązywaniu problemu, przy uruchomieniu obliczeń w kilku zakładkach równocześnie.

wyniki w tabelach 5.2, 5.4 i 5.3 na stronie 70 na których widać, że czas rozwiązywania problemu jest zbliżony dla każdej z kart.

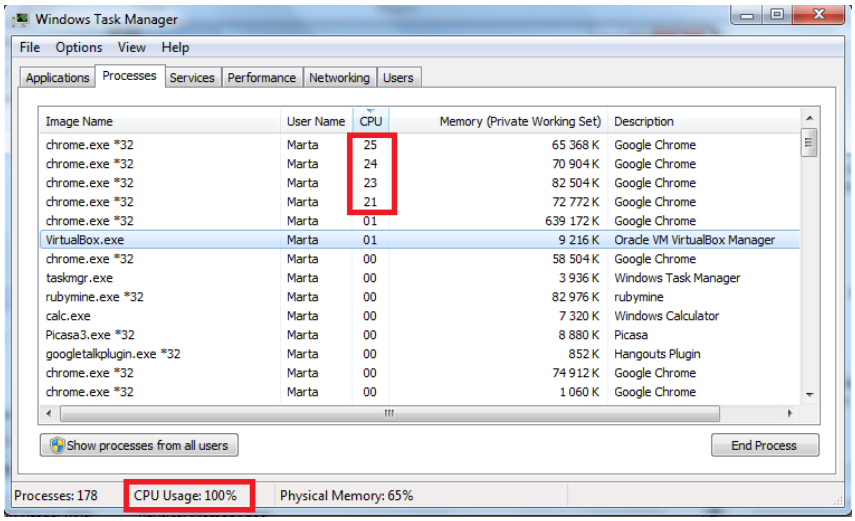
### 5.3.2. Skuteczność rekrutowania ochotników do obliczeń w przeglądarkowym modelu Volunteer Computing

Celem eksperymentu było sprawdzenie ile rozwiązań uda się zebrać udostępniając link prowadzący do strony internetowej, dzięki której można dołączyć do obliczeń. Link został udostępniony poprzez kilka internetowych kanałów społecznościowych:

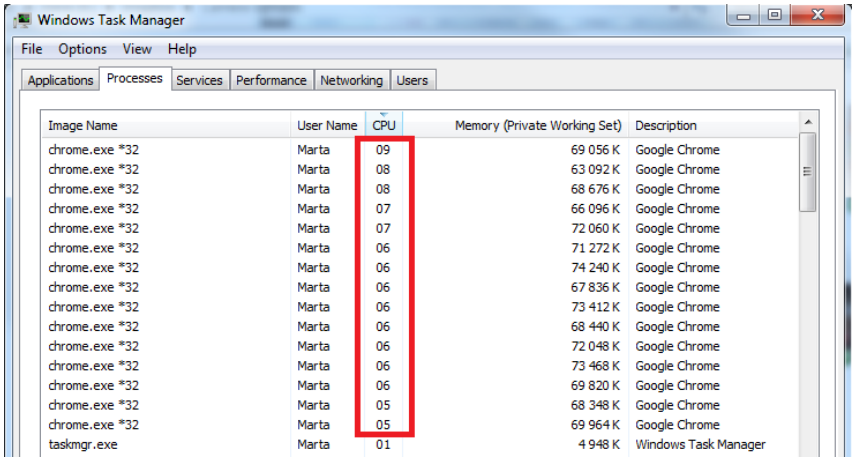
– na prywatnej tablicy w serwisie Facebook (520 znajomych)

– na tablicy Studenckiego Klubu Narciarskiego FIRN AGH w serwisie Facebook (180 obserwujących)

– w serwisie Wykop.pl




Rysunek 5.13: Podział zużycia procesora dla 4 zakładek przeglądarki Chrome prowadzącej obliczenia.




Rysunek 5.14: Podział zużycia procesora dla 15 zakładek przeglądarki Chrome prowadzącej obliczenia.

– z prywatnego konta w serwisie Twitter (120 obserwujących)

Przez 6 h od udostępnienia linku, stronę odwiedziło 221 unikatowych odwiedzających podczas 259 wizyt. W tym czasie do naszej bazy danych wpłynęło ponad 2000 rozwiązań. W ciągu kolejnych 24 godzin, do bazy spłynęło jeszcze ponad 2500 rozwiązań, co pokazuje, że część ludzi raz zarekrutowanych do obliczeń, została wierna eksperymentowi przez następny dzień. Po tygodniu od akcji promującej, w bazie było ponad 6 tysięcy nowych rozwiązań.

Przy najbardziej efektywnym użyciu zasobów średniej klasy komputera typu laptop (procesor Intel Core i5, 4 rdzenie), 6 h obliczeń w najbardziej wydajnym modelu (równolegle uruchomione 9 zakładek przeglądarki Chrome) przyniesie około 900 rozwiązań (6 h = 21600 s; Tempo 23.9 s / 1 rozwiązanie) Ten sam efekt udało się uzyskać, dzięki zaprzęgnięciu do obliczeń ochotników, w niecałe 2 godziny. Nie jest to wielkie przyspieszanie, ale należy zwrócić uwagę, że podjęcie obliczeń było czysto dobrowolne, a osoby biorące w nich udział nie były o to personalnie proszone, tylko same zdecydowały się dołączyć do projektu.

Ciekawym efektem było to, że osoby biorące udział w obliczeniach, poczuły nutkę rywalizacji w celu osiągnięcia przez swoją maszynę najlepszego wyniku obliczeń (tj. toru przejazdu, który narciarz pokonuje w najkrótszym czasie) i chętnie chwaliły się w serwisie społecznościowym Facebook osiągniętymi wynikami. Ochotnicy pozytywnie wypowiadali się też o tym, że na bieżąco mogli obserwować postęp obliczeń, a nawet aktualnie ewaluowaną

trasę. Osoby, które nie były obeznane w temacie narciarstwa alpejskiego, prosiły o wytłumaczenie dlaczego konfiguracja slalomu wygląda tak, a nie inaczej. Udało się zatem zainteresować samym problemem dużą grupę ludzi, co jest sukcesem samym w sobie, osiągniętym w znacznym stopniu dzięki wyborze architektury, który przyczynił się do łatwej dostępności platformy.

## 5.4. Podsumowanie

Podsumowując zaproponowane przez nas rozwiązanie architektoniczne, czyli platformę Volunteer Computing dostępną i wykonująca obliczenia wewnątrz przeglądarki internetowej wymienimy zalety i wady tego rozwiązania.

Zalety

- łatwa dostępność
  - brak konieczności instalacji dodatkowego oprogramowania po stronie ochotników
  - brak bariery wejścia do używania nowego oprogramowania - wszystko dzieje się z znanym środowisku przeglądarki i wygląda jak standardowa strona internetowa
  - prostota promowania platformy w portalach społecznościowych
- prosta i szybka implementacja
  - łatwa do zaimplementowania wizualizacja zjawisk/ obliczeń
  - wieloplatformowość

Dzięki tym zaletom technicznym, zostały uwidocznione fakty przemawiające za stosowaniem zaproponowanej architektury do analogicznych obliczeń. Jest to przede wszystkim propagowanie wiedzy o problemach naukowych - łatwy dostęp i atrakcyjna wizualizacja, która pozwala na wytłumaczenie istoty rozwiązywanego problemu.

Wady

- brak możliwości automatycznego uruchamiania i przerywania obliczeń w zależności od aktualnego zużycia procesora
- brak możliwości programowania konfiguracji przeprowadzanych obliczeń (np. równoczesnego uruchamiania obliczeń w X zakładkach przeglądarki)
- nie optymalna pod względem wydajności implementacja obliczeń - implementacja w języku JavaScript
- niewielka ilość bibliotek numerycznych i naukowych do obliczeń w środowisku JavaScript

Podsumowując, dochodzimy do wniosku, że zaproponowana przez nas architektura nie jest w tym momencie konkurencyjna dla obecnie funkcjonujących platform ze względu na wydajność i ograniczoną konfigurowalność. Jest natomiast ciekawym, komplementarnym do tradycyjnego podejścia rozwiązaniem, które, dzięki łatwej dostępności i braku bariery wejścia może być używane chociażby do promowania koncepcji Volunteer Computing czy też faktycznego prowadzenia obliczeń dla małych, amatorskich projektów.

Rozważmy możliwy scenariusz promowania idei Volunteer Computing na przykładzie projektu fightAIDS@home, którego celem jest znalezienie lekarstwa na AIDS. Aby przystąpić do obliczeń należy ściągnąć specjalną aplikację kliencką, analogiczną do aplikacji opisywanych w rozdziale **??** na stronie **??**. Dzięki niej możemy nie tylko kontrybuować swoimi zasobami, ale również na bieżąco śledzić wizualizowane w 3D postępy obliczeń. Największą barierą jest właśnie bariera wejścia. Proponujemy zatem, by stworzyć prosty prototyp, który będzie

symulował przystąpienie do obliczeń za pośrednictwem przeglądarki internetowej, oraz wykonywał i wizualizował - najprostsze choćby operacja w jakikolwiek sposób związane z projektem fightAIDS@home. Osoba, która trafi na stronę internetową z takim łekkim"klientem Volunteer Computing odczuje - choćby przez to, że usłyszy, że procesor jej PC-ta zaczął intensywnie pracować - jak to jest kontrybuować do projektu w szczytnym celu. Taka osoba, powinna dostać bardzo szybki feedback dotyczący postępu i tego w jak niewielkim stopniu przyczyniła się do realizacji celu projektu. Następnie powinna dostać zestawienie, jak dużo bardziej może pomóc, ściągając specjalistyczne oprogramowanie i dołączając do już prawdziwej wersji obliczeń. Oczywiście można sobie wyobrażić, że w tej wersji demonstracyjnej, po stronie klienta nie są wykonywane żadne konkretne obliczenia a całość ma tylko za zadanie zachęcić do ściągnięcia oprogramowania do prawdziwych obliczeń.

Przemyślenie: brak zarządzania, automatycznego uruchomiania obliczeń, ryje procek cały czas bez względu na to jak potrznuje go zwykły użytkownik komputera

co z tego że super konfiguracje jak nie ma narzędzia do uruchamiani ich w takich konfiguracjach, nikt nie będzie tego robił ręcznie.

potencjał edukacyjny z małym uczuciem, że pomaga się nauce

# 6. Podsumowanie

W rozdziale tym przedstawiono informacje .

## 6.1. Podrozdział