

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

Wydział Informatyki, Elektroniki i Telekomunikacji

KATEDRA INFORMATYKI



PRACA MAGISTERSKA

MARTA RYŁKO, ANNA SKIBA

**OPTYMALIZACJA TORU PRZEJAZDU W NARCIARSTWIE
ALPEJSKIM PRZY WYKORZYSTANIU VOLUNTEER
COMPUTING W ŚRODOWISKU PRZEGŁĄDAREK
INTERNETOWYCH**

OPIEKUN PRACY:
dr inż. Roman Dębski

Kraków 2013

OŚWIADCZENIE AUTORA PRACY

OŚWIADCZAMY, ŚWIADOMI ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ WYKONALIŚMY OSOBIŚCIE I SAMODZIELNIE (W ZAKRESIE WYSZCZEGÓLNONYM WE WSTĘPIE) I ŻE NIE KORZYSTALIŚMY ZE ŹRÓDEŁ INNYCH NIŻ WYMIESIONE W PRACY.

.....

PODPISY

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

**SKI-LINE OPTIMISATION IN ALPINE SKI RACING WITH THE
USE OF WEB BROWSER-BASED VOLUNTEER COMPUTING.**

SUPERVISOR:
Roman Dębski Ph.D

Krakow 2013

Serdecznie dziękuję ...

Spis treści

1. Wstęp.....	7
2. Wprowadzenie dziedzinowe	9
2.1. Jazda po zadanym torze w narciarstwie alpejskim.....	9
2.2. Fizyczny model narciarza.....	10
2.3. Inne podejście do problemu znajdowania optymalnego toru przejazdu.....	12
2.4. Wpływ pozycji narciarza na szybkość poruszania się.....	14
3. Optymalizacja	15
3.1. Optymalizacja lokalna i globalna	15
3.2. Algorytm ewolucyjny	15
3.3. Algorytm memetyczny	20
3.4. Hill Climbing.....	20
4. Platformy Volunteer Computing	23
4.1. Wprowadzenie teoretyczne.....	23
4.2. Great Internet Mersenne Prime Searchy.....	25
4.3. Distributed.net	26
4.4. Berkeley Open Infrastructure for Network Computing.....	27
4.5. Urządzenia mobilne w rozproszonych obliczeniach naukowych.....	27
4.6. Folding@home	28
4.7. Web Workers	30
5. Model i proponowane rozwiązanie.....	32
5.1. Model narciarza i środowiska.....	32
5.2. Opis matematyczny modelu	32
5.3. Numeryczne rozwiązanie problemu	34
5.3.1. Rozwiązywanie na płaszczyźnie stoku	35
5.4. Optymalizacja toru przejazdu.....	37
5.5. Modelowanie karania	40
5.6. Architektura systemu.....	41
5.6.1. Architektura prototypowa	42
5.6.2. Architektura docelowa	42
5.6.3. Aplikacja kliencka	43
5.6.4. Aplikacja serwerowa.....	44
5.6.5. Interfejs systemu udostępniony użytkownikowi	44

6. Wyniki	49
6.1. Weryfikacja modelu.....	49
6.1.1. Podstawowe założenia modelu	50
6.1.2. Eksperymenty w terenie.....	55
6.1.3. Jazda po łuku, a jazda w skos stoku.....	62
6.2. Optymalizacja toru przejazdu.....	62
6.2.1. Algorytm ewolucyjny.....	63
6.2.2. Lokalna optymalizacja.....	77
6.2.3. Strategie karania.....	83
6.3. Volunteer Computing.....	84
6.3.1. Wydajność przeglądarek dla intensywnych obliczeń.....	84
6.3.2. Skuteczność rekrutowania ochotników do obliczeń w przeglądarkowym modelu Volunteer Computing.....	87
6.4. Wnioski.....	90

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 pozwoliłoby znaleźć optymalny, tj. najszybszy poprawny tor przejazdu uwzględniający narzucone ograniczenia.

Znajdowanie toru pretendującego do bycia nazywanym optymalnym, szczególnie dla dłuższych tras, jest problemem złożonym obliczeniowo. Spróbujmy wyobrazić sobie realne zastosowanie, w którym trener czy też zawodnik, tuż przed zawodami czy treningiem, najpierw mapuje faktyczne rozstawienie bramek slalomowych za pomocą aplikacji mobilnej zainstalowanej na urządzeniu z bardzo dokładnym odbiornikiem sygnału GPS, a następnie chce w czasie prawie rzeczywistym otrzymać odpowiedź - tj. jak najlepszy tor, który powinien zostać wybrany podczas przejazdu. Ze względu na wymaganą szybkość odpowiedzi, obliczenia nie mogą być wykonywane na samym urządzeniu mobilnym. Interesującym rozwiązaniem jest użycie do tego celu nie tyle płatnych serwerów w chmurze, co wykorzystanie mocy obliczeniowej drzemiącej w maszynach innych fanów narciarstwa w modelu 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 ustalone na trasie bramki - wymuszające skręty.

W tej pracy podejmujemy się rozwiązania problemu optymalizacyjnego rozwiązywanego za pomocą symulacji komputerowej. Dotyczy on 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 jest nietrywialny i w celu badania go nieodłączne są osoby o różnych kompetencjach.

Problem jest na tyle złożony, że próby jego rozwiązania to tak naprawdę poszukiwanie rozwiązania problemu uproszczonego. 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 stworzyliśmy fizyczny model narciarza. W modelu tym, narciarz traktowany jest 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 modelo-

wany jest jako płaszczyzna o zadanym kącie nachylenia, na której za pomocą współrzędnych oznaczamy miejsce występowania bramek. Dużym wyzwaniem jest dobranie takiego przybliżenia trasy przejazdu, aby zachować równowagę między dobrym oddaniem rzeczywistej trasy a akceptowalnym czasem obliczania czasu przejazdu po trasie. Łamana, którą wybrałyśmy jako rozwiązanie spełniające obydwa te wymagania, jest wystarczająco dobrym przybliżeniem trasy przejazdu jeśli narzucimy na nią dodatkowe ograniczenia jak eliminacja ostrych kątów załamania.

Kluczową częścią naszego rozwiązania jest wykorzystanie algorytmu ewolucyjnego do znalezienia lokalnego optimum trasy, a następnie przeprowadzenie lokalnej optymalizacji celem wygładzenia znalezionego rozwiązania. Aby przyspieszyć obliczenia, została zastosowana architektura typu master-slave, bazująca na rozproszonych klientach wykonujących obliczenia i raportujących do głównego serwera. Obliczenia wykonywane są w środowisku przeglądarek internetowych w języku JavaScript w modelu Volunteer Computing.

Na kanwie naszego rozwiązania można budować systemy z innymi przypadkami użycia niż tylko wybieranie optymalnej trasy przejazdu. Jedną z możliwych dróg rozwoju jest aplikacja dla trenerów, wspomagająca proces ustawianiu trasy slalomu poprzez podpowiadanie 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, poprzez wykrywanie takich sytuacji i sugerowanie przestawienia bramki w lepsze miejsce.

Celem tej pracy jest zaprezentowanie rozwiązania, które za pomocą algorytmu ewolucyjnego i techniki lokalnej optymalizacji, użytej w środowisku rozproszonych obliczeń w modelu Volunteer Computing, umożliwia wyliczenie optymalnej trasy przejazdu narciarza po slalomie.

Na początku pracy znajduje się wprowadzenie teoretyczne do omawianych zagadnień oraz przedstawienie istniejących rozwiązań dla dwóch aspektów pracy - optymalizacji przejazdu oraz zastosowania Volunteer Computing. W rozdziale 2 opisane zostały reguły jazdy po slalomie, poruszanie się narciarza z fizycznego punktu widzenia oraz rodzaje optymalizacji, które zostały zastosowane w rozwiązaniu. Rozdział zamyka opis prac z tej tematyki. Rozdział 4 wprowadza pojęcia związane z konceptem Volunteer Computing oraz przedstawia ich użycie w kilku platformach.

Kolejny rozdział zawiera proponowane rozwiązanie, zaczynając od opisu wybranego modelu - zarówno środowiska, jak i sposobu poruszania się narciarza. Następnie zaprezentowane zostało w jaki sposób poszukiwany jest optymalny tor przejazdu. Reszta rozdziału poświęcona jest architekturze rozwiązania.

Następnie zebrane zostały uzyskane wyniki dotyczące nie tylko rezultatów optymalizacji, ale także skuteczności w uzyskiwaniu dużej liczby rozwiązań dzięki Volunteer Computing. Oprócz komputerowych symulacji zostały przeprowadzone i opisane w tym rozdziale również eksperymenty praktyczne mające na celu zweryfikowanie przyjętych założeń.

W rozdziale ostatnim umieszczone zostało podsumowanie zbierające wnioski z otrzymanych rezultatów oraz przedstawiające możliwe kierunki rozwoju.

2. Wprowadzenie dziedzinowe

Zrozumienie istoty przedstawionego w tej pracy rozwiązania 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 kilka 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 przejechaniu każdej z bramek znajdujących się na trasie przejazdu. Szczegółowe zasady dotyczące parametrów stoku i sprzętu określa regulamin organizacji FIS (Federation International du Ski), spośród których najbardziej istotnymi są:

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

Parametry te różnią się w zależności od dyscypliny. Najbardziej techniczną dyscypliną jest slalom, nazywany wcześniej slalomem specjalnym. Wyróżnikiem tej dyscypliny jest duża liczba bramek znajdujących się w nie-wielkiej odległości od siebie. Gęste ustawienie bramek wymusza częste skręty, bardzo liczy się zatem technika i zwinność narciarza. 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 35 metrów. Kolejne dyscypliny są jeszcze szybsze a promienie nart coraz większe. Bramki zarówno w super gigancie jak i jazdzie, są analogiczne do tych gigantowych, różnią się tylko szerokością płacht. Super gigant to konkurencja podobna do slalomu giganta, z tym, że odległości między bramkami są bardzo duże, a zatem zawodnicy uzyskują podczas przejazdu dużo większe prędkości. 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 80 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 zawsze ich ustawienie

jest 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*, który może występować w każdej z dyscyplin. Przelot polega na ustawieniu dwóch kolejnych bramek w taki sposób, by nie wymuszać skrętu pomiędzy nimi. Przelot stosowany jest w celu lepszego dostosowania ustawienia trasy przejazdu do konfiguracji terenu, np. gdy na trasie naturalnie występuje dłuższy skręt w jedną ze stron lub aby dać zawodnikowi możliwość rozpędzenia się. Kolejną figurą, stosowaną już tylko w slalomie, jest *łokiec*. Łokiec to dwie bramki ustawione w bliskiej odległości jedna pod drugą w linii spadku stoku. Są to bramki zamknięte, czyli takie, w których światło bramki jest równoległe do linii 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 przypadku łokcia, czyli jedna pod drugą, w linii spadku stoku, z tym, że zamiast dwóch, mogą to być trzy, cztery czy nawet pięć kolejno tak ustawianych bramek.

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. Czasem warto skręt rozpocząć wcześniej, tak by zmieścić się w kolejnej podkręconej bramce, czasem z kolei bardziej optymalne jest rozpędzenie się i rozpoczęcie skrętu z opóźnieniem, jeśli tylko dalsza konfiguracja slalomu pozwoli na zmieszczenie się w kolejnych bramkach, mimo posiadania znacznej prędkości.

2.2. Fizyczny model narciarza

Aby dobrze odwzorować jazdę narciarza po slalomie, należy mieć na uwadze zachodzące w jego środowisku zjawiska fizyczne. W tym rozdziale zostaną przedstawione przekrojowo te zjawiska.

Siła oporu powietrza

Siła oporu powietrza jest przykładem siły tarcia płynu. Zależność wartości tej siły od prędkości może być bardzo złożona. Tylko w specjalnych przypadkach równanie opisujące siłę oporu powietrza może być rozwiązane analitycznie. Siła oporu powietrza jest w przybliżeniu proporcjonalna do kwadratu prędkości poruszającego się obiektu i zależność tą, można opisać równaniem:

$$F_d = kv^2 = \frac{1}{2}C\rho Av^2 \quad (2.1)$$

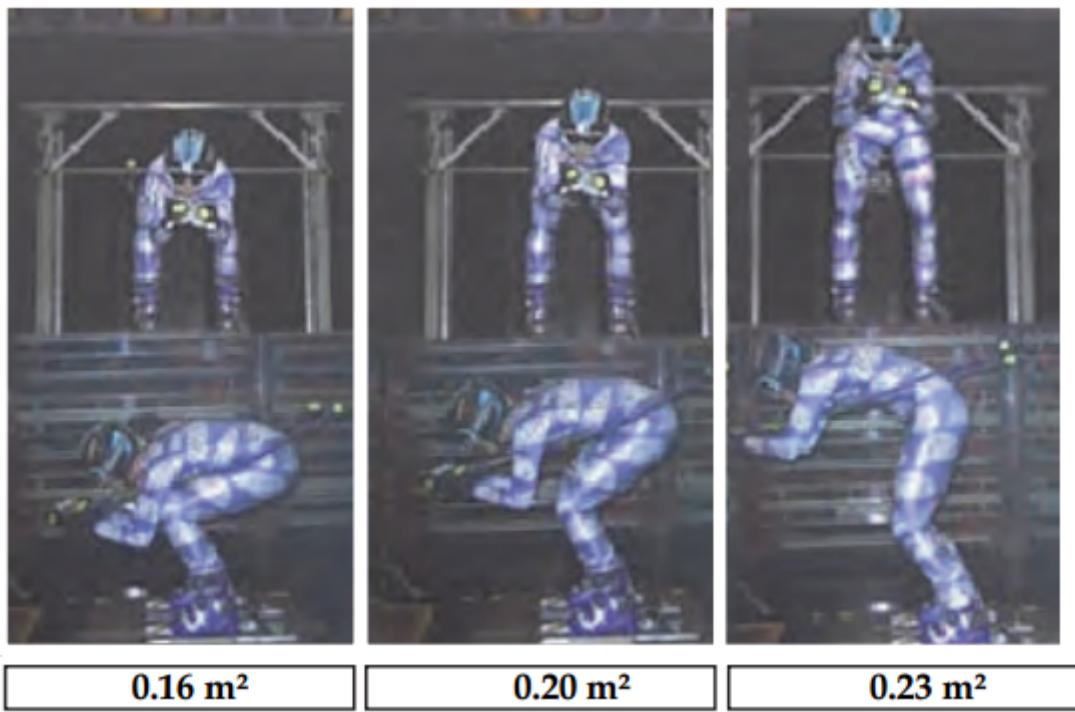
C - współczynnik oporu powietrza, typowe wartości oscylują między 0.4 a 1. Wartość współczynnika zależy między innymi od kształtu obiektu i można go wiązać z tzw. „opływowością” obiektu

ρ - gęstość powietrza w jednostce $\frac{kg}{m^3}$. 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 .

Pozycja narciarza ma znaczący wpływ na wartość współczynnika A, co widać na rysunku 2.2 na stronie 11. 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 jednocręciowe i ściśle przylegają do ciała. Nie posiadają żadnych

Rysunek 2.1: Wartość współczynnika A w zależności od pozycji narciarza. Grafika pochodzi z badań prowadzonych w tunelu aerodynamicznym IAT we Francji, opisanych przez Caroline Barelle z National Technical University of Athens z Grecji. [3]



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 znaczco zmniejsza 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 kilkadziesiąt setnych sekund.

Interakcja między śniegiem a nartami

To, że narty ślizgają się na śniegu zawdzięczamy skomplikowanym oddziaływaniom między powierzchnią narty a śniegiem czy lodem. Liczba czynników jakie wpływają na tę interakcję jest bardzo duża, a spośród 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
- wartość kąta nachylenia między powierzchnią ślizgu a podłożem

Jest jeszcze jeden czynnik wpływający na tę interakcję, tzw. *water suction* (w wolnym tłumaczeniu zasysanie wody). W temperaturze powyżej $-3^{\circ}C$, 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 [6], tarcie o śnieg ma dużo większy wpływ na czas przejazdu niż siły oporu powietrza.

Tarcie kinetyczne

Tarcie kinetyczne ma dużo większe znaczenie niż tarcie statyczne ponieważ determinuje jak duża siła musi działać, żeby zachować pożądaną prędkość podczas zjazdu. Współczynnik tarcia kinetycznego między śniegiem a nasmarowanymi nartami wynosi średnio 0.05 [6]. Współczynnik ten jednak może się znaczco zmieniać i w zależności od rodzaju smarów, sposobu smarowania oraz jakości śniegu wynosi między 0.001 a 0.3 [6]. 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. Inne podejścia do problemu znajdowania optymalnego toru przejazdu

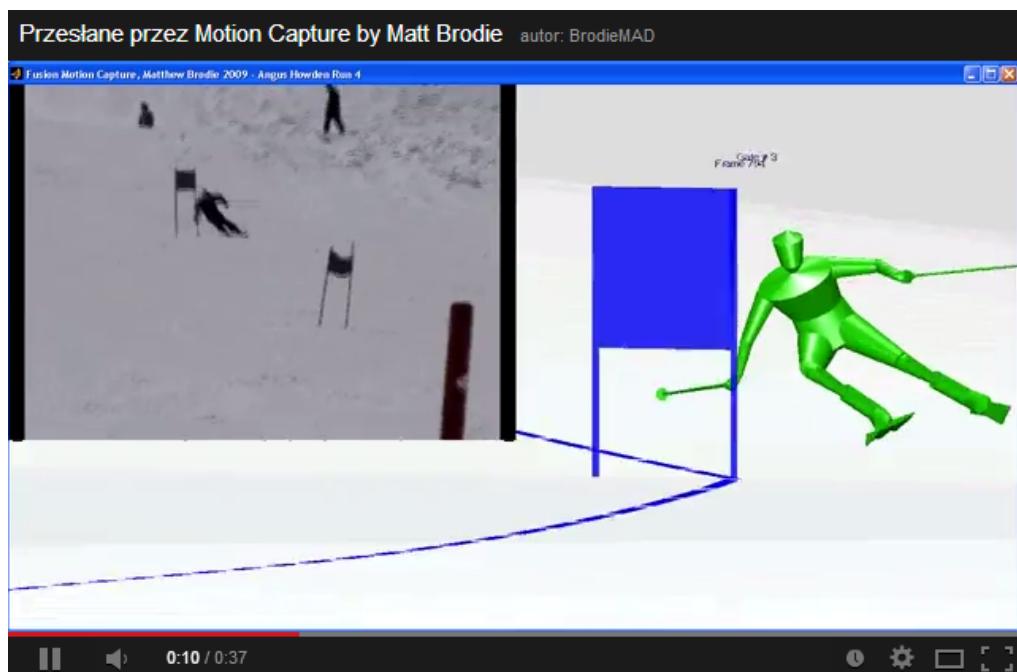
Matthew Brodie z Massey University w Nowej Zelandii jako pierwszy na świecie wykorzystał własnoręcznie zaprojektowany i wykonany system składający się z sieci miniaturowych czujników mocowanych na ciele zawodnika-narciarza, połączony z GPS w celu śledzenia i analizy ruchu narciarza jadącego po slalomie. System, nazywający się Motion Capture Fusion (FMC), został stworzony, by jak twierdzi autor, pomóc narciarzom ze światowej czołówki lepiej rozumieć, co dzieje się z nimi podczas jazdy, zoptymalizować jazdę, a w konsekwencji doprowadzić ich do większej liczby zwycięstw i złotych medali. Opis systemu oraz wnioski zamieścił w swojej pracy doktorskiej pod tytułem *Optimisation of Performance in Alpine Ski Racing with Fusion Motion Capture* [1]. Prezentację systemu oraz jego funkcjonowania można obejrzeć w kanale Matthew'a w serwisie YouTube.

Poniżej przedstawimy najważniejsze wnioski Matthew'a, zawarte w jego pracy doktorskiej

- Optymalna strategia na pokonywanie trasy slalomu jest kompromisem między ciasnymi skrętami, prowadzącymi do toru o mniejszej długości, a bardziej otwartymi skrętami o większym promieniu, które umożliwiają utrzymanie większej prędkości w skręcie.
- Najkrótsza droga wcale nie jest często najszybszą drogą. Każda bramka ma inny optymalny sposób na jej przejechanie.
- Zawodnicy często dobierają skręty o identycznym promieniu skrętu, bo jest to dla nich bardziej naturalne niż skręty o zmiennym promieniu. Tymczasem warto do każdej bramki dobierać optymalny promień i punkt rozpoczęcia skrętu.
- W niektórych konfiguracjach terenu, optymalny promień skrętu jest ograniczony poprzez maksymalny kąt pomiędzy krawędziami nart a śniegiem, jaki narciarz może utrzymać.
- Wyniki doświadczalne zaprzeczają wynikom laboratoryjnym i sugerują, że podczas przejazdu slalomu gainta, opór śniegu ma tak samo duże znaczenie, co opór powietrza.
- W pewnych warunkach, tj. przy prędkości mniejszej niż $20\frac{m}{s}$ i terenie o nachyleniu mniejszym niż 25%, narciarz może poprzez pracę mięśni - tzw. *pompowanie* - zwiększyć przyspieszenie w zakrętach, a więc w rezultacie zmniejszyć czas przejazdu.
- Dobór sprzętu, dostosowanego zarówno do konkretnej trasy i jej nachylenia, jak i do warunków fizycznych zawodnika, ma znaczny wpływ na otrzymywane przez niego wyniki.



Rysunek 2.2: Klatka z materiału wideo na kaneli Matthew'a umieszczonego w serwisie YouTube, na którym pokazuje przygotowania do testowania systemu na stoku narciarskim. Na kasku narciarza widać zamontowany GPS



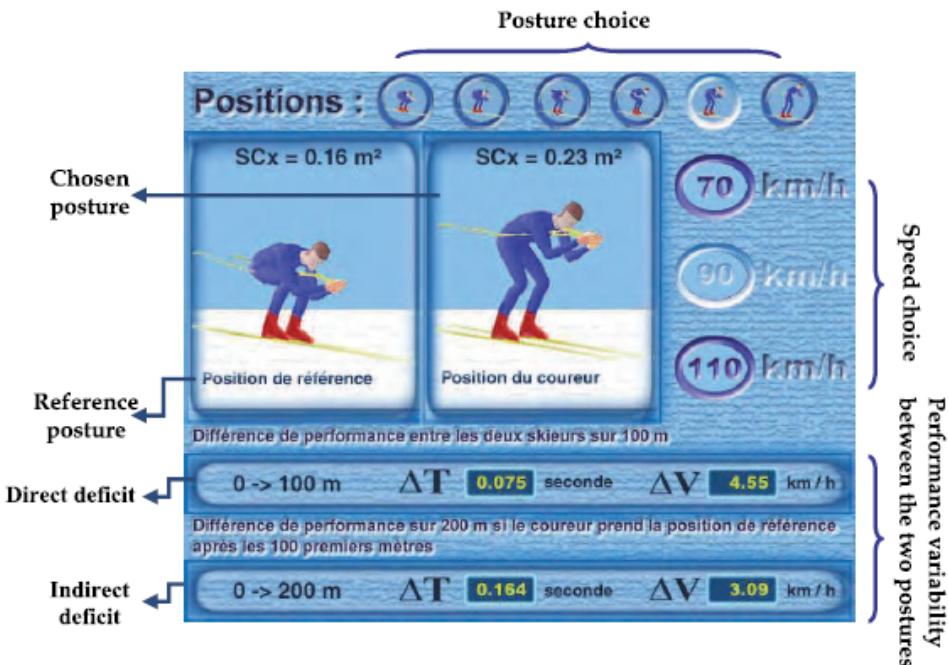
Rysunek 2.3: Klatka z materiału wideo na kaneli Matthew'a umieszczonego w serwisie YouTube, która przedstawia wizualizację zebranych na stoku danych w stworzonym przez Matthew'a oprogramowaniu.

2.4. Wpływ pozycji narciarza na szybkość poruszania się

W nawiązaniu do sekcji teoretycznej 2.2 ze strony 10, w której jest zaznaczony wpływ sylwetki narciarza na wartość siły oporu, warto pokazać aplikację stworzoną przez Francuską Federację Narciarską.

Aplikacja jest skierowana do narciarzy z federacji i ma im pokazać jak bardzo wybór pozycji wpływa na ich ostateczny wynik na zawodach. Użytkownik ma do wyboru sześć różnych pozycji zjazdowych oraz trzy wartości prędkości początkowych. Może wybrać pozycję bazową oraz pozycję docelową oraz prędkość jaką ma będąc w pozycji bazowej. Program symuluje jaka będzie zmiana prędkości narciarza po przejściu do pozycji docelowej po 100 metrach jazdy w dół w linii prostej. Co więcej, program wskazuje jaka będzie różnica w prędkości narciarza po 200 metrach, z założeniem, że po 100 metrach narciarz wrócił do pozycji referencyjnej. Jest to o tyle istotne, że pozwala uświadomić zawodnikom, że powinni wykorzystywać każdą sytuację do przyjęcia jak najbardziej optymalnej pozycji.

Zrzut ekranu aplikacji jest na rysunku 2.4 na stronie 14, a aplikacja opisana jest w pracy [3].



Rysunek 2.4: Zrzut ekranu z aplikacji Francuskiej Federacji Narciarskiej, zaczerpnięty z pracy AirDrag [3]

3. 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 najlepszego. Zatem, dana jest funkcja, nazywaną funkcją celu, która każdemu punktowi reprezentującemu rozwiązanie problemu przypisuje jakąś wartość oceniającą jego jakość. Wśród wszystkich rozwiązań poszukujemy takiego, dla którego wartość tej funkcji będzie jak najmniejsza (bądź jak największa) - najlepsza z naszego punktu widzenia. Trudność w znalezieniu takiego rozwiązania zależy od charakteru funkcji celu, a czasem także od nieznajomości jej analitycznej postaci.

3.1. 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. Przykładem optymalizacji lokalnej jest algorytm Hill Climbing.

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.

3.2. Algorytm ewolucyjny

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

“Algorytm ewolucyjny przetwarza populację osobników, z których każdy jest propozycją rozwiązania postawionego problemu. Działa on w środowisku, które można zdefiniować na podstawie problemu rozwiązywanego przez algorytm. W środowisku każdemu osobnikowi jest przyporządkowana wartość liczbową, określająca jakość reprezentowanego przez niego rozwiązania; wartość ta jest nazywana przystosowaniem osobnika.” [2]

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 działania algorytmu ewolucyjnego Na początku działania algorytmu ewolucyjnego generowana jest populacja bazowa oraz obliczane przystosowanie każdego z jej osobników. Przeważnie osobniki te generowane są całkowicie losowo, ale można także wprowadzić konkretne osobniki np. o znany dobrym przystosowaniu do środowiska.

Główna część algorytmu opiera się na powtarzaniu pętli, w której wykonywane są kolejne kroki przedstawione w 1.

Algorithm 1 Schemat działania algorytmu ewolucyjnego

reprodukcia

operacje genetyczne

ocena

sukcesja

Często reprodukcję i sukcesję łączy się pod nazwą selekcja.

Reprodukcia 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 genotypów 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.

Decyzja o zakończeniu działania algorytmu przeważnie podejmowana jest w oparciu o badanie wartości 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 przystosowania rozwiązania, więc nie możemy stwierdzić kiedy przystosowanie jest odpowiednie i czy nie może się jeszcze znacznie poprawić.

Kodowanie osobników W przypadku algorytmów genetycznych 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 “na 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 działania. 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 działania algorytmów ewolucyjnych.

1. Prosty algorytm genetyczny

Prosty algorytm genetyczny został zaproponowany w roku 1975 przez John'a Holland'a w [12].

Poniżej umieszczony jest schemat tego algorytmu (Algorytm 2 na podstawie algorytmu umieszczonego w [2]).

Algorithm 2 Prosty algorytm genetyczny

```
t = 0
P0 = createInitPop()
while stopCondition == false do
    Tt = createTempPop(Pt)
    Tt = crossPop(Tt)
    Ot = mutatePop(Tt)
    Pt+1 = Ot
    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.

2. Strategia (1+1)

Strategia (1+1) jest strategią ewolucyjną zaproponowaną przez Schwefel'a w 1965 r. [18], a później analizowana również w pracach Rechenberg'a [15], [16]. W algorytmie tym mamy do czynienia z populacją składającą się tylko z 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:

$$Y_i^t = X_i^t + \sigma \xi_{N(0,1),i} \quad (3.1)$$

Wartość σ 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 σ przeszukując tylko najbliższą przestrzeń.

Do wyznaczania σ powstał następujący algorytm zwany regułą 1/5 sukcesów ([16]):

- (a) Jeśli przez kolejnych k pętli algorytmu mutacja powoduje powstanie lepszego osobnika w więcej niż 1/5 wszystkich mutacji, to zwiększamy σ : $\sigma' = c_i \sigma$. Wartość c_i wyznaczona empirycznie wynosi $\frac{1}{0.82}$ ([21])
- (b) Gdy dokładnie 1/5 kończy się sukcesem, wartość σ pozostaje bez zmian.
- (c) Jeśli nie zachodzi żadne z powyższych wartości σ jest zmniejszana: $\sigma' = c_d \sigma$. Gdzie c_d powinna wynosić 0.82 ([21])

Opisane poniżej strategie są rozwinięciem strategii (1+1) i zostały zaproponowane przez Schwefel'a w [19], [20], [21].

3. Strategia $(\mu + \lambda)$

Strategia $(\mu + \lambda)$ jest rozwinięciem strategii (1+1). μ oznacza ilość osobników w populacji początkowej, a λ 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 μ najlepszych osobników do nowej populacji bazowej - zarówno spośród populacji potomnej, jak i starej populacji bazowej zawierających łącznie $\mu + \lambda$ osobników. Algorytm 3 przedstawia schemat tej strategii (bazując na algorytmie umieszczonym w [2]).

Algorithm 3 Strategia ewolucyjna $(\mu + \lambda)$

```

 $t = 0$ 
 $P^0 = createInitPop(\mu)$ 
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 chromosom przechowujący wektor σ 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 σ losujemy jeszcze jedną zmienną losową o rozkładzie normalnym ($\xi_{N(0,1),i}$) i obliczamy nowe wartości odchyleń z wektora σ :

$$\sigma'_i = \sigma_i e^{(\tau' \xi_{N(0,1)} + \tau \xi_{N(0,1),i})} \quad (3.2)$$

Gdzie τ oraz τ' są parametrami algorytmu, a ich wartości powinny wynosić według [22]:

$$\tau = \frac{K}{\sqrt{2n}} \quad (3.3)$$

$$\tau' = \frac{K}{\sqrt{2\sqrt{n}}} \quad (3.4)$$

gdzie:

- K - stała, najczęściej stosuje się wartość 1 ([22])
- n - wymiarowość zadania

Beyer w 2007 r. napisał, że empirycznie sprawdzono, że należy wybierać wartość τ proporcjonalnie do $\frac{1}{\sqrt{n}}$. [4]

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

$$X'_i = X_i + \sigma'_i \xi_{N(0,1),i} \quad (3.5)$$

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

Algorytm ewolucyjny wybiera osobniki lepiej przystosowane, uwzględniając przy tym również 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 składowe wektorów, także wartości σ .

4. Strategia (μ, λ)

Strategia $(\mu + \lambda)$ posiada pewne wady, które wyeliminowano za pomocą nowej strategii (μ, λ) . 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. Algorytm 4 prezentuje kolejne kroki schematu tego algorytmu (bazując na algorytmie umieszczonym w [2]).

Algorithm 4 Strategia ewolucyjna (μ, λ)

```

 $t = 0$ 
 $P^0 = createInitPop(\mu)$ 
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

```

3.3. Algorytm memetyczny

Richard Dawkins zaproponował w [7] pojęcie memu - “the basic unit of cultural transmission, or imitation”. Algorytmy ewolucyjne opierają się na ewolucji biologicznej, Dawkins zaproponował, że istnieje również rozwój kulturowy. Na tej podstawie wprowadzono algorytmy memetyczne, które stanowią rozwinięcie algorytmów ewolucyjnych poprzez dodanie do nich lokalnej optymalizacji. Może być ona wprowadzana na różnych etapach algorytmu ([5]):

- na początku, w celu wygenerowanie potencjalnie lepszej populacji początkowej
- w fazie oceny przystosowania
- po mutacji i krzyżowaniu
- po zakończeniu fazy ewolucyjnej do ostatecznego poprawienia rozwiązania

Łącząc te dwa algorytmy ze sobą, możemy wykorzystać i połączyć ich zalety. Uwzględnienie algorytmu lokalnej optymalizacji pomaga w skutecznym znajdowaniu optimum lokalnego, jednak aby móc przeszukiwać wszystkie optima w celu znalezienia najlepszego - globalnego, potrzebujemy eksploracji jaką daje nam algorytm ewolucyjny.

3.4. Hill Climbing

Podrozdział bazuje na opisie algorytmu Hill Climbing w [17].

Algorytm Hill Climbing jest jedną z metod przeszukiwania lokalnego. W podstawowej wersji algorytmu zwanej *Greedy local search*, w każdej iteracji zmieniając wartość rozwiązania w jednym z wymiarów 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.

Istnieje wiele rodzajów algorytmu rozwijających wersję podstawową:

- *Stochastic hill climbing* - wybiera kolejny punkt losowo, prawdopodobieństwo wyboru może zależeć od tego jak duża poprawa wiąże się z wybraniem danego punktu,
- *First-choice hill climbing* - wybiera pierwszego z generowanych kolejno sąsiadów, którego wartość jest lepsza niż obecny stan. Rozwiązanie to warto stosować jeśli istnieje wielu sąsiadów,

- *Random-restart hill climbing* - uruchamiając wielokrotnie algorytm rozpoczynając w losowych punktach możemy zwiększyć prawdopodobieństwo znalezienia rozwiązania globalnego.

Przeszukiwanie przestrzeni dyskretnej sprowadza się do sprawdzania 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. Pseudokod algorytmu przedstawiony jest poniżej (Algorytm 5).

Algorithm 5 Hill Climbing w przestrzeni dyskretnej

```

current = startPoint
foundBetter = true
while foundBetter == true do
    foundBetter = false
    neighbours = getNeighbours(current)
    for all neighbour in neighbours do
        if neighbour.isBetterThan(current) 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 rozwiązań. 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 ϵ wyznaczający tę różnicę. Pseudokod algorytmu przedstawiony jest na stronie 22 (Algorytm 6).

Algorithm 6 Hill Climbing w przestrzeni ciągłej

```
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.isBetterThan(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
```

4. Platformy Volunteer Computing

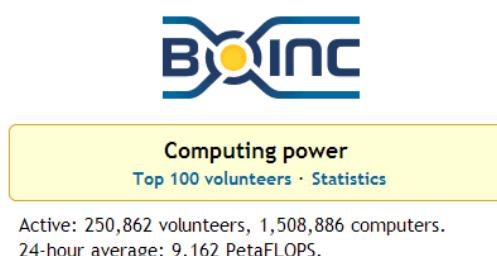
W rozdziale tym przedstawimy istniejące rozwiązania zarówno architektoniczne, które umożliwiają rozproszone obliczenia w modelu Volunteer Computing.

4.1. Wprowadzenie teoretyczne

Volunteer Computing to model obliczeniowy, w którym zwykli ludzie czy też organizacje, nazywani dalej ochronikami, dobrowolnie udostępniają swoje zasoby obliczeniowe, by uruchamiać na nich obliczenia związane z różnorakimi projektami. Są to przeważnie projekty naukowe, których celem jest rozwiązanie problemów i zaadań matematycznych czy też problemów dotyczących ludzkość, lub dążących do lepszego poznania świata i wszechświata. Dzięki platformom używającym modelu Volunteer Computing, każdy człowiek może mieć wkład w rozwiązywanie tych problemów.

Procesory w komputerach osobistych są bezczynne przez około 80% czasu. Równocześnie wiele naukowych problemów związanych na przykład z modelowaniem zmian klimatu potrzebuje ogromnych zasobów obliczeniowych. Używanie do tego celu superkomputerów jest bardzo drogie. Połączenie tych faktów doprowadziło do stworzenia modelu, wykorzystującego zasoby obliczeniowe ochroników, którzy chcą się nimi świadomie dzielić.

Moc obliczeniowa najmocniejszego superkomputera z listy TOP500, na listopad 2013, jest tego samego rzędu co moc obliczeniowa generowana przez ochroników używających platformy do Volunteer Computing'u - BOINC opisanej w rozdziale 27, na dzień 26 Grudnia 2013. Moc ta wynosi około 9 Peta flopsów dla BOINC i 33 Peta flopsów dla National Super Computer Center in Guangzhou, China z listy TOP500. Flops to jednostka wyrażająca moc obliczeniową i wskazującą ile operacji zmiennoprzecinkowych na jedną sekundę komputer może wykonywać. Moc rzędu 9 peta flops to $9 * 10^{15}$ operacji zmiennoprzecinkowych na sekundę



Rysunek 4.1: Zrzut ekranu ze strony internetowej <https://boinc.berkeley.edu/>, z dnia 26/12/2013 pokazujący średnią moc obliczeniową systemu z ostatnich 24 godzin.

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Super Computer Center in Guangzhou China	Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH-Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808

Rysunek 4.2: Zrzut ekranu ze strony <http://www.top500.org/lists/2013/11/> pokazujący pierwsze miejsce na liście TOP500 z listopada 2013 roku.

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. Czasem zdarza się, że ochotnicy z premedytacją przekłamują wysyłane do centralnego serwera wyniki swoich obliczeń. Z uwagi na anonimowość ochotników, utrudnione jest w takich przypadkach ich dyscyplinowanie czy też wyłączenie z projektu. Ochotnicy nie są wynagradzani finansowo za uczestnictwo w projekcie.

Organizacja czy osoba chcącą 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 obliczeniowego, na zainstalowanie aplikacji udostępnianej przez dostarczyciela 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. Ma też prawo oczekiwania, 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, jest atrakcyjnym celem ataków zmierzających do przejęcia tych maszyn przez hakerów, do niezgodnych z prawem celów.

Przeważnie model komunikacyjny systemu Volunteer Computing uwzględnia tylko komunikację poszczególnych klientów z centralnym serwerem i nie zakłada bezpośredniej komunikacji między klientami. [23]

Volunteer Computing pierwotnie zakładał, że obliczenia są wykonywane na zwykłych komputerach osobistych (PC). Liczba komputerów tego typu jest nieporównywalnie większa od liczby wyspecjalizowanych komputerów o dużej mocy obliczeniowej i jest szacowana na ponad miliard, choć obecnie liczba sprzedawanych komputerów tego typu z roku na rok spada. Dodatkowo, z przyczyn ekonomicznych, na rozwój tych maszyn producenci sprzętu przeznaczają bardzo duże fundusze, więc ich moc i zdolności obliczeniowe stale rosną. Warto jednak zwrócić uwagę na fakt, że obecnie najbardziej dynamiczny rozwój skierowany jest na urządzenia przenośne, o czym mowa w kontekście Volunteer Computing będzie w dalszej części rozdziału.

Ważnym aspektem, który istotnie wpływa na stosowanie modelu w praktyce, jest koszt prowadzenia obliczeń. Model zakłada, że dołączanie się do obliczeń jest dobrowolne 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. Gdy ochotnik przystępuje do projektu i udostępnia swoje moce obliczeniowe, można założyć, że jest zainteresowany, bądź chce się zainteresować problemem, który pomaga rozwiązać. Warto to zainteresowanie wykorzystać, by za pomocą przystępnych wizualizacji przedstawić mu sedno rozwiązywanego zadania, nakreślić mu problem z różnych perspektyw i wyjaśnić mu jaki jest cel prowadzonych obliczeń. Wy tłumaczony w atrakcyjnej formie problem z dziedziny fizyki czy biologii, połączony z niewymagającym dużego wysiłku wezwaniem do działania - dołączeniem do obliczeń - jest dobrym materiałem do udostępniania w portalach społecznościowych, a więc ma spory potencjał na organiczne rozprzestrzenianie się i zwiększanie zasięgu całego przedsięwzięcia obliczeniowego.

Klasyczny model obliczeń Volunteer Computing składała się z aplikacji klienckich, pobieranych z internetu oraz zainstalowanych na komputerze osobistym, oraz serwera, zarządzającego wysyłaniem zadań i odbieraniem rozwiązań.

Konieczność pobierania i instalowania aplikacji na komputerze implikuje wymóg tworzenia i utrzymywania wersji aplikacji pod każdy znaczący system operacyjny. Wybór odpowiedniej komplikacji programu klienckiego oraz proces jego instalacji jest pierwszą barierą jaką napotyka osoba chcąca mieć swój wkład do naukowego projektu.

Wobec stale rosnącej popularności urządzeń nie będących komputerami osobistymi, a również posiadającymi niewykorzystywaną w pełni mocą obliczeniową, również i one zaczynają być uwzględniane w modelu Volunteer Computing. Konsola Play Station 3 była pierwszym urządzeniem tego typu. W sierpniu 2013 udostępniono natomiast aplikację do naukowych projektów, dzięki której można włączyć się do projektów obliczeniowych poświęcając zasoby obliczeniowe swojego urządzenia z systemem Android. [13]

W ostatnich latach można zaobserwować trend przenoszenia oprogramowania, które dotychczas zainstalowane było na komputerach jako natywna aplikacja, do modelu SAAS (Software As A Service). SAAS to model dostarczania oprogramowania, w którym wszelkie interakcje z oprogramowaniem dokonywane są za pośrednictwem przeglądarki internetowej, nie wymagając instalacji dodatkowego oprogramowania po stronie użytkownika. W modelu SAAS, aplikacji uruchomiona jest na zdalnym serwerze, a wszelkie dane, które przetwarza przechowywane są również na serwerach w chmurze. Programy pocztowe, funkcjonalne systemy CRM, czy odtwarzacze muzyczne coraz częściej dostępne są właśnie w modelu SAAS. Taka zmiana możliwa była dzięki rozwojowi interpreterów Java Script we współczesnych przeglądarkach, oraz implementacji zaawansowanych elementów ze specyfikacji HTML5, w tym, Web Workers.

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

4.2. Great Internet Mersenne Prime Searchy

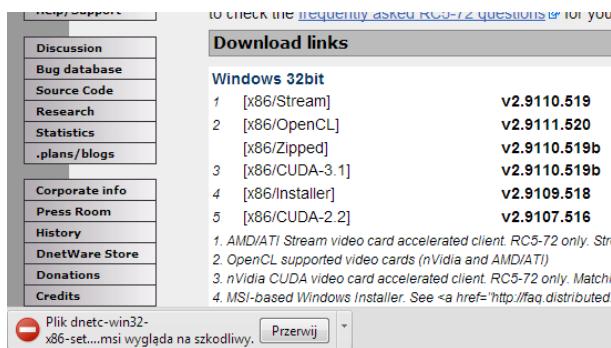
Pierwszym projektem wykorzystującym Volunteer Computing jest *GIMPS* (Great Internet Mersenne Prime Search) - rozpoczęty na początku 1996 roku a trwający do dziś projekt, którego celem jest znalezienie jak największej ilości specyficznych liczb pierwszych - tzn. Liczb Pierwszych Mersenne'a. Do tej pory kolektywny wysiłek doprowadził do odnalezienia czternastu takich liczb. Zsumowana moc obliczeniowa systemu dałaby teoretycznie, wg. danych na listopad 2013 roku, 330-te miejsce w rankingu TOP500 - rankingu najmocniejszych komputerów na świecie.

System składa się z aplikacji klienckich stworzonych dla 9 różnych wersji systemów operacyjnych (m.in. Windows 32/64-bit, Mac OS X, Linux 32/64-bit, FreeBSD 32/64-bit). Twórcy zapewniają, że kod programu jest wysoce zoptymalizowanym kodem asemblerowym. Po uruchomieniu, program nawiązuje kontakt z serwerem nazwanym *PrimeNet*, aby otrzymać porcję zadania obliczeniowego do przetworzenia.

GIMPS jest ciekawym projektem, który ma już na swoim koncie sukcesy i osiągnął duży zasięg i skalę, co potwierdzają publikowane regularnie informacje o nowych znaleziskach oraz wielkości sieci komputerów biorących udział w obliczeniach. Interfejs i wygoda użytkowania (ang. *User Experience*), które oferuje platforma, są jednak mało atrakcyjne i archaiczne i zupełnie nieprzystosowane do użytkowników współczesnego internetu.

4.3. Distributed.net

Kolejną znaczącą platformą do Volunteer Computing jest stworzony w 1997 roku *distributed.net*. Platforma 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 algorytm typu *brute-force*, która to dzięki modelowi Volunteer Computing przyniosła pozytywne skutki po 250 dniach poszukiwań całej przestrzeni rozwiązań.



Rysunek 4.3: 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 chcącą dołączyć do ochotników nie ma jednak łatwego zadania i proces dołączania do programu nie jest przyjemnym doświadczeniem. Należy wejść na stronę *distributed.net* i wybrać odpowiedni dla swojego systemu operacyjnego program kliencki. Wybór jest spory i może przysporzyć problemów mniej obeznanym w komputerach osobom, 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 zakończonym pobieraniu, użytkownik może dostać groźnie wyglądający komunikat, w którym zostanie ostrzeżony przez system operacyjny, że program jest potencjalnie szkodliwy dla komputera, jak przedstawiono na rysunku 4.3 na stronie 26.

Ufając dostawcy, po zdecydowaniu się na zainstalowanie programu, otwiera się mało przyjazny interfejs konsolowy, który widać na rysunku 4.4 na stronie 26.

The screenshot shows a terminal window titled 'distributed.net client'. The log output starts with processor detection and loading of crunchers, followed by keyserver synchronization and project state retrieval. It then enters a loop where it retrieves packers from the server, processes them, and runs micro-benchmarks to select the fastest core. The log ends with the start of a benchmark for core #0.

```

distributed.net client
Client Edit View Help
[Aug 17 20:11:35 UTC] Automatic processor detection found 4 processors.
[Aug 17 20:11:35 UTC] Loading crunchers with work unit size 1024...
[Aug 17 20:11:36 UTC] connected to server.v2.distributed.net:2064...
[Aug 17 20:11:36 UTC] The keyserver says "key pool (cdy)".
[Aug 17 20:11:36 UTC] retrieved project state data from server. (cached)
[Aug 17 20:13:59 UTC] OGR-NG: retrieved packet 96 of 96 (100.00%)
[Aug 17 20:14:21 UTC] rc5-72: retrieved stats units 96 of 96 (100.00%)
[Aug 17 20:14:21 UTC] connection closed.
[Aug 17 20:14:21 UTC] Automatic processor type detection did not
recognize the processor (tag: "100062A")
[Aug 17 20:14:21 UTC] OGR-NG: running micro-bench to select fastest core...
[Aug 17 20:14:21 UTC] usm selected core #0 (Celeron 4-24-8).
[Aug 17 20:14:21 UTC] OGR-NG #:0 loaded 2/2/1-21-5-4-24-8.
[Aug 17 20:14:21 UTC] OGR-NG #:1 loaded 2/2/1-21-5-4-24-12
[Aug 17 20:14:21 UTC] OGR-NG #:2 loaded 2/2/1-21-5-4-24-13
[Aug 17 20:14:21 UTC] OGR-NG #:3 loaded 2/2/1-21-5-4-24-15
[Aug 17 20:14:21 UTC] OGR-NG #:4 loaded 2/2/1-21-5-4-24-16
[Aug 17 20:14:21 UTC] OGR-NG: 92 packers remain in buff_in.og2
[Aug 17 20:14:21 UTC] OGR-NG: 0 packers are in buff_out.og2
[Aug 17 20:14:21 UTC] 4 crunchers ('a'-d') have been started.
[Aug 17 20:14:21 UTC] OGR-NG #:2/2/1-21-5-4-24-8 [0]
[Aug 17 20:14:21 UTC] OGR-NG #:3/2/1-21-5-4-24-16 [1]
[Aug 17 20:14:21 UTC] OGR-NG #:4/2/1-21-5-4-24-16 [2]
[Aug 17 20:14:21 UTC] OGR-NG #:1/2/1-21-5-4-24-8 [3]
[Aug 17 20:15:19 UTC] rc5-72: benchmark for core #0 (gas 1-pipe)
0.00:00:16.38 [4,813,691 keys/sec]
[Aug 17 20:15:19 UTC] rc5-72: using core #1 (ses 2-pipe).

```

Rysunek 4.4: Mało przyjazny interfejs konsolowy programu *distributed.net*

4.4. Berkeley Open Infrastructure for Network Computing

Do stworzenia platformy BOINC (Berkeley Open Infrastructure for Network Computing), obecnie najbardziej znaczącego open-source'owego middleware'u do obliczeń w modelu 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, na 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 rozwiązania były w całości fałszowane.

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

Serwer BOINC może być uruchomiony na jednej lub wielu maszynach z systemem operacyjnym Linux, a bazuje 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 toolkitie *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 atrakcyjny graficznie sposób np. animacje czy też wykresy wyjaśniające prowadzone obliczenia.

Obecnie prawie trzy miliony ludzi udostępnia zasoby swoich osobistych komputerów do obliczeń związanych tylko 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.

4.5. Urządzenia mobilne w rozproszonych obliczeniach naukowych

Sekcja napisana na podstawie [13].

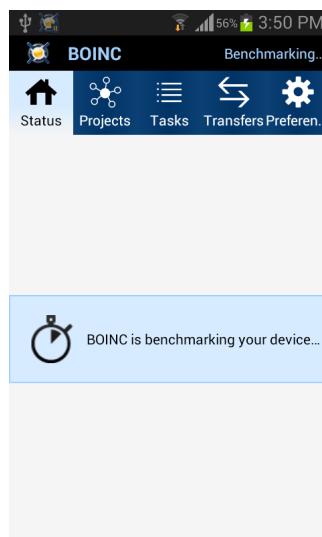
Przez dziesięciolecia miliony ludzi udostępniali zasoby swoich maszyn, aby wspomóc naukę. Jednak komputery typu PC są coraz mniej popularne, a ich sprzedaż spada o prawie 8% rocznie. Prognozy jasno wskazują na to, że ten trend się utrzyma. Z drugiej strony, bardzo dynamicznie rozwija się rynek urządzeń mobilnych, w tym smartphonów. Według serwisu Bloomberg, do końca roku 2013, na rynek zostanie dostarczonych 919 milionów nowych urządzeń, co jest 27%-owym wzrostem w stosunku do roku 2012. [13]

Do niedawna, telefony tego typu nie miały wystarczających zasobów, by prowadzić obliczenia naukowe. Obecnie mają nawet cztery rdzenie i mogą wykonywać półtora miliarda operacji numerycznych na sekundę (1.5 giga

FLOPSów), co odpowiada około jednej piątej tego co potrafią współczesne komputery osobiste. David Anderson, naukowiec z Berkeley, który współtworzy mobilną aplikację, 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 komputerów osobistych. Między innymi z tego powodu warto zacząć uwzględniać urządzenia mobilne w roli potencjalnego wykonawcy intensywnych obliczeń.

Ciekawą koncepcją jest obliczeniowa sieć stworzona z ogromnej liczby urządzeń, która mogłaby rywalizować z takimi dostawcami usług jak Amazon Web Services. W tym momencie wartość rynku komercyjnych obliczeń w chmurze osiągnęła 131 miliardów dolarów. Sieć stworzona z telefonów może być jego tańszym odpowiednikiem i umożliwić na przykład firmie farmaceutycznej płacenie po kilkadziesiąt centów miesięcznie za udostępnienie swojego urządzenia np. przez noc do potrzebnych jej obliczeń. Wizję tą roztacza Bernie Meyerson, wiceprezydent innowacji w firmie IBM. [13]

W sierpniu 2013 w 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. Rozważanie użycia 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 (prowadzi do szybszego zużycia baterii). Warunkami tymi może być np. podłączenie do źródła zasilania oraz poziom zużycia baterii powyżej zadanego progu.

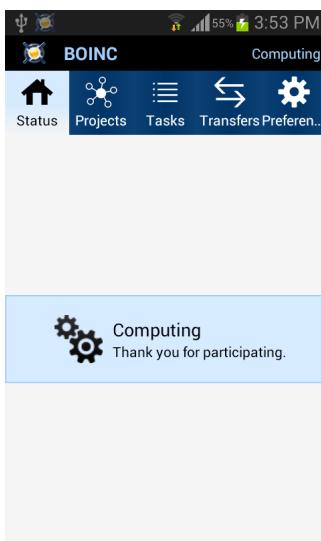


Rysunek 4.5: Proces benchmarkowania urządzenia, po pierwszym uruchomieniu aplikacji BOINC na system Android

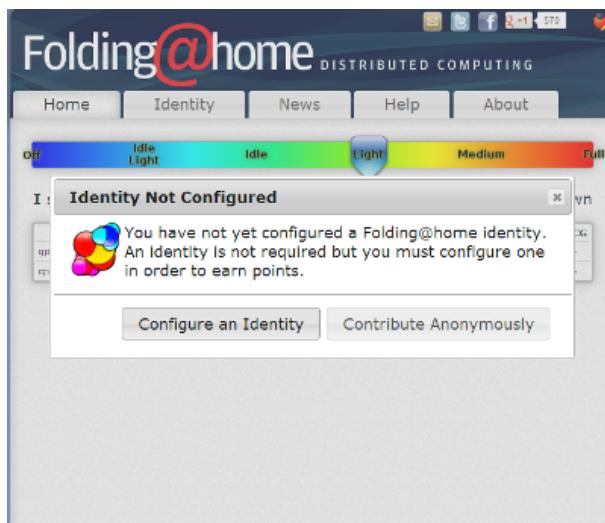
4.6. Folding@home

Projekt Folding@home wywodzący się z Amerykańskiego Uniwersytetu Stanford był pierwszym projektem w modelu Volunteer Computing, w którym wzięto pod uwagę, że potencjał Volunteer Computing nie zamknięty tylko na komputerach osobistych i wykorzystaniu jednostek CPU, ale także coraz bardziej rozwijanych jednostek GPU, procesorach na PlayStation 3.

Folding@home jest jednym z największych systemów komputerowych na świecie. Jego moc obliczeniową szacuje się na około 14 petaFLOPSów, czyli na większą wartość niż moc wszystkich projektów, które są przetwarzane na wywodzącej się z Berkeley platformie BOINC. W 2007 r. został wpisany do księgi rekordów Guinnesa



Rysunek 4.6: Aplikacja BOINC w trakcie obliczeń dokonywanych na platformie z systemem Android



Rysunek 4.7: Przyjazny użytkownikowi ekran powitalny interfejsu aplikacji *Folding@home*

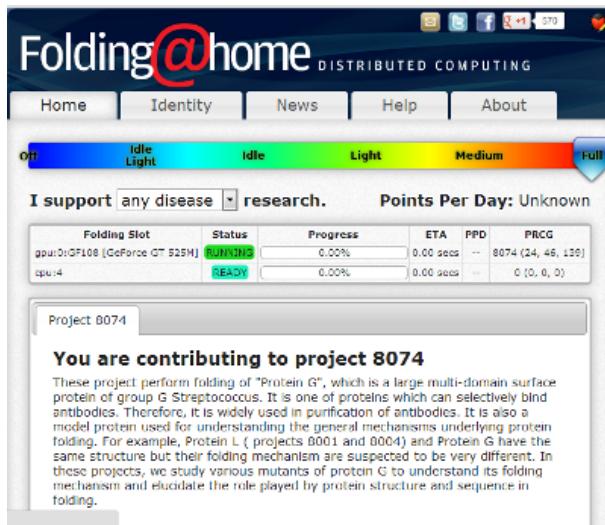
jako system rozproszony o największej mocy obliczeniowej. Stan na czerwiec 2013 r. wskazuje prawie pół miliona aktywnych CPU oraz prawie 25 tysięcy aktywnych GPU.

Folding@home 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, ochronicy biorący udział w symulacjach, mogą śledzić swój wkład w projekt Folding@home poprzez stronę internetową.

Można zaobserwować, że techniki grywalizacyjne zaczęły być coraz szerzej wprowadzane do projektów Volunteer Computing w celu utrzymania zainteresowania ochotników w branii udziału w projekcie.

Grywalizacja to wykorzystanie technik stosowanych do stymulowania zaangażowania graczy w grach (np. poprzez przyznawanie punktów czy odznak), aby wpływać na zachowanie ludzi w sytuacjach nie związanych z grami.

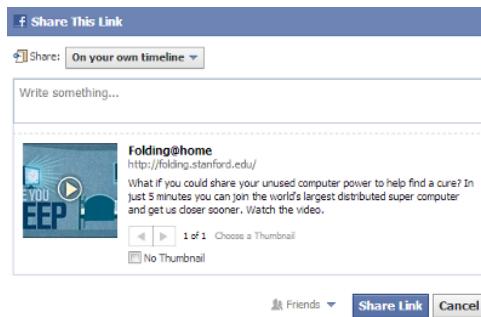
Folding@home wprowadza elementy grywalizacyjne poprzez zastosowanie systemu punktowego. Osoby udostępniające swoje zasoby dostają punkty za każde wykonane zadanie, dodatkowe punkty można zdobywać za szybkie wykonanie pewnych zadań, które są szczególnie wymagające obliczeniowo albo mają większy priory-



Rysunek 4.8: Monitorowanie postępu obliczeń poprzez interfejs aplikacji *Folding@home*

te ze względu na wartość naukową. Dzięki włączeniu do programu pod jednym loginem wielu swoich maszyn, można gromadzić więcej punktów. Ogólnym założeniem systemu punktowego jest motywowanie ochotników do coraz większego zaangażowania i włączania znajomych do wspólnej zabawy łączonej z grywalizacją. 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, dającym warstwę komunikacji z uruchomioną na lokalnej maszynie aplikacją kliencką wykonującą obliczenia. Twórcy platformy starają się ułatwić rozprzestrzenianie się informacji o możliwości włączania się do naukowego projektu dając możliwość udostępniania informacji o programie poprzez portale społecznościowe takie jak Facebook czy Twitter.



Rysunek 4.9: Zachęcanie do udostępniania informacji o programie *Folding@home* w sieciach społecznościowych

4.7. Web Workers

Podrozdział napisany jest na bazie [11].

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 w silnikach przeglądarek internetowych to Java Script. Java Script to jednowątkowy język programowania, co uniemożliwia wykonywanie wielu zadań równolegle. Zlecając więc skryptowi intensywne obliczenia, na cały czas ich trwania, interfejs użytkownika strony internetowej byłby zawieszany i nie odpowiadałby na akcje użytkownika. Takie zachowanie byłoby

nie do przyjęcia dla użytkownika strony. Przeglądarki bronią użytkownika przed takim zachowaniem skryptów na stronie i czasami zdarza się jeszcze zobaczyć okno z ostrzeżeniem, że skrypt przestał odpowiadać i z możliwością manualnego jego zatrzymania.

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

Web Workery nie mają dostępu do struktury DOM, obiektu *window* ani *document*. Zewnętrzne skrypty wykorzystywane przez workera muszą być serwowane z tej samej domeny co kod workera. Ograniczenia to nazywane jest fachowo *Same-origin policy*.

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

5. Model i proponowane rozwiązanie

W rozdziale tym przedstawiono informacje dotyczące zastosowanego rozwiązania. Opisane zostały przyjęte założenia dotyczące modelu, matematyczny opis tego modelu oraz szczegóły zastosowanego sposobu optymalizacji. Na końcu rozdziału przedstawiona została architektura rozwiązania wraz z opisem powstałej aplikacji końcowej.

5.1. Model narciarza i środowiska

W zaproponowanym rozwiążaniu zostały podjęte pewne decyzje odnośnie reprezentacji środowiska oraz narciarza.

Zostało przyjęte, że stok traktowany jest jako płaszczyzna, która jest nachylona do powierzchni ziemi pod kątem pod określonym przez stałą α . Założenie co do płaskiej powierzchni stoku jest tylko ograniczeniem przyjętym do testów. Umożliwia to łatwiejszą analizę wyników, niezaburzonych zmianami nachylenia terenu. Jednak stworzony program może zostać w prosty sposób zmodyfikowany tak, aby zamodelować również bardziej skomplikowaną powierzchnię.

Narciarz traktowany jest jako punkt materialny o masie m .

5.2. Opis matematyczny modelu

Oznaczenia:

- m - masa
- g - przyspieszenie ziemskie
- α - kąt nachylenia stoku

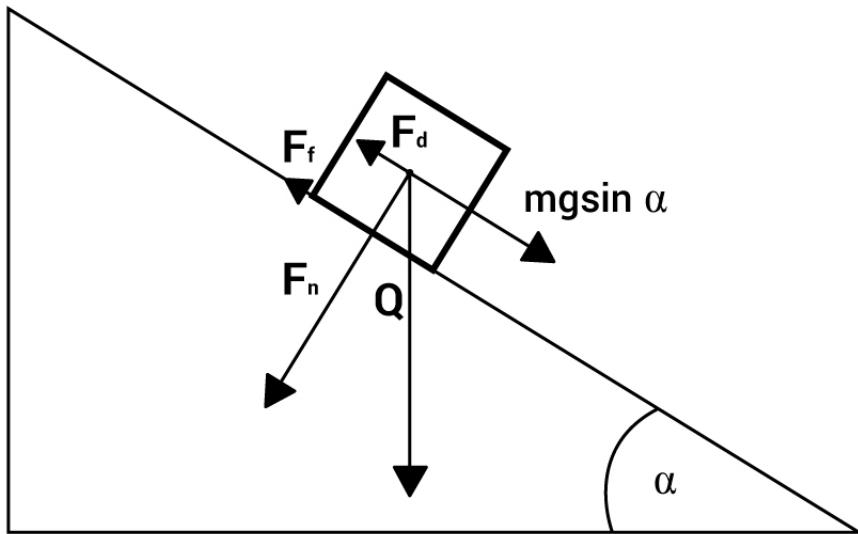
Siła grawitacji działająca na obiekt o masie m :

$$Q = mg \quad (5.1)$$

Siła ta może zostać rozłożona na dwie składowe względem powierzchni stoku:

- równoległą, wynoszącą Q_a ,
- prostopadłą F_n .

Narciarz poruszający się w dół stoku jest modelowany jako punkt materialny poruszającego się po równej pochyłej. Rozkład sił w takim przypadku rozrysowany jest na rysunku 5.1.



Rysunek 5.1: Rozkład sił działających na narciarza przemieszczającego się na równej pochyłej

Składowa siły grawitacji równoległa do powierzchni stoku jest równa:

$$Q_a = mg \sin \alpha \quad (5.2)$$

Siła Q_a jest siłą ściągającą narciarza w dół stoku.

Składowa siły grawitacji prostopadła do powierzchni stoku, czyli nacisk jaki narciarz wywiera swoją masą na powierzchnię stoku, jest równa:

$$F_n = mg \cos \alpha \quad (5.3)$$

Wartość tej siły wpływa na siłę tarcia działającą na narciarza:

$$F_f = \mu F_n = \mu mg \cos \alpha \quad (5.4)$$

Oprócz siły tarcia uwzględniamy również siłę oporu powietrza. Zależy ona od prędkości poruszania się obiektu. Prędkość będziemy wyrażać jako pierwszą pochodną położenia, oznaczaną przez \dot{x} :

$$F_d = k \dot{x}^2 \quad (5.5)$$

$$k = \frac{1}{2} C \rho A \quad (5.6)$$

gdzie:

C - współczynnik oporu

ρ - gęstość powietrza

A - powierzchnia narciarza prostopadła do kierunku ruchu

Więcej o sile oporu powietrza jest opisane w sekcji 2.2 na stronie 10.

5.3. Numeryczne rozwiązywanie problemu

Z drugiej zasady dynamiki Newton'a wiemy, że:

$$m\vec{a} = \sum_i \vec{F}_i \quad (5.7)$$

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

$$m\vec{a} = \vec{Q}_a + \vec{F}_f + \vec{F}_d \quad (5.8)$$

A zatem, uwzględniając kierunek sił otrzymujemy:

$$ma = Q_a - F_f - F_d \quad (5.9)$$

Wypadkowa siła działająca na narciarza jest sumą sił: ściągającej oraz oporów (tarcia i oporu powietrza).

Korzystając z wcześniejszych równań 5.2, 5.4 i 5.5 opisujących siły Q_a , F_f oraz F_d otrzymujemy:

$$ma = mg \sin \alpha - \mu mg \cos \alpha - k\dot{x}^2 \quad (5.10)$$

Wyraźmy teraz przyspieszenie jako drugą pochodną przemieszczenia:

$$a = \ddot{x} \quad (5.11)$$

Podstawiając do równania:

$$m\ddot{x} = mg \sin \alpha - \mu mg \cos \alpha - k\dot{x}^2 \quad (5.12)$$

Zatem po podzieleniu przez m:

$$\ddot{x} = g \sin \alpha - \mu g \cos \alpha - \frac{k}{m} \dot{x}^2 \quad (5.13)$$

Jest to równanie różniczkowe zwyczajne drugiego rzędu. Aby rozwiązać to równanie numerycznie, przedstawiemy to równanie jako układ równań różniczkowych zwyczajnych rzędu pierwszego. W tym celu wprowadzamy nową zmienną v (odpowiadającą prędkości) będącą pierwszą pochodną przemieszczenia, a drugą pochodną zastępujemy pierwszą pochodną prędkości:

$$\begin{cases} v = \dot{x} \\ \ddot{x} = \dot{v} \end{cases} \quad (5.14)$$

Zatem wykorzystując powyższe równania otrzymujemy następujący układ równań:

$$\begin{cases} \dot{v} = g \sin \alpha - \mu g \cos \alpha - \frac{k}{m} v^2 \\ v = \dot{x} \end{cases} \quad (5.15)$$

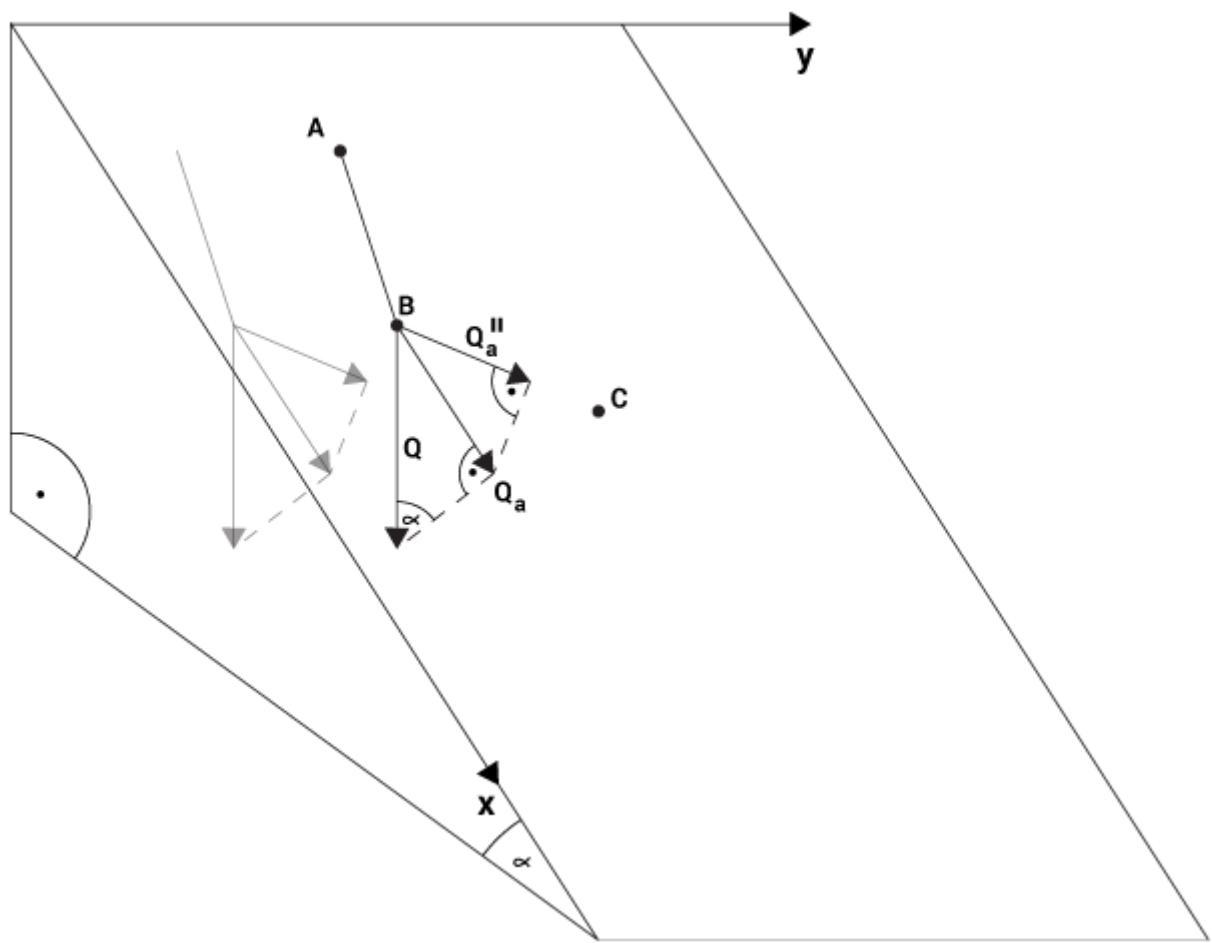
W wielu językach programowania dostępne są funkcje biblioteczne, które potrafią znaleźć rozwiązanie układu równań różniczkowych zwyczajnych pierwszego rzędu, takich jak nasz układ 5.15. W celu jego rozwiązania została wykorzystana funkcja dopri (ang. *Numerical integration of ODE using Dormand-Prince RK method*) z biblioteki Numeric Javascript.

5.3.1. Rozwiązanie na płaszczyźnie stoku

Powyższy układ równań (5.15) opisuje poruszanie się punktu materialnego po równej pochyłej. Jednak w przypadku narciarza przemieszczającego się po stoku musimy uwzględnić również możliwość poruszania się w poprzek stoku, a nie tylko w dół.

Założymy, że narciarz porusza się po torze będącym łamaną. W dalszej części pracy zostanie pokazane, że takie ograniczenie może być dobrym sposobem na przybliżenie rzeczywistego toru jazdy, które jednocześnie znacznie ułatwia sterowanie ruchem narciarza.

Rozpatrzmy teraz jak będzie wyglądał układ sił działających na narciarza (5.2):



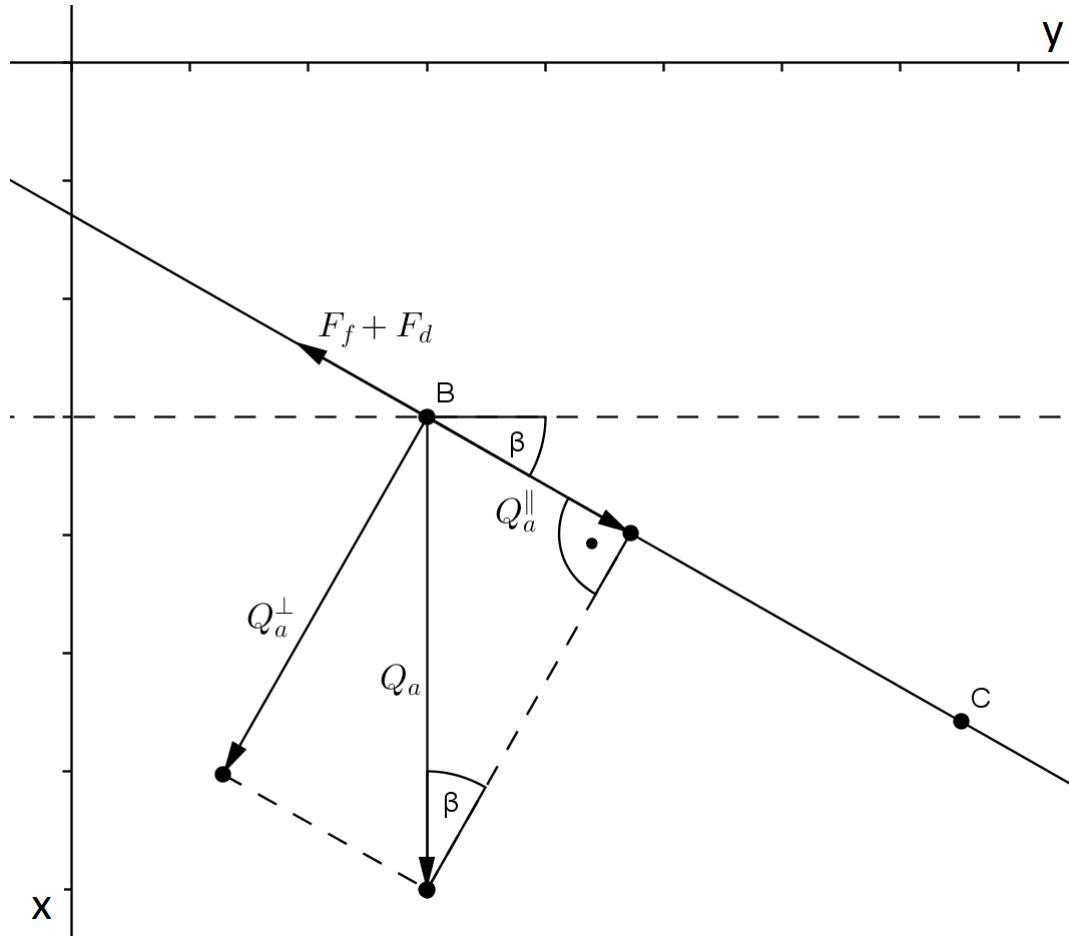
Rysunek 5.2: Rozkład wybranych sił działających na narciarza przemieszczającego się na płaszczyźnie stoku

Na płaszczyźnie stoku zostały oznaczone trzy punkty - A, B i C, przez które kolejno przejeżdża narciarz. Zakładając, że narciarz znajduje się w punkcie B, oznaczona została siła grawitacji Q , działająca na niego i skie-

rowana pionowo (w stosunku do powierzchni ziemi). Składowa tej siły - Q_a , jest rzutem na powierzchnię stoku i jest równa (jak w przypadku równania 5.2):

$$Q_a = mg \sin \alpha \quad (5.16)$$

α jest kątem nachylenia płaszczyzny stoku. Po zrzutowaniu siły Q_a na odcinek BC otrzymujemy siłę Q_a^{\parallel} - jej kierunek jest zgodny z kierunkiem poruszania się narciarza. Pozostałe składowe sił Q oraz Q_a są równoważone przez siłę reakcji podłoża wywieraną na narciarza.



Rysunek 5.3: Rozkład wybranych sił działających na narciarza przemieszczającego się na płaszczyźnie stoku

Na rysunku 5.3 przedstawiona została płaszczyzna stoku z punktami B i C oraz siłami Q_a oraz jej składowymi Q_a^{\parallel} oraz Q_a^{\perp} . Kąt β jest kątem pomiędzy wektorem przemieszczenia a poziomą linią na płaszczyźnie stoku. Wartość siły ściągającej wynosi w tym przypadku:

$$Q_a^{\parallel} = mg \sin \alpha \sin \beta \quad (5.17)$$

Zaznaczone została również siła stanowiąca sumę sił oporów $F_f + F_d$. Zatem nasze równanie ruchu 6.1 będzie wyglądało następująco:

$$\ddot{x} = g \sin \alpha \sin \beta - \frac{F_f + F_d}{m} \quad (5.18)$$

Wykorzystując wzory na siły oporów (5.4 oraz 5.5) otrzymamy:

$$\ddot{x} = g \sin \alpha \sin \beta - \frac{\mu F_n + k \dot{x}^2}{m} \quad (5.19)$$

gdzie F_n to siła nacisku narciarza, która pozostaje taka sama jak w przypadku równi pochyłej (5.3):

$$F_n = mg \cos \alpha \quad (5.20)$$

Zatem:

$$\ddot{x} = g \sin \alpha \sin \beta - (\mu g \cos \alpha + \frac{k}{m} \dot{x}^2) \quad (5.21)$$

W naszym rozwiążaniu zastosowany został zaznaczony na rysunku układ współrzędnych XY. Powyższe równanie zapiszmy teraz uwzględniając przemieszczanie się narciarza w oznaczonych kierunkach - wzdłuż i wszerz stoku. Otrzymamy wtedy następujące równania:

$$\begin{cases} \ddot{x}_x = (g \sin \alpha \sin \beta - (\mu g \cos \alpha + \frac{k}{m} \dot{x}^2)) \sin \beta \\ \ddot{x}_y = (g \sin \alpha \sin \beta - (\mu g \cos \alpha + \frac{k}{m} \dot{x}^2)) \cos \beta \end{cases} \quad (5.22)$$

Po wprowadzeniu jak poprzednio dodatkowych zmiennych, aby zredukować równania do równań różniczkowych zwyczajnych pierwszego rzędu, wprowadzamy dodatkowe zmienne, w tym wypadku prędkości v_x i v_y oraz pamiętamy o zależności między nimi a pochodną przemieszczenia:

$$\begin{cases} v_x = \dot{x}_x \\ v_y = \dot{x}_y \\ \dot{x} = \sqrt{v_x^2 + v_y^2} \end{cases} \quad (5.23)$$

Dodatkowo można zauważyć, że jeśli v_x stanowi pierwszą pochodną x_x to prawdziwe są także poniższe równania:

$$\begin{cases} \dot{v}_x = \ddot{x}_x \\ \dot{v}_y = \ddot{x}_y \end{cases} \quad (5.24)$$

Wprowadzając te informacje do układu równań 5.22, otrzymujemy:

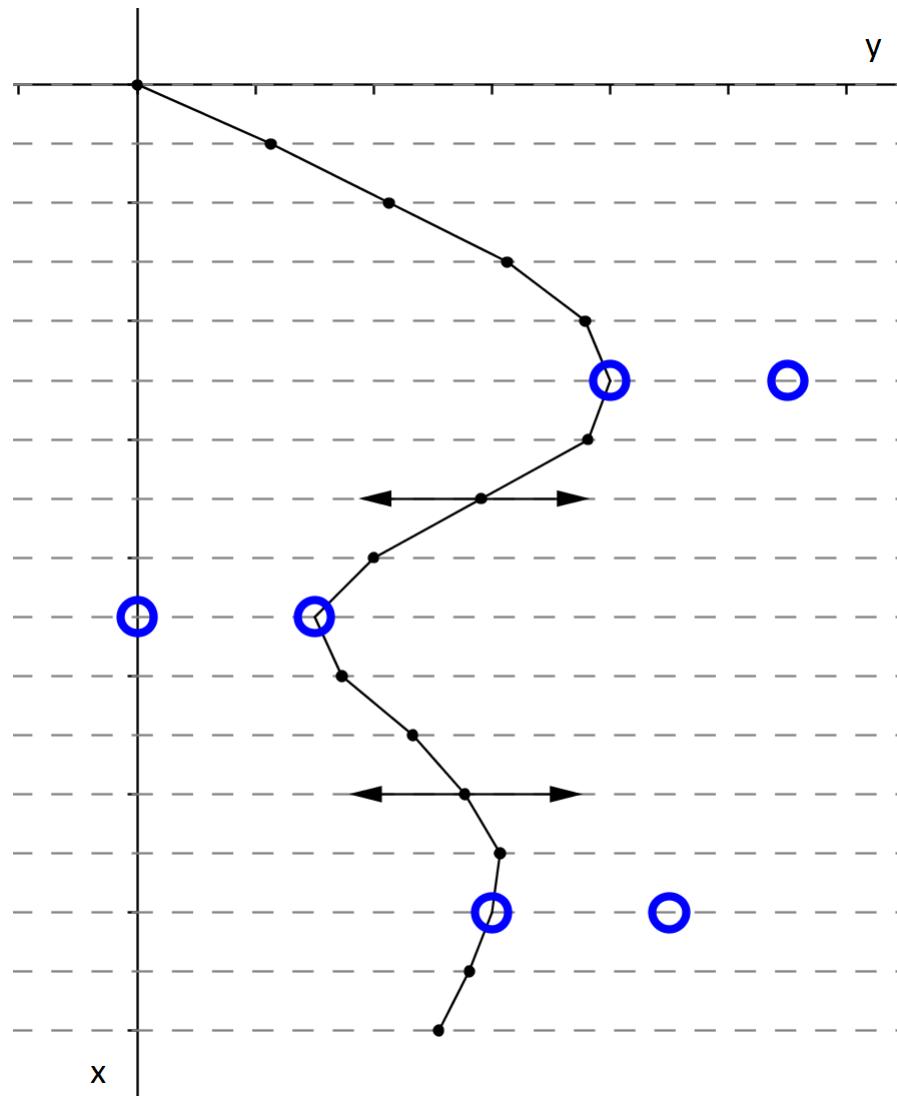
$$\begin{cases} v_x = \dot{x}_x \\ v_y = \dot{x}_y \\ \dot{v}_x = \ddot{x}_x = (g \sin \alpha \sin \beta - (\mu g \cos \alpha + \frac{k}{m} \dot{x}^2)) \sin \beta \\ \dot{v}_y = \ddot{x}_y = (g \sin \alpha \sin \beta - (\mu g \cos \alpha + \frac{k}{m} \dot{x}^2)) \cos \beta \end{cases} \quad (5.25)$$

Taki układ równań różniczkowych można rozwiązać wspomnianą wcześniej funkcją *dopri*.

Zainteresowani modelowaniem ruchu narciarza i optymalizacją toru jazdy mogą przeczytać również pracę [9].

5.4. Optymalizacja toru przejazdu

Aby znaleźć rozwiązanie problemu optymalizacji, należy przyjąć jakiś sposób reprezentacji każdego z rozwiązań. W rzeczywistości tor przejazdu narciarza to ślad, który pozostawiają narty na śniegu w trakcie przemieszczania się po stoku. Jak opisano w podrozdziale 5.3.1, w celu uproszczenia sposobu przemieszczania się narciarza, zdecydowano, że będzie się on poruszać po łamanej. Zatem do reprezentacji rozwiązania można przyjąć zbiór punktów, przez które kolejno przejeżdża narciarz, poruszając się między tymi punktami wyłącznie po linii prostej.



Rysunek 5.4: Sposób reprezentacji rozwiązań problemu

W zaproponowanym rozwiążaniu narzucamy z góry, co ile metrów w pionie stoku ma znajdować się punkt, przez który będzie musiał przejechać narciarz. Można wyobrazić to sobie jako zbiór poziomych linii, z której każda wyznacza możliwe położenie pojedynczego punktu. Na rysunku 5.4 zostały przedstawione te linie oraz bramki (kolor niebieski). Strzałki pokazują jak można przesuwać każdy z tych punktów w celu zmiany trasy przejazdu i otrzymania nowego rozwiązania. Zostało narzucone, że narciarz musi przejechać jak najbliżej każdej wewnętrznej części bramki, co oznacza dołożenie punktów w tym miejscu. Ogranicza to znacznie liczbę rozwiązań, które należy sprawdzić i dzięki temu przyspiesza działanie algorytmu. Takie założenie bazuje również na doświadczeniu narciarzy z jazdy po slalomach, gdyż w większości przypadków przejeżdżanie tuż przy bramkach jest najkorzystniejsze. Zdarza się oczywiście, że np. dla tzw. bramek otwartych opisanych w rozdziale 2.1 często lepiej wybrać trochę inną trasę. W tych przypadkach konieczne jest zrezygnowanie z takiego ograniczenia, jednak należy liczyć się z tym, że czas poszukiwania najlepszego rozwiązania wydłuży się, zwłaszcza jeśli takich miejsc jest więcej.

Zatem pozostaje określić, w jaki sposób możemy stwierdzić, że dane rozwiązanie jest najlepsze. W tym przypadku chcemy, aby narciarz w jak najkrótszym czasie dotarł do mety prawidłowo przejeżdżając przez wszystkie bramki. Mając dane rozwiązanie w postaci punktów wyznaczających łamana, obliczamy ile czasu zajmie narciarzowi przejechanie po tej trasie. Im mniejsza wartość tym rozwiązanie jest lepsze. Zatem funkcja celu dla tego

problemu mając na wejściu ciąg punktów wyznaczających trasę przejazdu, zwraca czas potrzebny na jej pokonanie.

Algorytm ewolucyjny Opisana w poprzednim podrozdziale reprezentacja rozwiązania to w zastosowanym algorytmie ewolucyjnym pojedynczy osobnik, a punkty składające się na to rozwiązanie, aściślej, ich położenie w pozycji poziomej, określają genotyp każdego osobnika. Ponieważ korzystamy ze strategii ewolucyjnych, nie wprowadzamy tu kodowania binarnego, pozycja każdego punktu jest zapamiętywana jako wartość rzeczywista.

Dodatkowo do genotypu wchodzi także zestaw parametrów σ , które w strategiach ewolucyjnych używane są podczas mutacji, tak jak opisano to w rozdziale 3.2 w części "Typy algorytmów ewolucyjnych" w opisie strategii ewolucyjnych. Każdemu punktowi przypisana jest osobna wartość σ - reprezentująca odchylenie standardowe wartości y tego punktu.

Zastosowany algorytm wykorzystuje strategię $(\mu + \lambda)$ opisaną w rozdziale 3.2. Jako początkową populację wybieramy losowe osobniki - wartości y punktów są ograniczone jedynie przez wartości y położenia dwóch najbliższych bramek. Jest to kierowane koniecznością zadania o szybsze znalezienie rozwiązania - zbyt duże odległości można z góry odrzucić opierając się na doświadczeniach z rzeczywistej jazdy narciarza po slalomie. Wielkość populacji bazowej μ jest jednym z parametrów programu, wartość ta wynosi w testach 30 - 60. Wartość parametru λ także jest parametrem, w testach użyto wielkości 100 - 200. Wartości te porównywalne są z sugerowanymi w literaturze. [2]

Szkielet algorytmu zgodny jest z zastosowaną strategią - po wylosowaniu z istniejącej populacji populacji tymczasowej o wielkości λ , dokonuje się na jej osobnikach operacji genetycznych, najpierw krzyżowania, a następnie mutacji na osobnikach otrzymanych z krzyżowania. Kolejnym krokiem jest ocenienie nowych osobników i wybór spośród nich oraz populacji początkowej osobników o najlepszym przystosowaniu, tak aby one stanowiły nową populację bazową.

Krzyżowanie

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

Mutacja

Po krzyżowaniu mamy znowu w populacji tymczasowej λ 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 i tak te punkty należą do rozwiązania najlepszego.

Warunek zakończenia

Wybór warunku 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 dobrego rozwiązania, ale w połączeniu z dodatkowymi mechanizmami może być skuteczną metodą podjęcia decyzji o zakończeniu optymalizacji.

W rozwiązaniu brany jest 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 najlepszego. Oczywiście sterując liczbą iteracji, przez które sprawdzamy zmiany, oraz wiel-

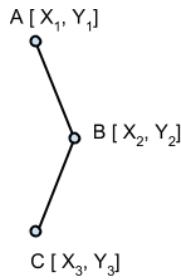
kością rozrzutu populacji możemy znajdować lepsze lub gorsze rozwiązania kosztem wydłużenia lub skrócenia czasu obliczeń.

Hill climbing Problem optymalizacji trasy narciarza można rozwiązywać stosując algorytm ewolucyjny, jednak problemem może być długi czas wykonywania się programu. Słaba poprawa wyników może występować zwłaszcza w końcowej fazie działania. Widoczne są wtedy niepotrzebne próby przeszukiwania zbyt odległych rozwiązań, a jednak wciąż znalezione rozwiązanie nie jest jeszcze tak dobre, jak można by tego oczekiwac. Wiedząc, że rozwiązanie jest już dosyć bliskie najlepszemu można z dużym prawdopodobieństwem założyć, że wystarczy znaleźć rozwiązanie lokalnie optymalne, aby było ono satysfakcyjne. Oczywiście nie mamy pewności, że będzie to rozwiązanie globalnie optymalne, ale najczęściej takiej pewności mieć nie możemy. Problemem może wciąż być jednak decyzja kiedy należy przejść na algorytm lokalnej optymalizacji.

Zastosowanie algorytmu lokalnej optymalizacji powinno pomóc w końcowej fazie poszukiwań. Z tego powodu użyty został algorytm Hill climbing opisany w rozdziale 3.4. 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 znaczących rezultatów, są mniejsze niż narzucony parametr ϵ , algorytm zatrzymuje swoje działanie. Wartość ϵ wynosi przeważnie w testach 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 (odpowiada to wielkości 0.5 metra). Zmiana ta wynika z założenia, że aby znaleźć rozwiązanie jak najlepsze, nie trzeba zmieniać wartości y żadnego z punktów o więcej niż wybrane 0.5 m.

5.5. Modelowanie karania

W celu jak najdokładniejszego zamodelowania zmian prędkości podczas poruszania się po łamanej, testowany był szereg modeli. Konieczność wprowadzenia takich mechanizmów staje się oczywista, gdy uświadomimy sobie, że im mniejszy jest kąt pomiędzy kolejnymi odcinkami pokonywanej łamanej, tym bardziej jest to kosztowne w realnej sytuacji pod względem utraty prędkości. Dla zobrazowania sytuacji, rozważmy przejście ze stanu jazdy w linii spadku stoku do jazdy prostopadłej do linii spadku stoku. Aby tego dokonać narciarz musi przyhamować do zera. Z kolei im kąt jest bardziej zbliżony do 180 stopni tym mniejszy jest efekt przyhamowania, gdyż w realnym przypadku, zmiana kierunku jazdy jest wykonywana w płynnym łuku.



Rysunek 5.5: Fragment łamanej jako toru przejazdu

Strategia sumowania pochodnych Pierwsza strategia opiera się na obliczaniu kolejno przybliżenia drugich pochodnych funkcji za pomocą ilorazów różnicowych w każdym kolejnym punkcie trasy a następnie zsumowaniu ich bezwzględnych wartości.

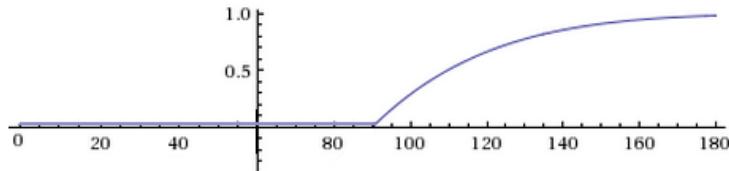
$$\ddot{d}_i = \frac{x_1 + x_3 - 2x_2}{(y_3 - y_2)(y_2 - y_1)} \quad (5.26)$$

$$k = \sum_{i=1}^{n-1} |\ddot{d}_i|^3 \quad (5.27)$$

Powstałą liczbę k można traktować jako współczynnik, którego mała bezwzględna wartość oznacza bardzo łagodne kąty przejścia na całej trasie natomiast im większa „kanciastość” trasy tym większa wartość tego współczynnika. Dodatkowo, przed zsumowaniem, wartość bezwzględna przybliżonej drugiej pochodnej jest podnoszona do trzeciej potęgi. Wyliczony współczynnik dla każdej z rozważanych w algorytmie tras jest mnożony przez obliczony czas przejazdu tej trasy. Dopiero taka liczba jest traktowana jako wartość funkcji celu algorytmu genetycznego. Minusem tej strategii jest przeskakowanie realnego czasu przejazdu narciarza po zadanym torze, a zatem brak możliwości porównania go do wyników uzyskiwanych innymi metodami.

Strategia proporcjonalnego zmniejszania wartości wektora prędkości W tej strategii, dla każdego wierzchołka łamanej, obliczamy indywidualny współczynnik kary m z zakresu od 0 do 1. Długość wektora prędkości w każdym kolejnym punkcie będzie mnożona przez współczynnik dla tego punktu. Modelowane będzie zatem przyhamowywanie - tym większe, im większy jest kąt pomiędzy kolejnymi punktami łamanej. Współczynnik w danym punkcie zależy od kąta ABC z rysunku 5.6 pomiędzy poprzednim a następującym punktem. Zaproponowana została następująca funkcja, zmierzająca do zera dla kąta 90 stopni i mniejszego, i zmierzająca do jedynki dla 180 stopni.

$$m(\alpha) = \begin{cases} 0.01, \alpha \in (0, 90) \\ 1 - \left(\frac{\alpha}{180} - 1.5\right)^6, \alpha \in [90, 180) \end{cases} \quad (5.28)$$



Rysunek 5.6: Wykres funkcji

Strategia karania za każdą zmianę krawędzi Ostatnia strategia przyjmuje, że nie ma potrzeby wykonywać dodatkowych skrętów pomiędzy bramkami. Choć teoretycznie jest możliwe, by taki dodatkowy skręt pozwalał uzyskać lepszy czas, to jednak jest to niespotykane podczas rzeczywistej jazdy po slalomach. Dlatego w strategii tej zlicza się liczbę zmian kierunku jazdy na testowanym torze i porównuje ją z ilością bramek wymuszających skręt na trasie. Otrzymujemy zatem liczbę nadmiarowych zmian krawędzi, które w każdym przypadku powodują gorszy czas przejazdu niż gdyby ich nie było. Aby to zatem zamodelować, w tej strategii dodajemy do wyliczonego czasu przejazdu stałą liczbę sekund kary za każdą niepotrzebną zmianę krawędzi i to traktujemy jako wartość funkcji celu algorytmu ewolucyjnego.

5.6. 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 wszystkie potrzeby.

5.6.1. Architektura prototypowa

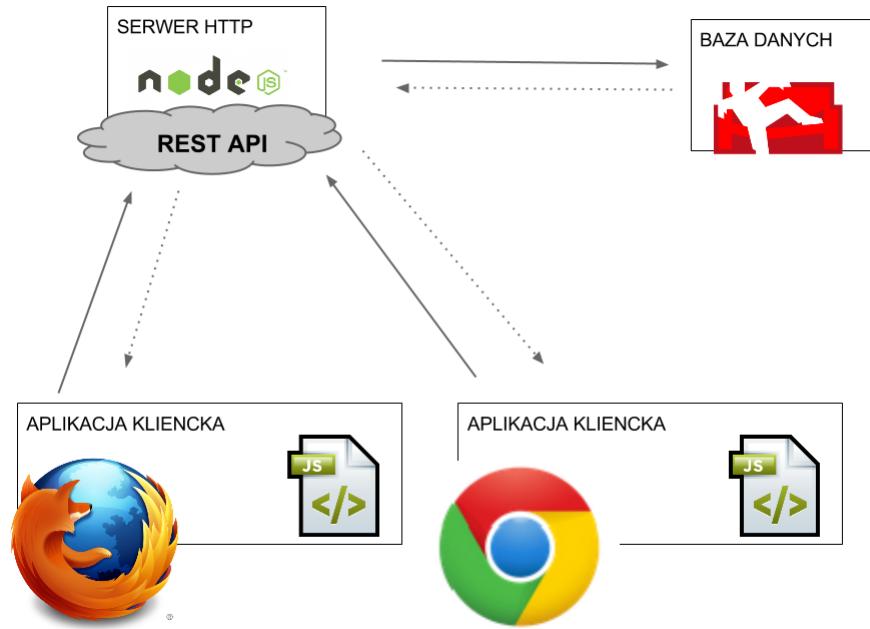
Pierwszym wybranym przez nas środowiskiem tworzenia systemu był język Python i dostępne dla niego moduły: VPython umożliwiający wizualizację w 2D i 3D oraz biblioteki numeryczne NumPy i SciPy wykorzystywane do rozwiązywania równań różniczkowych.

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 udokumentowane 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. Stanowiło to poważny problem, 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. Wiedziałyśmy, że warunkiem koniecznym by zrealizować tę wizję jest łatwość w dołączaniu do obliczeń, wizualizacji wyników i bezproblemowa konfiguracja. Biorąc to pod uwagę, zdecydowałyśmy, że potrzebujemy innego środowiska.

5.6.2. 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 egzemplarz problemu, który jest następnie lokalnie przetwarzany i wizualizowany klientowi na bieżąco. Po skończonym obliczeniu, rozwiązanie jest umieszczane na serwerze, a klient może przejść do obliczania kolejnego lub jeszcze raz tego samego problemu.

Architektura naszego systemu składa się z dwóch głównych, niezależnych części składowych. Pierwszą z części jest aplikacja kliencka przeprowadzająca obliczenia i wizualizację oraz komunikująca się z serwerem zarządzającym obliczeniami. Drugim jest aplikacja *lekkiego* serwera http, jako warstwa wystawiająca REST API. Warstwą persistencką jest dokumentowa, nierelacyjna baza danych CouchDB. Schemat architektury prezentuje rysunek 5.7.



Rysunek 5.7: Uproszczony schemat architektury docelowej

5.6.3. Aplikacja kliencka

Aplikacja kliencka to aplikacja webowa stworzona przy użyciu frameworku Chaplin, wprowadzającego dobre praktyki w strukturyzowaniu aplikacji Java Script'owych wykorzystujących bibliotekę Backbone.js. Aplikacja w całości napisana jest w języku Coffee Script. Jest to język programowania ze składnią podobną do składni Python'a, kompilujący się do Java Script'u. Powstały po komplikacji kod Java Script jest czytelny i przyjazny do debugowania. Programując w Coffee Script można używać wszystkich bibliotek i modułów napisanych w Java Script, co daje wiele możliwości z powodu szerokiego zakresu dostępnych bibliotek w tym języku. Zdecydowanie możemy się używać tego języka przede wszystkim z uwagi na składnię przypominającą dobrze znany nam język Python, większą przejrzystość i czytelność kodu niż Java Script, jednocześnie zachowując szerokie możliwości tego języka.

Aplikacja ma zasadniczo trzy warstwy:

- warstwę prezentacyjną i wizualizacyjną dla rozwiązywanego problemu,
- warstwę komunikacji z serwerem,
- warstwę obliczeniową.

Komunikacje pomiędzy poszczególnymi warstwami aplikacji klienckiej w postaci diagramu sekwencji przedstawia Diagram 5.8 na stronie 46.

Warstwa prezentacji

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

- wyjaśnienie użytkownikowi jaki problem jest rozwiązywany oraz jak może pomóc,

- umożliwienie rozpoczęcia wzięcia udziału w obliczeniu,
- prezentowanie trasy narciarskiej (układu bramek), pobranej z serwera do obliczeń,
- wizualizacja na bieżąco wyników obliczeń w postaci rozważanego jako najlepszy toru przejazdu,
- wizualizacja statystyk algorytmu obliczającego w postaci wykresów.

Warstwa komunikacyjna

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

- pobieranie instancji problemu do obliczenia,
- wysyłanie na serwer rozwiązania zadanego problemu,
- pobierania z serwera informacji o obecnie najlepszym rozwiązaniu tego problemu.

Warstwa obliczeniowa

Zastosowanie w naszej architekturze Web Worker’ów do wykonywania obliczeń jest dobrze uzasadnione, biorąc pod uwagę ich specyfikację, opisaną szerzej w rozdziale 4.7. 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 otrzymywana trasę.

5.6.4. 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 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 5.9 na stronie 47 pokazuje sekwencję komunikacji pomiędzy komponentami systemu oraz warstwą persystencji.

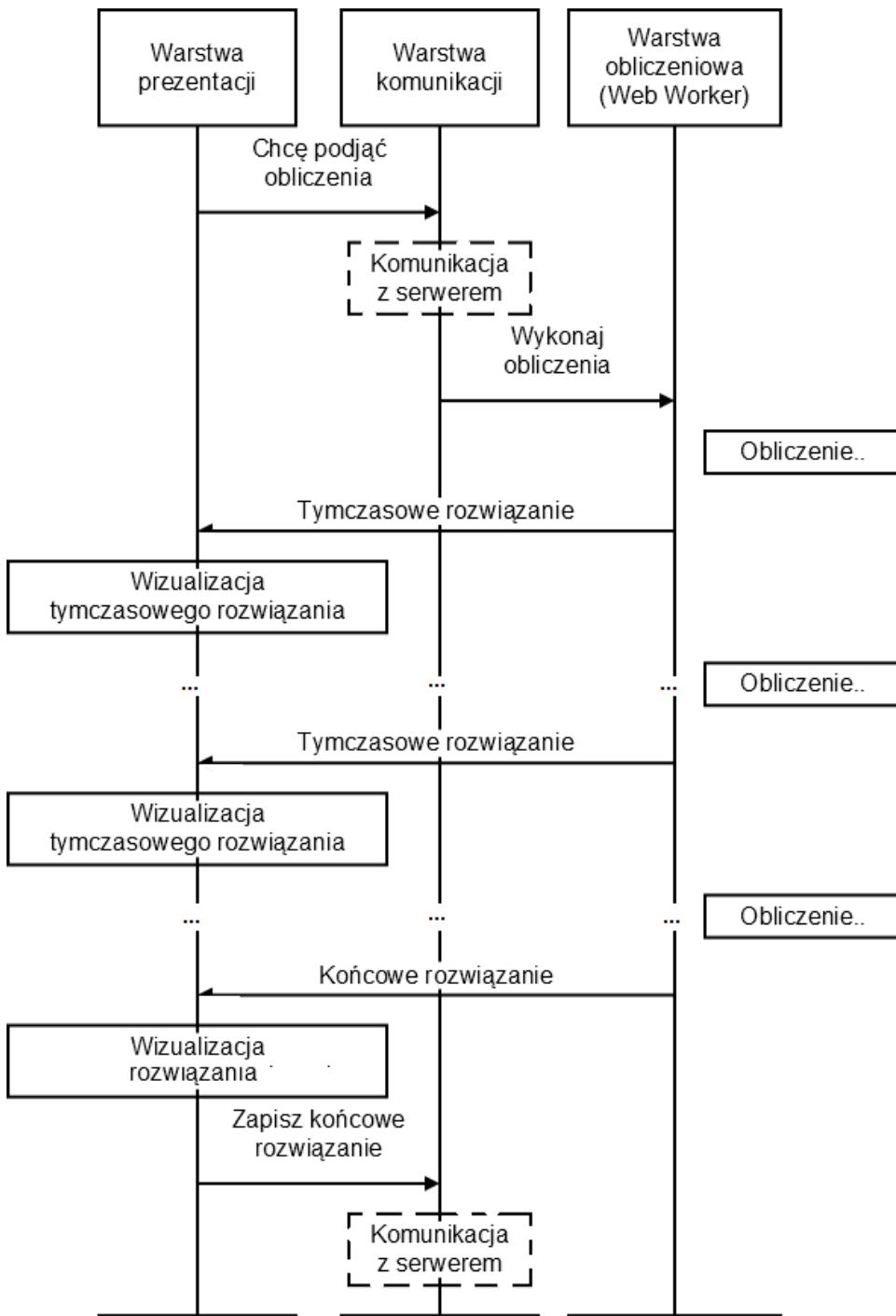
5.6.5. Interfejs systemu udostępniony użytkownikowi

Efektem widocznym dla końcowego użytkownika powstałą na potrzeby tej pracy aplikacji jest strona www, dostępna pod adresem URL - <http://giant-client.herokuapp.com/>. Na stronie, poza krótkim wytlumaczeniem celu projektu i sposobu w jaki każdy odwiedzający może pomóc w obliczeniach, znajduje się jedno wyraźne tzw. “call to action” - duży przycisk zachęcający do rozpoczęcia obliczeń.

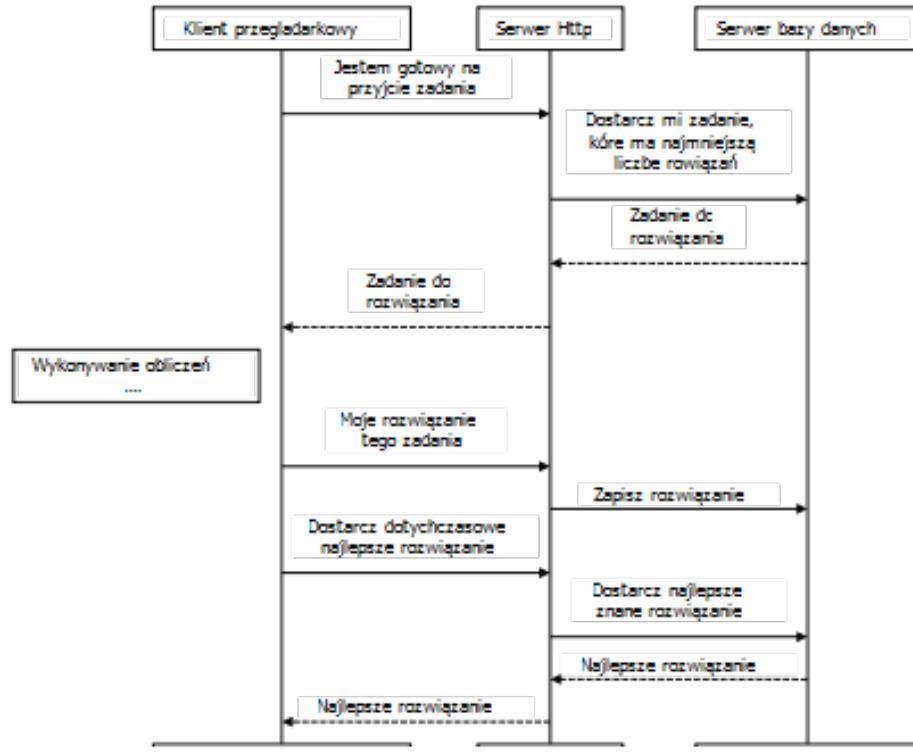
Po uruchomieniu obliczeń osoba odwiedzająca stronę dostaje na bieżąco informacje zwrotne dotyczące efektów jej współpracy. Na stronie:

- rysowana jest cały czas aktualnie rozważana jako najlepsza trasa przejazdu,

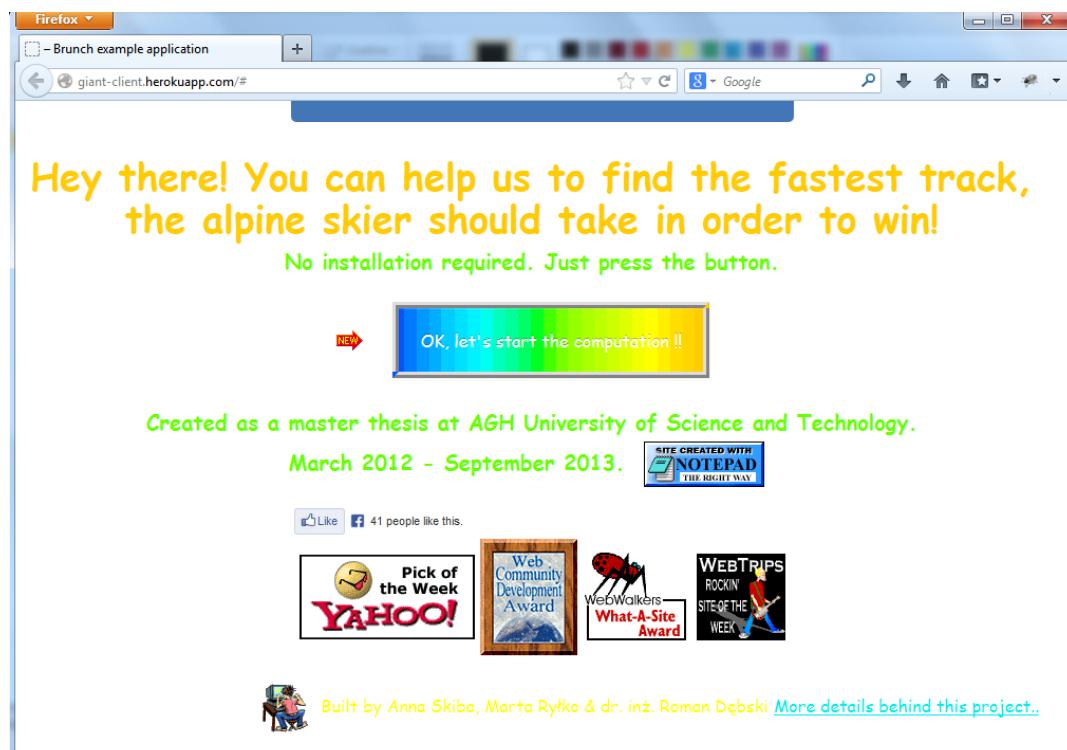
- wyświetla się obliczony czas przejazdu po aktualnie rozważanym torze przejazdu,
- wyświetla się najlepszy, dotychczas znaleziony, czas przejazdu dla aktualnie rozważanej konfiguracji trasy,
- rysowane są wykresy przedstawiające statystyki algorytmu ewolucyjnego (aktualny Fitness populacji - najlepszy, najgorszy oraz średnia dla wszystkich osobników).



Rysunek 5.8: Uproszczony diagram sekwencji pomiędzy poszczególnymi warstwami aplikacji klienckiej. Warstwa obliczeniowa przekazuje do warstwy prezentacji wiele tymczasowych rozwiązań oraz, po zakończeniu pełnego cyklu obliczeń, finalne, najlepsze rozwiązanie, które następnie przekazywane jest na serwer.



Rysunek 5.9: Uproszczony diagram sekwencji pomiędzy poszczególnymi komponentami systemu oraz warstwą persystencji



Rysunek 5.10: Strona główna zachęcającą do uruchomienia obliczeń



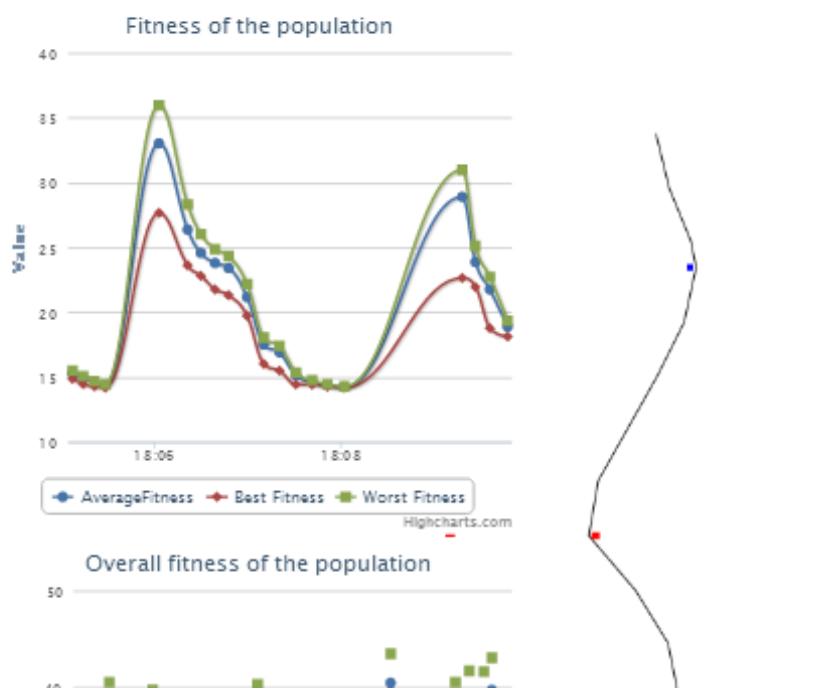
What you see here is a giant slalom and our algorithm that finds the fastest tract for the skier. The problem instance was requested from our server. All the computations are run in your browser and will be send to our server once they are done.

Number of solved problems: 54

Currently you found: 17.24609375

Best time you found: 13.955078125

Best time we have so far: 13.857421875



Rysunek 5.11: Aplikacja w trakcie prowadzenia obliczeń

6. Wyniki

W rozdziale tym przedstawiono informacje na temat wyników przeprowadzonych eksperymentów. Najpierw przedstawione zostały testy zarówno teoretyczne, jak i w praktyce pokazujące poprawność zastosowanego modelu. Następnie pokazane zostały wyniki optymalizacji algorytmu ewolucyjnego, lokalnej optymalizacji oraz mechanizmu karania dla różnych parametrów. Na koniec omówione zostały wyniki dotyczące zastosowanego rozwiązania architektonicznego oraz skuteczności Volunteer Computing.

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 $0^\circ C$ 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
- $k = \frac{1}{2}C\rho A$ - współczynnik oporu powietrza

Opis powyższych stałych można znaleźć w rozdziałach 2.2 oraz 3.2.

W rozdziale zastosowano następujące oznaczenia:

- ES - strategia ewolucyjna (ang. *Evolution Strategy*) - w tym przypadku $(\mu + \lambda)$
- HC - Hill climbing
- PF - funkcja kary (ang. *Punishment Function*)

W eksperymetach wszystkie współrzędne punktów są w układzie kartezjańskim, dwuwymiarowym, zorientowanym na płaszczyźnie stoku. Oś x skierowana jest wzdłuż stoku, natomiast oś y w poprzek - szczegółowy opis znajduje się w rozdziale 5.

6.1. Weryfikacja modelu

W tej sekcji zostało pokazane, że model, który został przyjęty jest prawidłowy. W poniższych eksperymetach zostanie sprawdzone, czy zależności pomiędzy zmianą poszczególnych zmiennych modelu, takich jak masa, nachylenie stoku oraz opory ruchu, a otrzymywany czasem przejazdu, są zgodne z prawami fizyki. Kolejne eksperymenty zweryfikują również, czy czas przejazdu po teoretycznie najszybszej trasie jest podobny do tego, który można wyliczyć ze wzorów przedstawionych w artykułach naukowych. Na koniec zostanie sprawdzone czy możliwe jest przybliżanie jazdy narciarza jazdą po łamanej i jak gęsto muszą być rozmieszczone punkty przegięcia, żeby nie stracić na dokładności wyników.

6.1.1. Podstawowe założenia modelu

Masa i nachylenie stoku

Na początek zostały przeprowadzane proste eksperymenty, które miały udowodnić, że przyjęty model jest prawidłowy. 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.

Przypomnijmy równanie 6.1, które opisuje model poruszania się narciarza:

$$\ddot{x} = g \sin \alpha - \mu g \cos \alpha - \frac{k}{m} \dot{x}^2 \quad (6.1)$$

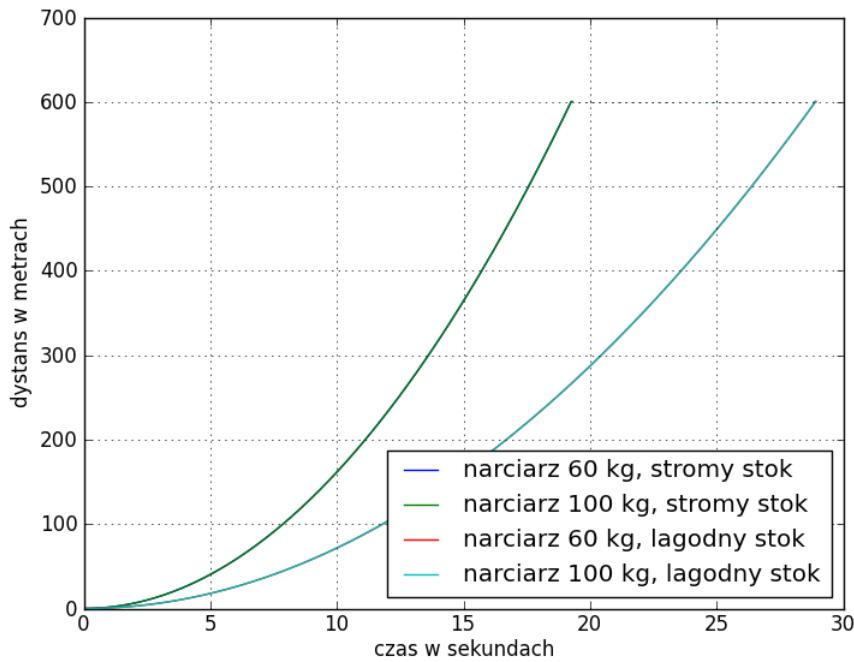
Zauważmy, że masa narciarza wpływa jedynie na wartość przyspieszenia wynikającą z siły oporu powietrza. Natomiast kąt nachylenia α wpływa zarówno na siłę ściągającą, jak i na siłę tarcia. W opisany eksperymencie zostanie pokazane, że:

- 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 mniejsze jest opóźnienie wynikające z siły oporu powietrza, co skutkuje szybszym przejazdem narciarza (przy założeniu, że frontalna powierzchnia narciarza A nie zmienia się).

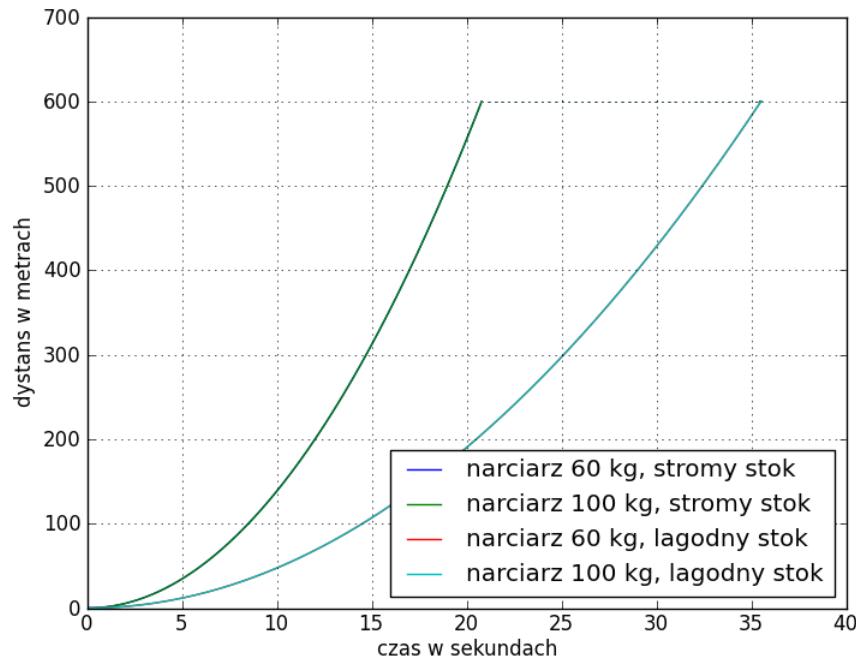
eksperymentu Eksperyment został przeprowadzony na dwóch modelach stoków narciarskich. Oba mają 600 m długości co odpowiada rzeczywistej długości stoku Harenda w Zakopanem. Stoki modelowane są jako równe pochyłe o stałym nachyleniu. Aby zweryfikować dodatkowo prawidłowość modelu, w eksperymencie zostały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

- Harenda - ok. 20° (0.3367 radianów)
- Kotelnica - ok. 8° (0.1470 radianów)

Zostały przeprowadzone trzy próby, których wyniki można zobaczyć odpowiednio na kolejnych wykresach 6.1, 6.2, 6.3. W pierwszej części eksperymentu uwzględniono tylko i wyłącznie siłę grawitacji, w drugiej dołożona została wyłącznie siła tarcia, natomiast w trzeciej uwzględniono wszystkie siły oporu, a więc także siłę oporu powietrza.

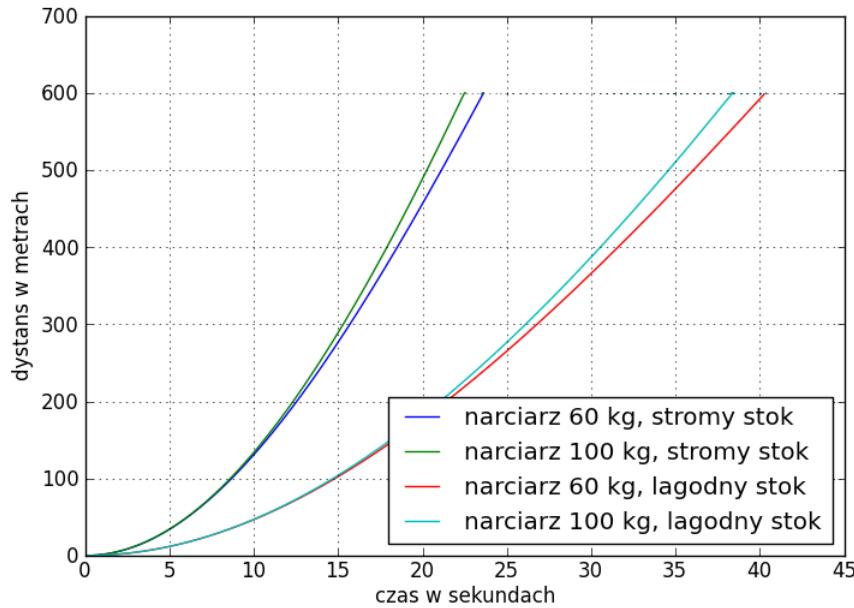


Rysunek 6.1: Porównanie czasu przejazdu narciarzy o różnych masach, na stokach o różnym nachyleniu bez uwzględniania sił oporu. Linie wykresów dla różnych mas narciarzy i tego samego rodzaju stoku pokrywają się.



Rysunek 6.2: Porównanie czasu przejazdu narciarzy o różnych masach, na stokach o różnym nachyleniu z uwzględnieniem siły tarcia jako jedynej siły oporu. Linie wykresów dla różnych mas narciarzy i tego samego rodzaju stoku pokrywają się.

Rezultaty eksperymentu



Rysunek 6.3: Porównanie czasu przejazdu narciarzy o różnych masach, na stokach o różnym nachyleniu z uwzględnieniem zarówno siły tarcia jak i oporu powietrza

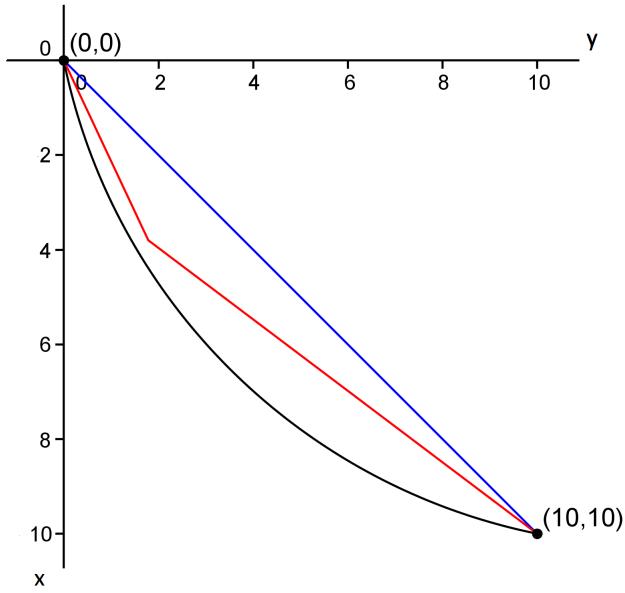
Można zaobserwować, że w każdym z trzech przypadków na stromym stoku narciarze przejeżdżają ten sam dystans w krótszym czasie. W pierwszej i drugiej części eksperymentu nie ma znaczenia masa narciarza - czas przejazdu jest taki sam. Jednak na wykresie 6.1 czas przejazdu wynosi odpowiednio ok. 19 i 28 s., natomiast jeśli uwzględnimy siłę tarcia, czasy te wydłużają się do ok. 21 i 35 s.

Analizując wykres 6.3 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 podczas narciarskich zawodów wynoszą na stoku Harenda średnio ok. 40 s. Wydaje się więc rozsądne, że przejazd na wprost może zajmować ok. 22 s - czyli wartość, która została otrzymana w eksperymencie. W porównaniu z poprzednimi częściami eksperymentu łatwo spostrzec, że w ostatniej części czasu przejazdu są dłuższe - 22-23 oraz 38-40 s.

Czas przejazdu w zależności od toru jazdy

Optymalizacja trasy narciarza polega na znalezieniu linii najszybszego przejazdu. Warto tu zauważyć, że wbrew pozorom, nie zawsze jest to najkrótsza linia - odcinek łączący punkt początkowy z końcowym. W późniejszym eksperymencie zostanie pokazane, jak zmienia się czas przejazdu w zależności od linii poruszania się narciarza.

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, tory te będą różne.



Rysunek 6.4: Przykładowe tory przejazdu od punktu (0,0) do punktu (10,10)

Opis eksperymentu W eksperymencie sprawdzone zostanie, ile czasu zajmuje narciarzowi przejechanie z punktu (0,0) do punktu (10,10). Na rysunku 6.4 pokazane są przykładowe trasy przejazdu pomiędzy tymi punktami. Aby uprościć eksperyment, wyniki sprawdzone zostaną przy pominięciu wkładu wszystkich oporów. Gdyby narciarz poruszał się po linii prostej, bezpośrednio do punktu końcowego, czas przejazdu można wyliczyć przekształcając wzór:

$$s = \frac{at^2}{2} \quad (6.2)$$

i otrzymując:

$$t = \sqrt{\frac{2s}{a}} \quad (6.3)$$

gdzie:

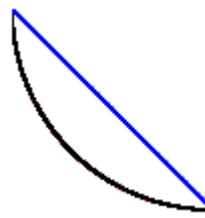
– $s = 10\sqrt{2}$ - odległość między punktem początkowym i końcowym

– $a = g \sin(\frac{\pi}{12}) \sin(\frac{\pi}{4})$ - 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, przyjmując $g = 9.81$, czas przejazdu po linii prostej powinien wynosić około:

$$t \simeq \sqrt{\frac{2 * 10\sqrt{2}}{9.81 * \sin(\frac{\pi}{12}) \sin(\frac{\pi}{4})}} \simeq 3.969 \quad (6.4)$$

Teraz zostanie sprawdzone 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 na rysunku 6.5 widzimy jak wygląda trasa przejazdu w dwóch przypadkach.



Rysunek 6.5: Tory przejazdu w eksperymencie porównującym czas przejazdu dla różnych linii jazdy

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

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 wynika z niedokładności obliczeń numerycznych. Dodatkowym problemem jest fakt, że nie możemy wyliczyć dokładnego czasu potrzebnego narciarzowi na dojście do określonego punktu - możemy jedynie próbować zbliżać się do punktu końcowego, robiąc coraz mniejsze "kroki", starając się znaleźć jak najbliżej niego.

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 uwzględniając dokładność pomiaru, która wynosi jedną setną 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 jednak pojechać dłuższą trasą, ale taką, na której nabierzemy większej prędkości i później zyskamy na kolejnym odcinku.

Pozostaje jeszcze pytanie, jaka jest zatem najszybsza linia przejazdu pomiędzy dwoma punktami. Problem ten został już rozwiązany - jest to fragmentem cykloidy. Dokładniejsze informacje można znaleźć m.in. w [14] oraz [10], a także [8]. W [10] znajduje się wzór na najszybszy 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 s. Jak widać, wynik ten jest lepszy niż otrzymany w naszym eksperymencie - od czasu jazdy po ćwiartce koła o ok. 0.05 s., co nie jest różnicą bardzo dużą, ale znaczącą dla zwycięstwa.

Jazda po łamanej

W kolejnym kroku zostanie sprawdzone, 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, z ilu segmentów składać się będzie łamana.

Opis eksperymentu W eksperymencie zostanie sprawdzone 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 6.1.1).

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

Rezultaty eksperymentu Poniżej znajdują się obrazki (Rysunek 6.6) przedstawiające poszczególne próby przejazdu po łamanej składającej się odpowiednio z 2-6 punktów pośrednich (linia szara) oraz pierwotną linię najszybszego przejazdu (linia niebieska). Liczba punktów na łamanej nie bierze pod uwagę punktu startowego, a jedynie kolejne punkty włącznie z punktem na metie.



Rysunek 6.6: Linie przejazdu w poszczególnych eksperymentach. Linia niebieska to najszybsza linia przejazdu, linia szara to łamana przybliżająca linię niebieską

W tabeli 6.1 przedstawione są uzyskane wyniki w zależności od ilości punktów pośrednich oraz różnica czasu w porównaniu do najkrótszego czasu przejazdu - 3.623 s.

Tablica 6.1: Czasy przejazdu po łamanej w zależności od liczby punktów przełamania

ilosc punktow posrednich	czas przejazdu [s]	roznica 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 w stosunku do czasu najszybszego. Wydaje się, że wystarczającym powinno być, aby na odcinku takiej długości określić trzy punkty pośrednie, gdyż różnica dla tego przypadku wynosi ok. 0.023 s co stanowi ok. 0.6% czasu najszybszego, a więc nie przekracza 1%. 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.

6.1.2. Eksperymenty w terenie

Zainteresowanie tematyką narciarską skłoniło autorki pracy do przetestowania w terenie pewnych zależności i zjawisk opisywanych w tym rozdziale. Dzięki życzliwości Studenckiego Klubu Narciarskiego FIRN działającego na Akademii Górnictwo-Hutniczej w Krakowie, była możliwość dokonania pomiarów przy użyciu specjalistycznego sprzętu do pomiaru czasu na stoku narciarskim. Zostały ustalone następujące cele eksperymentów:

- eksperymentalne wyznaczenie współczynnika tarcia statycznego między śniegiem a ślizgiem nart w zależności od sposobu nasmarowania ślizgu narty oraz jakości ślizgu,

- eksperymentalne porównanie wpływu powierzchni narciarza na uzyskiwane czasy przejazdu, poprzez testowanie różnych pozycji przejazdu oraz różnych rodzajów odzieży wierzchniej,
- próba porównania szybkości przejazdu z punktu A do B w linii prostej do jazdy po łuku.

Współczynnik tarcia

Założenia i warunki eksperimentu W celu zmierzenia współczynnika tarcia, została wykorzystana drewniana deska, która została przykryta warstwą ubitego, mokrego, marcowego śniegu. Przygotowane zostały cztery pary nart:

- **A** - Narta *Atomic SL 11*, 150 cm długości, stary, nieprzygotowany i bardzo przesuszony ślizg.
- **B** - Narta *Head Worldcup Rebel S*, 155 cm długości, bardzo dobrze przygotowany ślizg, nasmarowany na gorąco niefluorowym smarem uniwersalnym (Toko System 3)
- **C** - Narta *Völkl Racetiger GS*, 175 cm długości, nowy, bardzo dobrze przygotowany pod zawody ślizg. Ślizg był posmarowany między innymi smarami fluorowymi.
- **D** - Narta *Atomic Race GS 12*, 175 cm długości, stary, nieprzygotowany, jednak nie do końca przesuszony ślizg

Ślizg każdej narty przed eksperimentem był wycierany, ale i tak było na nim osadzonych kilka kropel stopyionego śniegu. Wyłożona ubitym śniegiem deska o długości około 2.5 metra została z jednej strony podparta o podłożę, pozostającą pod niewielkim kątem w stosunku do niego. Na desce odmierzono 2 metry od punktu podparcia i oznaczono ten punkt.

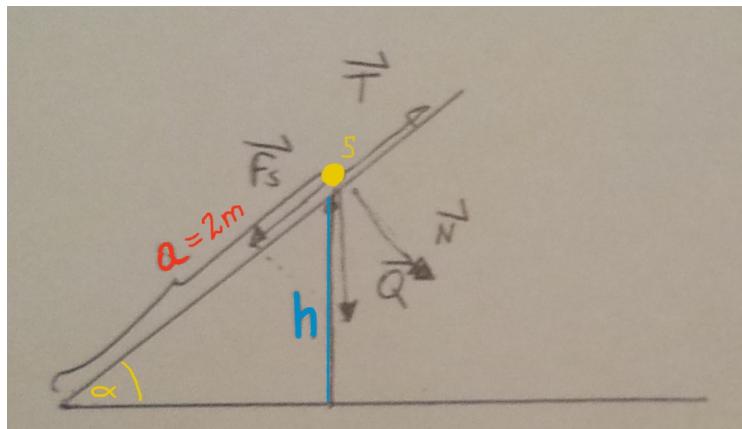


Rysunek 6.7: Narty biorące udział w eksperymencie. Oznaczenia są używane w dalszej części opisu

Przebieg eksperymentu dla każdej z nart

- wyznaczany był środek ciężkości narty i oznaczany markerem,
- na desce kładziona była narta, tak, by środek ciężkości znajdował się w oznaczonym wcześniej punkcie odległym o 2 metry od punktu podparcia,
- stopniowo zwiększany był kąt nachylenia deski do podłożu, aż do momentu w którym narta straciła przyczepność i zaczynała zsuwać się po śniegu,
- mierzona była wysokość w linii prostej między podłożem a punktem gdzie znajdował się środek ciężkości narty.

Dla każdej z nart przeprowadzone zostały dwie próby i jako wynik traktowana była średnia z tych prób. Eksperiment miał na celu pozyskanie wyniku podglądowego, dlatego też liczba dwóch prób została uznana za wystarczającą. Jako, że zauważony został podczas eksperymentu ogromny wpływ wilgotności ślizgu na wynik, zostały również przeprowadzone próby dla dwóch rodzajów nart z niewytartym ślizgiem, na którym zebrana była znaczna ilość wody.



Rysunek 6.8: S - środek ciężkości narty; a - odległość między środkiem ciężkości narty a punktem podparcia deski; h - zmierzona eksperymentalnie wysokość graniczna, po przekroczeniu której narta zaczęła się zsuwać

Wyniki i Wnioski Aby obliczyć współczynnik tarcia wykorzystana została następująca zależność: dopóki narta pozostaje w spoczynku oznacza to, że siła tarcia równoważy siłę ściągającą działającą na ciało. Zatem będąc na granicy zsuwania się, tarcie statyczne osiąga swoją maksymalną wartość. Zatem:

$$T = F_s \quad (6.5)$$

$$\mu Q \cos \alpha = Q \sin \alpha \quad (6.6)$$

$$\mu \cos \alpha = \sin \alpha \quad (6.7)$$

$$\mu = \tan \alpha \quad (6.8)$$

$$\tan \alpha = \frac{\sin \alpha}{\sqrt{1 - \sin^2 \alpha}} \quad (6.9)$$

$$\sin \alpha = \frac{h}{a} \quad (6.10)$$

$$\mu = \frac{\frac{h}{a}}{\sqrt{1 - \frac{h}{a}}} \quad (6.11)$$

W eksperymencie długość a była stała i zawsze wynosiła 2 metry. Wyniki pomiarów oraz obliczone wartości znajdują się w tabeli 6.2.

Wyniki okazały się być zgodne z przewidywaniami. Jakość ślizgu, stan jego przygotowania oraz rodzaj smaru mają znaczący wpływ na wartość współczynnika tarcia, a w konsekwencji na szybkości poruszania się na nartach. Ciekawym wnioskiem było to, że mokry ślizg ma znacznie większy współczynnik tarcia niż suchy. Wychodzi zatem jak ważna jest odpowiednio nałożona na ślizg struktura umożliwiająca odpływ wody. Narta B miała świeżą strukturę, podczas gdy narta D prawie jej nie miała.

Jak widać, wyliczone eksperymentalne współczynniki tarcia dla niezawilgoconych ślizgów mieszczą się w przedziale 0.125 - 0.193, przedział ten zawiera się w przedziale 0.001 - 0.3 podanym we wstępnie teoretycznym w sekcji 2.2 na stronie 12.

Tablica 6.2: Wyniki eksperimentu na współczynnik tarcia w zależności od rodzaju ślizgu

		Narta A	Narta B	Narta C	Narta D
wysokość na której zaczęło się osuwanie w metrach	osuszony ślizg	0.35	0.26	0.235	0.28
	wilgotny ślizg	—	0.54	—	0.66
współczynnik tarcia	osuszony ślizg	0.193	0.139	0.125	0.151
	wilgotny ślizg	—	0.316	—	0.403



Rysunek 6.9: Warunki przeprowadzania eksperimentu. Widać deskę z ubitą warstwą śniegu oraz ułożoną na niej nartę

Eksperymenty w ruchu Przeprowadzone zostały dodatkowo eksperymenty na stoku narciarskim. Pierwszy z nich polegał na porównaniu czasów przejazdu w linii spadku stoku tego samego narciarza w następujących niezmiennych warunkach:

- na odcinku o długości 28 metrów, w linii spadku stoku,
- stoku o stałym nachyleniu około 8° ,
- narciarz rozpoczyna przejazd bez odpychania się kijami,
- narciarz porusza się po wyjeżdżonym torze (brak świeżego śniegu),
- narciarz ma ten sam strój (kurtka narciarska i spodnie narciarskie) i pokonuje trasę w takiej samej pozycji - pozycji zrównoważonej).

Warunkami zmiennymi eksperymentu były użyte narty

- nieprzygotowane narty A,
- bardzo dobrze przygotowane narty B,

Pomiar prędkości był dokonywany przy użyciu sprzętu o dokładności co do setnej części sekundy. Sprzęt składa się z bramki startowej, którą narciarz przekracza, uruchamiając zegar. Zegar jest wyłączany automatycznie, gdy narciarz przetnie laserowy promień przeprowadzony wzdłuż mety. Zegar używany w eksperymencie można zobaczyć na rysunku 6.10 na stronie 59.



Rysunek 6.10: Zegar marki *Brower*, użyty do przeprowadzenia eksperymentu. Sprzęt użyczony dzięki życzliwości Studenckiego Klubu Narciarskiego FIRN AGH

Zmierzone czasu były istotnie różne: 6.48 s dla narciarza na bardzo dobrze przygotowanych nartach i 7.19 s dla bardzo słabo przygotowanych nart. Należy zwrócić uwagę, że jest to różnica o ponad 0.7 s i to wypracowana na bardzo krótkim odcinku. Czas przejazdu na złe przygotowanych nartach jest o ponad 10% dłuższy. Zazwyczaj przejazd zawodnika na trasie zawodów narciarskich trwa między 30 a 120 s, więc różnica będzie się znaczco pogłębiać. Zostało zatem potwierdzone eksperymentalnie, że warto dbać o ślizgi narty, nie tylko dla zwiększenia komfortu jazdy i lepszej konserwacji sprzętu, ale również dla osiągania lepszych wyników w zawodach.

Kolejny eksperyment miał na celu zbadanie jak wpłynie na czas przejazdu poruszanie się po świeżym śniegu, zamiast po wyślizganym już torze. Jako stały warunek eksperymentu dodajemy zatem użyte narty - w tym przypadku dobrze przygotowane. W tym eksperymencie narciarz ubrany był w gumę narciarską i poruszał się za każdym razem w pozycji obniżonej tzn. „na jajo”. Zmieniany był natomiast tor przejazdu:

- wyślizgany tor,
- cienka warstwa świeżego śniegu.

Zmierzone czasy były ponownie istotnie różne. Przejazd po wyślizganym torze zajął 5.98 s, natomiast po dziewczyczym torze aż 7.19 s. Jest to ponownie, gigantyczna różnica w ujęciu tak krótkiego czasu przejazdu - czas po dziewczyczym śniegu jest o ponad 22% gorszy.

Podsumowując eksperymenty dynamiczne: zarówno rodzaj podłoża, jak i stan ślizgu narty wpływają w bardzo znaczący sposób na wartość współczynnika tarcia, a w konsekwencji na czas pokonywania trasy.

Siła oporu powietrza

Założenia i warunki eksperymentu Kolejny eksperyment przeprowadzany był już tylko na stoku narciarskim - Siepraw Ski w pobliżu Krakowa. Eksperyment miał na celu sprawdzenie zależności pomiędzy powierzchnią narciarza a czasem jego przejazdu. Ponieważ za zmienną eksperymentu przyjmujemy tylko powierzchnię narciarza, sprawdzamy tak naprawdę zależność pomiędzy tą powierzchnią a wartością sił oporu powietrza. Przygotowane zostały następujące scenariusze:

- narciarz ubrany jest w tzw. *gumę narciarską* przylegającą do ciała. Strój ten opisany jest szerzej w sekcji 2.2 na stronie 10. Narciarza w tym stroju można zobaczyć na rysunku 6.12 na stronie 61. Narciarz przyjmuje dwa warianty pozycji:
 - pozycję podwyższoną,
 - pozycję na tzw. jajo (widoczną na rysunku 6.12).
- narciarz ubrany jest w klasyczny strój narciarski, czyli spodnie narciarskie i kurtkę. Narciarz przyjmuje pozycję podwyższoną - zobrazowane to jest na rysunku 6.11 na stronie 61

Dodatkowo podczas eksperymentu panują następujące stałe warunki:

- kolorowym sprayem wyznaczono linię wzdłuż linii spadku stoku o długości 28 metrów,
- stok jest o stałym nachyleniu około 8 stopni,
- narciarz rozpoczyna przejazd bez odpychania się kijami,
- narciarz porusza się po wyjeżdżonym torze (brak świeżego śniegu),
- bezwietrzna pogoda, warunki atmosferyczne nie zmieniają się podczas trwania eksperymentu,
- narciarz korzysta za każdym razem z tego samego sprzętu narciarskiego.

Przebieg eksperymentu dla każdego wariantu pozycji

- narciarz przyjmuje odpowiednią pozycję i rozpoczyna jazdę, co powoduje samoczynne uruchomienie zegara,
- narciarz nie wykonuje żadnych ruchów, stara się tylko prowadzić narty wzdłuż wyznaczonej linii,
- narciarz przekracza linię mety, co powoduje automatyczne zatrzymanie zegara,
- czas przejazdu jest notowany.

Wyniki i Wnioski W pozycji podwyższonej, ubrany w klasyczny strój narciarski narciarz potrzebował 6.48 s na przebycie wyznaczonego odcinka (rysunek 6.11). Zmiana stroju na bardziej opływowy - tj. ubranie tylko gumy narciarskiej, pozwoliła zredukować czas przejazdu o 6.7% - do 6.04 s. Jest to bardzo duża różnica na tak niewielkim odcinku. Kolejna optymalizacja, czyli przyjęcie pozycji zjazdowej tzw. na jajo (widocznej na rysunku 6.12), pozwoliła zredukować czas przejazdu o kolejny 1% uzyskując czas 5.98 s.

Udało się doświadczalnie pokazać, że zmiana powierzchni narciarza ma istotny wpływ na czas przejazdu. Warunki prowadzenia eksperymentu i posiadanego sprzęt nie pozwoliły wyizolować zmian wielkości powierzchni narciarza, od zmian współczynnika C , z równania na wartość siły oporu powietrza ze wstępem teoretycznego ze strony 10. Należy bowiem zwrócić uwagę, że guma narciarska nie tylko zmniejsza powierzchnię narciarza, ale znaczco zwiększa jego „opływowość”, a więc zmniejsza współczynnik C .



Rysunek 6.11: Narciarz w pozycji podwyższonej, ubrany w klasyczny strój narciarski składający się z kurtki narciarskiej i spodni narciarskich



Rysunek 6.12: Narciarz w pozycji zjazdowej, ubrany w tzw. gumę narciarską

6.1.3. Jazda po łuku, a jazda w skos stoku

Nawiązując do eksperymentu przeprowadzonego z użyciem stworzonego przez nas modelu komputerowego, z sekcji 6.1.1 ze strony 52, który wykazał, że najbardziej optymalny pod względem czasowym tor przejazdu z punktu A do B, wcale nie jest linią prostą między tymi punktami, a raczej fragmentem cykloidy, zaprojektowany został analogiczny eksperiment na stoku narciarskim. Warunki wykonania eksperymentu nie są idealne, aczkolwiek wystarczające, by zauważać zakładaną zależność.

Opis eksperymentu Na stoku oznaczony został punkt startowy (A) oraz linia mety z oznaczonym środkiem mety (B). Następnie na śniegu została oznaczona sprayem linia prosta między punktami startu a środkiem mety. Stok miał stałe, niewielkie nachylenie. Narciarz miał za zadanie na jak najbardziej płasko prowadzonych nartach pokonać zaznaczony odcinek z A do B. Został zanotowany czas z kilku prób, a następnie obliczona została średnia: 6.12 s. Kolejnym zadaniem narciarza było przejechanie po płynnym łuku - z punktu A do B. Po kilku próbach doboru odpowiedniego promienia skrętu udało się przeprowadzić kilka prób zakończonych znalezieniem się w punkcie B. Tor przejazdu po łuku został oznaczony sprayem dla lepszego zobrazowania eksperymentu. Czasy przejazdu z tych prób zostały zanotowane i również obliczona została średnia - wynosząca 5.49 s. Czas przejazdu istotnie różni się zatem w zależności od toru. Oczywiście w grę wchodzi tu ogromna liczba czynników między innymi inne tarcie pomiędzy krawędziami narty a śniegiem, niż w przypadku tarcia ślizgu narty o śnieg. Niezaprzeczalny jest jednak fakt, że czas przejazdu był o ponad 11% dłuższy dla jazdy w linii prostej w stosunku do jazdy po łuku. Warunki przeprowadzania eksperymentu są dobrze widoczne na rysunku 6.13 na stronie 62.



Rysunek 6.13: Eksperiment porównujący przejazd pomiędzy punktami, wybierając jazdę po wycinku łuku, a jazdę w skos stoku na płaskich nartach

6.2. Optymalizacja toru przejazdu

W tej części zostaną pokazane wyniki jakie zostały otrzymane jako efekt działania algorytmu ewolucyjnego w celu znalezienia optymalnej trasy przejazdu. Zostanie sprawdzone jak poszczególne parametry algorytmu wpływają na czas jego działania oraz uzyskiwane wyniki. Spróbujemy także rozwiązać problem dla trudniejszych przy-

padków: zwiększając liczbę punktów, które stanowią rozwiążanie oraz optymalizując trasę dłuższego slalomu. Następnie zostanie przedstawione jak optymalizacja lokalna wpływa na czas wykonania programu.

6.2.1. 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 teraz przetestowane, jak sprawdza się algorytm ewolucyjny 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 wertykal - figurę slalomową - szerzej opisaną w sekcji 2.1, często stosowaną w slalomie, która ma wymuszać zmianę rytmu, a więc będzie dobrym sprawdzianem dla zastosowanego algorytmu.

Zakładamy, że narciarz startując ma prędkość początkową $1\frac{m}{s}$. Wynika to z tego, 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 6.1.1 zostało stwierdzone, że odległość 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 ewolucyjnego są takie, jak wymienione wcześniej w rozdziale 5.4. Wykorzystany został algorytmu ($\mu + \lambda$), gdzie $\mu = 30$, a $\lambda = 100$. W każdej iteracji najpierw krzyżowane są osobniki z populacji tymczasowej, losując pary rodzicielskie, a w wyniku krzyżowania powstaje znów 100 osobników. Następnie mutacji podlegają wszystkie z tych osobników. Jedynym dodatkowym założeniem jest nie zmienianie pozycji punktów przejazdu przez bramki - narciarz ma przejeżdżać 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 ewolucyjnym jest sprawdzenie, czy populacja, na której działały 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 rozwiążaniu tym istnieją jednak trzy parametry, których wpływ na ostateczny wynik został zbadany w tym eksperymencie.

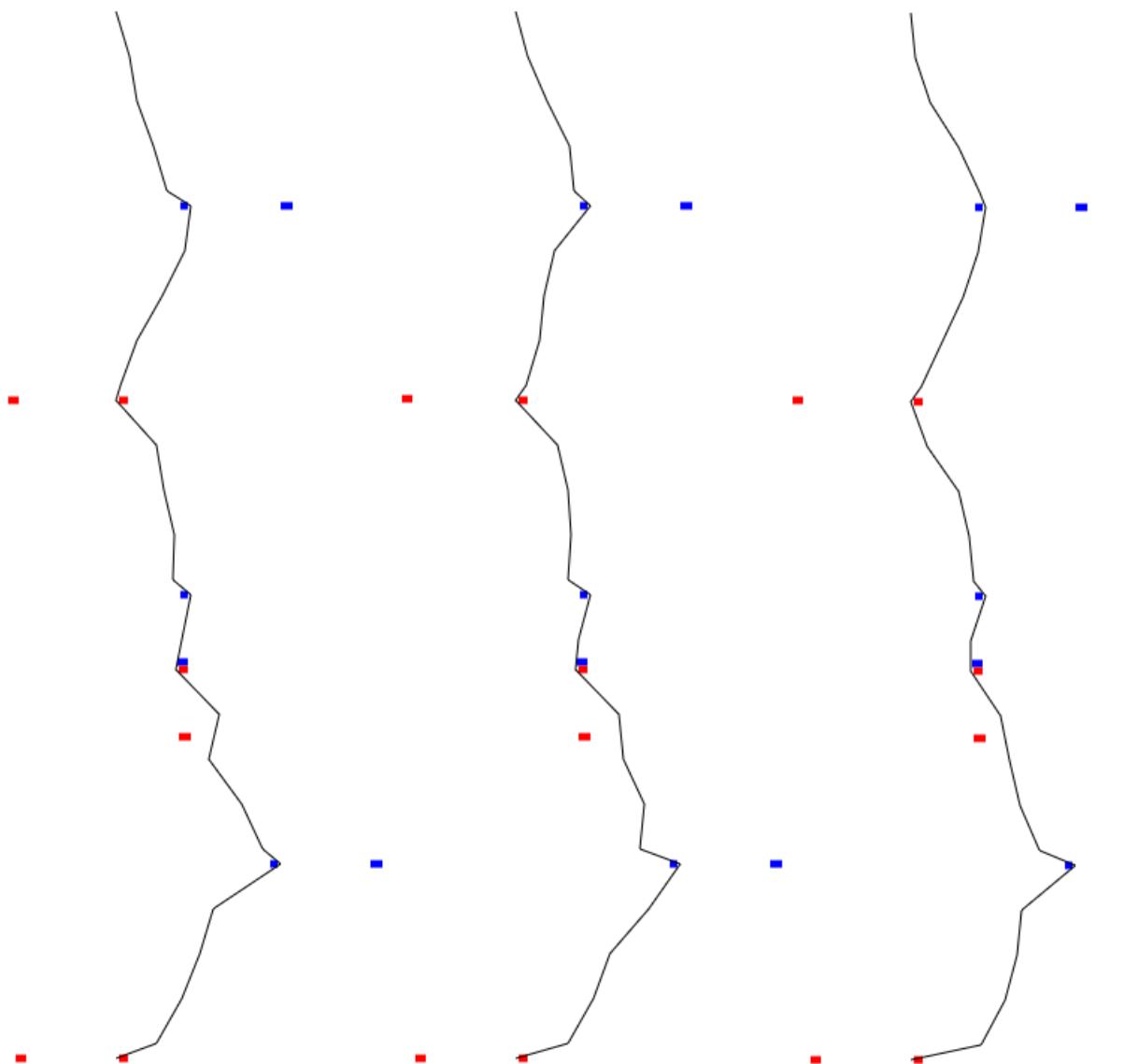
Eksperyment 1. W pierwszym eksperymencie parametry zakończenia algorytmu wynoszą:

- 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, poprzez znalezienie różnicy tych wartości i podzielenie otrzymanej wartości przez wartość najlepszą (aby uzyskać wartość w % mnożymy przez 100%),
- zmiana najlepszego osobnika jest uznawana za wystarczająco dużą, jeśli wyniesie przynajmniej 0.3 s,
- liczba iteracji, przez które czekamy na zmianę najlepszego osobnika, to na początek 7.

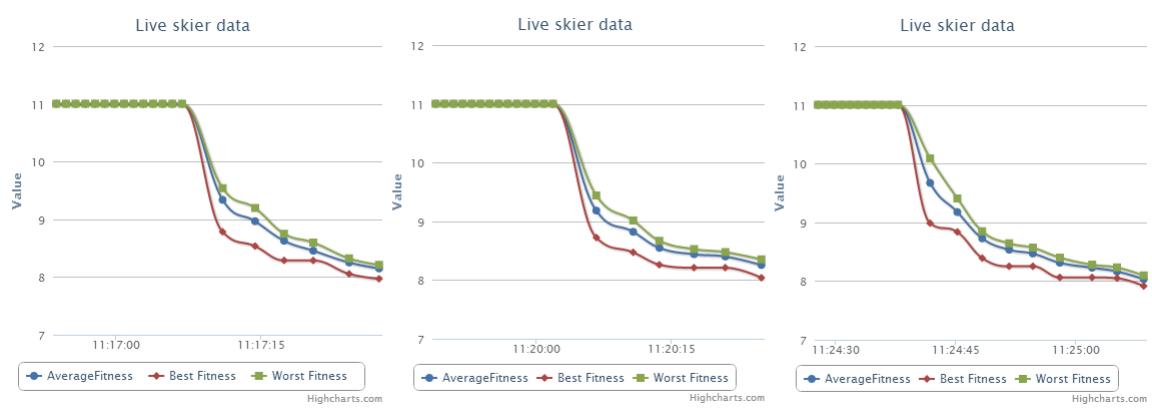
W związku z tym, że zachowanie algorytmu jest niedeterministyczne, dla każdego zestawu parametrów zostały przeprowadzone po trzy eksperymenty, aby można było uśrednić wyniki i porównać eksperymenty między sobą.

Poniżej (rysunki 6.14 oraz 6.15) 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 (kolor niebieski). Kolejność slalomów odpowiada kolejności wykresów.



Rysunek 6.14: Wyniki z kolejnych prób dla parametrów zakończenia: 10%, 0.3 s i 7



Rysunek 6.15: Wykresy z kolejnych prób dla parametrów zakończenia: 10%, 0.3 s i 7

Oto otrzymane czasy przejazdu dla kolejnych prób:

Tablica 6.3: Czasy przejazdu dla kolejnych prób wraz z liczbą iteracji oraz czasem działania algorytmu

numer próby	czas przejazdu [s]	liczba iteracji	czas działania [s]
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 s.

Liczba iteracji w przypadku tych wyników wyniosła 7-9. Algorytm bardzo szybko dochodził do rozwiązania, a czas działania to tylko ok. 20 s. Należy zwrócić uwagę, że biorąc pod uwagę rozproszony model prowadzenia obliczeń, czas działania algorytmu może być znacznie wydłużany - tak by narzuty na komunikację i zarządzanie wysyłanymi zadaniami nie były znaczącą częścią czasu wykonania.

Jednocześnie, co obserwujemy na rysunkach, wyniki nie są satysfakcyjne - 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.

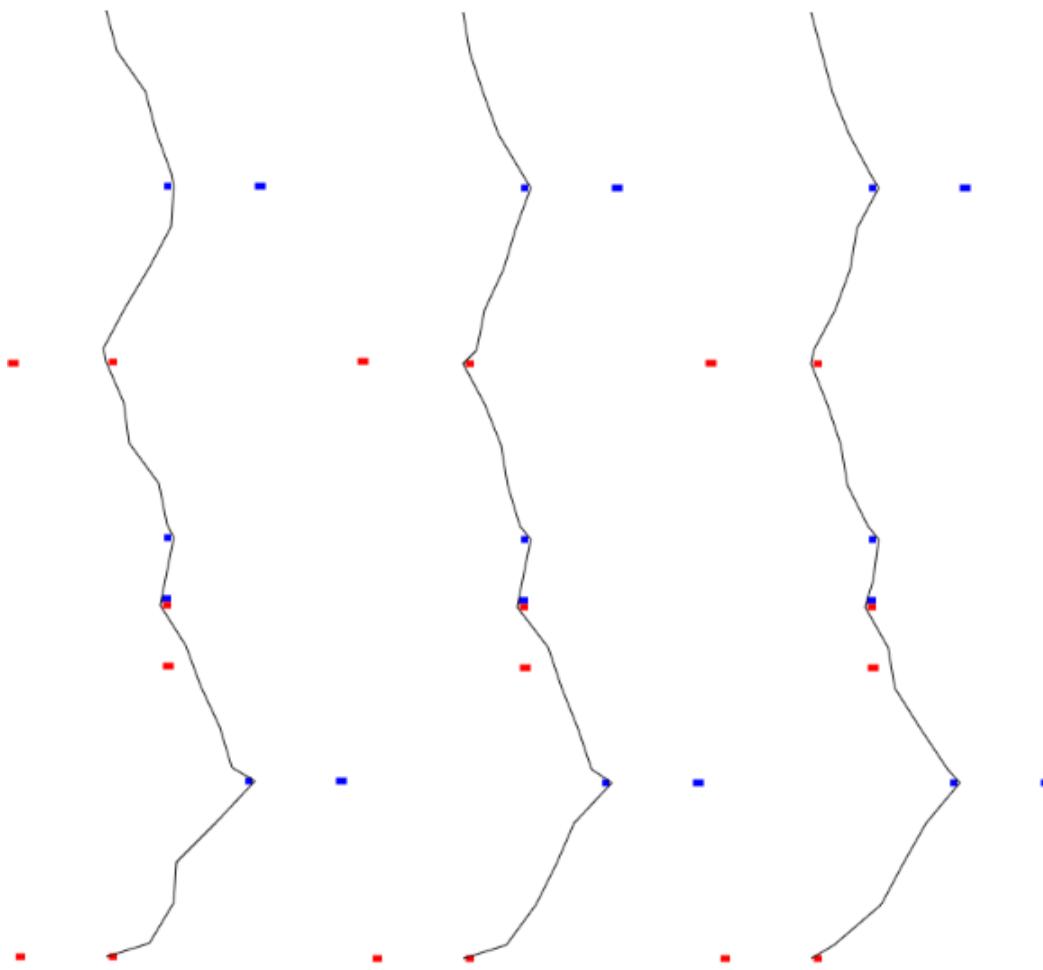
Eksperyment 2. W tym eksperymencie została wprowadzona zmiana w wartości parametru oznaczającego różnicę w fitness najlepszego osobnika. Zamiast wartości 0.3 s użyto 0.1 s. Sprawdzone zostało zarówno, czy czas działania algorytmu się wydłuża jak i czy uda się przez to otrzymać lepszy wynik.

Na rysunkach 6.16 oraz 6.17 a także w tabeli 6.4 zebrane są wyniki eksperymentu.

Tablica 6.4: Czasy przejazdu dla kolejnych prób wraz z liczbą iteracji oraz czasem działania algorytmu

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

Średnia uzyskanych czasów przedstawionych w tabeli 6.4 wynosi: 7.834 s.



Rysunek 6.16: Wyniki z kolejnych prób dla parametrów zakończenia: 10%, 0.1 s i 7



Rysunek 6.17: Wykresy z kolejnych prób dla parametrów zakończenia: 10%, 0.1 s i 7

Wyniki poprawiły się, linia przejazdu jest dużo gładzsza, a czasy poprawiły się o ok. 0.15 s, czyli o ok. 2%. Jednak czas działania algorytmu wydłużył się do 45-50 s, a liczba iteracji wyniosła 14-16. Oczywiście czas ten jest wciąż na tyle mały, że możemy wprowadzać dalsze zmiany, które wydłużą czas działania i jednocześnie umożliwiają znalezienie lepszego rozwiązania.

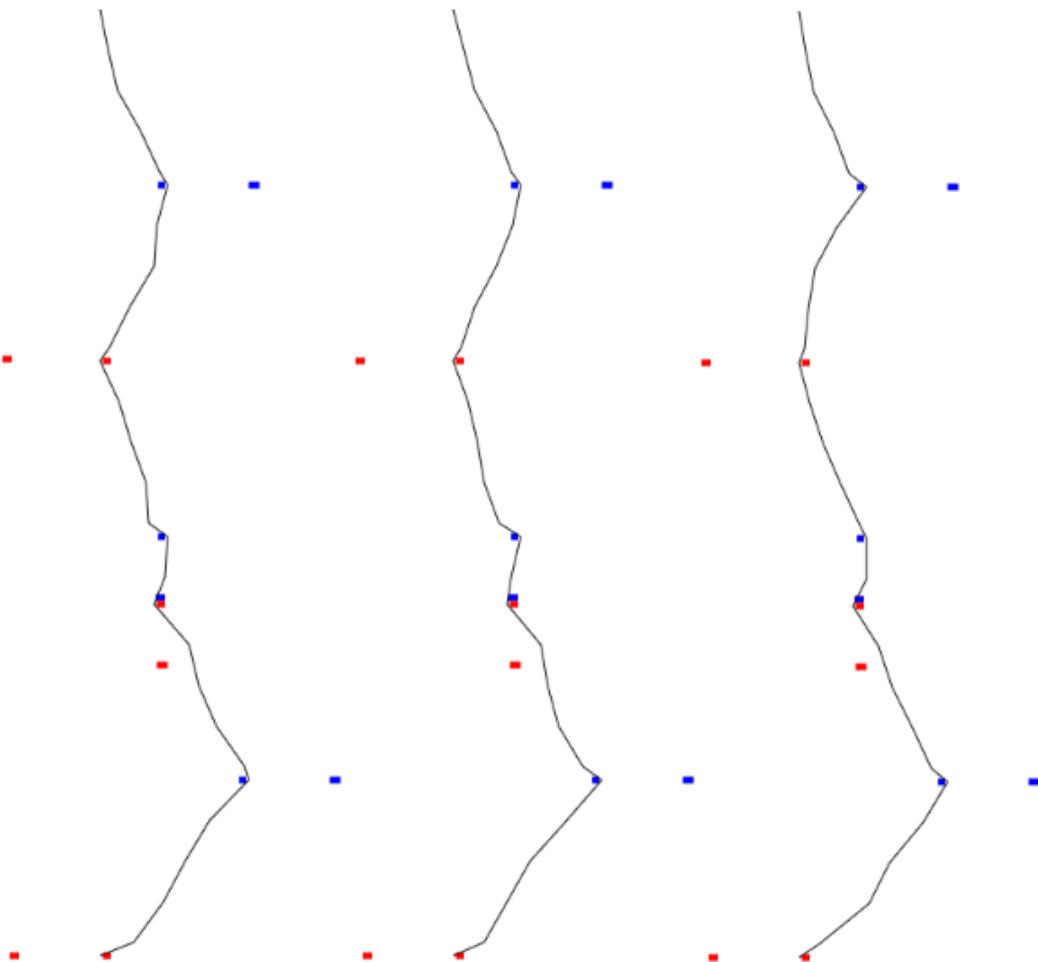
Eksperyment 3. W takim razie zdecydowano zmienić również parametr dotyczący zróżnicowania populacji - zamiast 10%, w tym eksperymencie wynosi on 1%.

Poniżej znajdują się zebrane wyniki w tabeli 6.5 oraz na rysunkach 6.18 oraz 6.19.

Tablica 6.5: Czasy przejazdu dla kolejnych prób wraz z liczbą iteracji oraz czasem działania algorytmu

numer próby	czas przejazdu [s]	liczba 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 s.



Rysunek 6.18: Wyniki dla kolejnych prób eksperymentu dla parametrów zakończenia: 1%, 0.1 s i 7



Rysunek 6.19: Wykresy dla kolejnych prób eksperymentu dla parametrów zakończenia: 1%, 0.1 s i 7

Jak widać wyniki są niewiele lepsze niż te poprzednie. Zatem ten warunek praktycznie nie zmienił zachowania. Pozostał natomiast jeszcze warunek liczby iteracji, przez które sprawdzane jest, czy poprawia się fitness najlepszego osobnika.

Eksperyment 4. Zmieniamy teraz wartość liczby iteracji, przez które czekamy na zmianę najlepszego osobnika z 7 na 10.

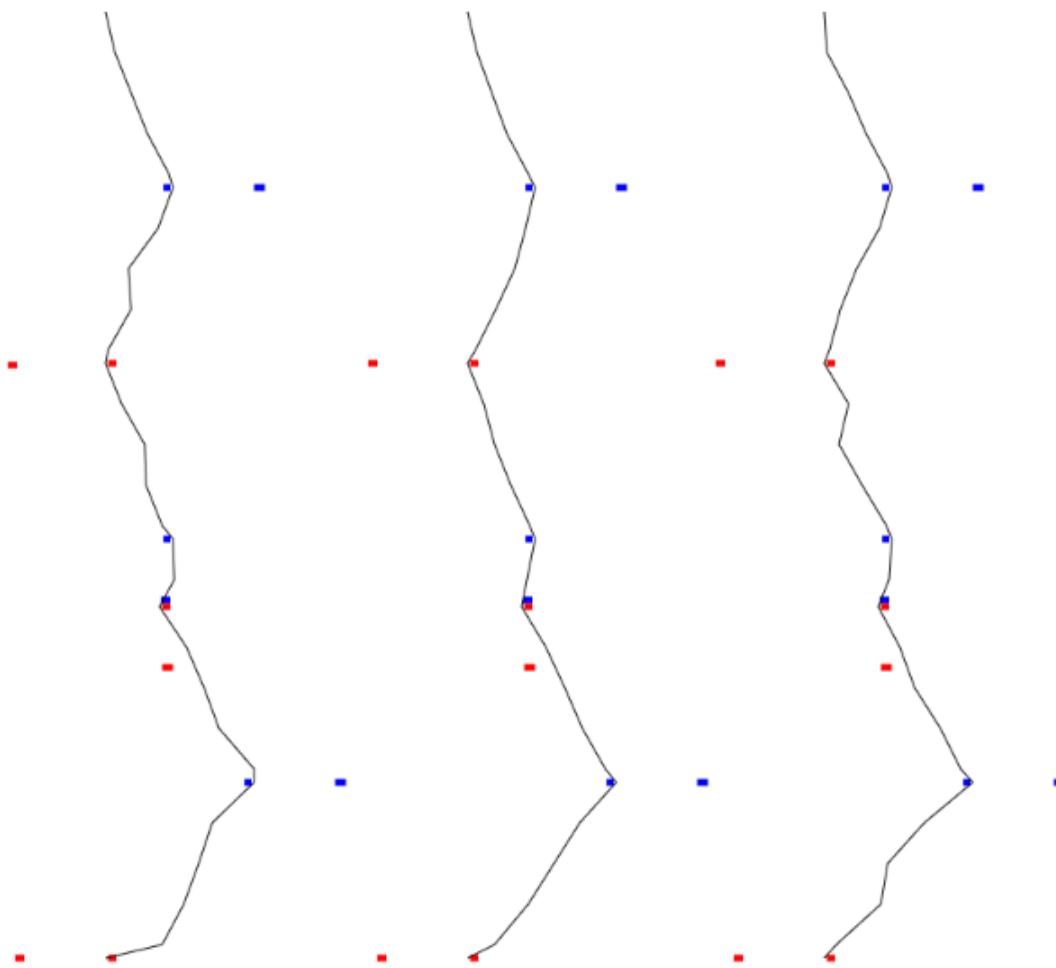
Tablica 6.6: Czasy przejazdu dla kolejnych prób wraz z liczbą iteracji oraz czasem działania algorytmu

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

Średnia wyników wynosi: 7.824 s. Warto zwrócić uwagę, że średnio w każdym eksperymencie wykonywanych jest 1950 symulacji. Zależy to od liczby osobników w populacji oraz populacji tymczasowej, a więc parametrów μ (30) i λ (100), a także od liczby iteracji wykonywanych w każdym eksperymencie (średnio: 15 iteracji, a zatem $(30 + 100) * 15 = 1950$).



Rysunek 6.20: Wykresy dla kolejnych prób eksperymentu dla parametrów zakończenia: 1%, 0.1 s i 10



Rysunek 6.21: Wyniki dla kolejnych prób eksperymentu dla parametrów zakończenia: 1%, 0.1 s i 10

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

Warto jednak zauważyć, jaka jest charakterystyka trasy otrzymanej tu jako najlepsze rozwiązanie (o numerze próby 2). Tor jazdy jest po prostu łamaną, której wierzchołki znajdują się w miejscach bramek - pomiędzy kolejnymi bramkami linie są właściwie odcinkami co powoduje konieczność wykonywania bardzo ostrych skrętów. Ten problem zostanie rozwiązany w dalszej części, w rozdziale 6.2.3 opisującym wyniki zastosowania funkcji kary.

W trakcie szczegółowej analizy wyników z kolejnych iteracji (widocznych na wykresach), łatwo zauważać, że na początku poprawa wyników jest znacząca z każdą iteracją, a pod koniec zmiany są już niewielkie. Można zatem pomyśleć o wprowadzeniu innego algorytmu, kiedy zmiany są już słabo widoczne, który skuteczniej znajdzie lepsze rozwiązanie. Takim rozwiązaniem jest na przykład zastosowanie lokalnej optymalizacji. Wyniki tego podejścia opisane są w sekcji 6.2.2.

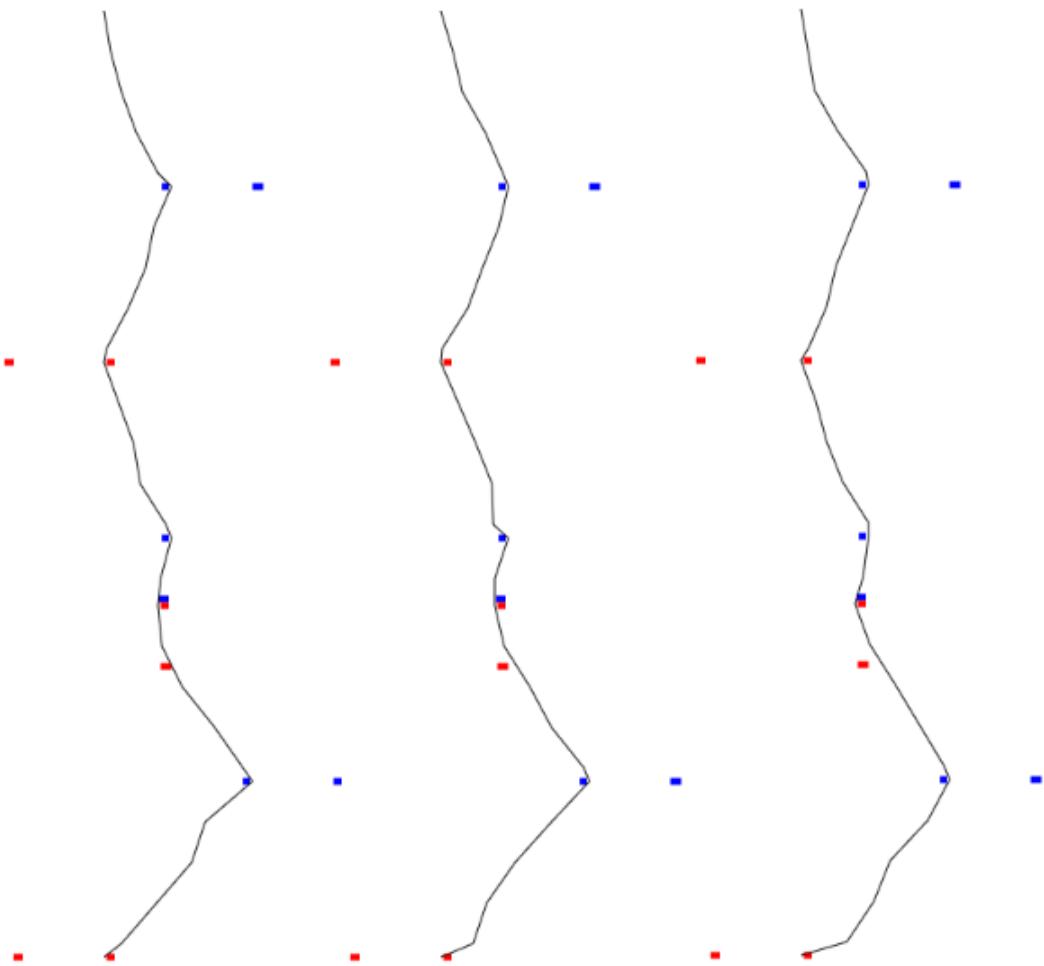
Zanim przejdziemy jednak do wyników wprowadzenia lokalnej optymalizacji omówione zostaną eksperymenty sprawdzające ostatnie parametry algorytmu ewolucyjnego: μ i λ , a następnie pokazane zostanie rozwiązanie trudniejszego zadania, uzyskanego poprzez zwiększenie liczby punktów stanowiących rozwiązanie.

Parametry μ i λ

Do tej pory początkowa populacja liczyła 30 osobników, a populacja tymczasowa - 100. W literaturze najczęściej wymienia się wartości około 50 i 200 [2], warto zatem spróbować zwiększyć te parametry, by sprawdzić

więcej przypadków rozwiązań. Pozostałe parametry pozostają jak w poprzednim eksperymencie.

Eksperyment 1. W tym eksperymencie zostało sprawdzone jak wpływa na wyniki zmiana powyższych parametrów, na początek dla wartości odpowiednio 50 i 150.



Rysunek 6.22: Wyniki dla kolejnych prób eksperymentu dla parametrów $\mu = 50$ i $\lambda = 150$



Rysunek 6.23: Wykresy dla kolejnych prób eksperymentu dla parametrów $\mu = 50$ i $\lambda = 150$

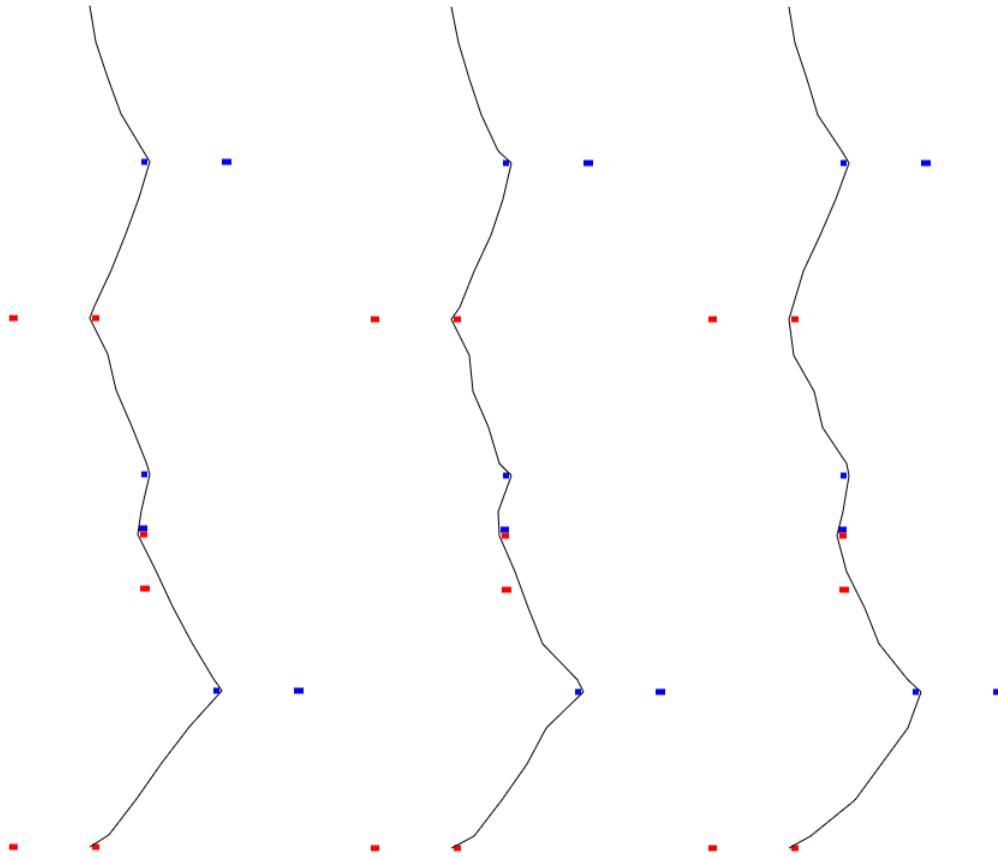
Tablica 6.7: Czasy przejazdu dla kolejnych prób wraz z liczbą iteracji oraz czasem działania algorytmu ($\mu = 50$ i $\lambda = 150$)

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

Średnia wyników podanych w tabeli 6.7 wynosi: 7.799 s. W tym przypadku średnio w każdym eksperymencie zostało wykonanych około 3670 simulacji.

Można się było spodziewać poprawy wyników, jako, że rozpatrywano naraz więcej osobników sprawdzając więcej możliwych rozwiązań, a więc zwiększaając 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. To pokazuje, 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 s.

Eksperyment 2. W kolejnym eksperymencie sprawdzono jak zmienią się wyniki jeśli wartości μ i λ wyniosą odpowiednio 60 i 200.



Rysunek 6.24: Wyniki dla kolejnych prób eksperymentu dla parametrów $\mu = 60$ i $\lambda = 200$

Rysunek 6.25: Wykresy dla kolejnych prób eksperymentu dla parametrów $\mu = 60$ i $\lambda = 200$ Tablica 6.8: Czasy przejazdu dla kolejnych prób wraz z liczbą iteracji oraz czasem działania algorytmu ($\mu = 60$ i $\lambda = 200$)

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

Średnia wyników wynosi: 7.762 s. Średnio w każdym eksperymencie zostało wykonanych 4680 symulacji.

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 otrzymano najlepszy czas - 7.729 s. Widzimy również, że zgodnie z oczekiwaniemi, wydłużył się znów czas działania algorytmu. Minimalny czas to 118 s, czyli około 2 minuty, a wykonanie trwało maksymalnie prawie 3 minuty. Takie czasy już można uznać za wystarczające, aby warto było rozsyłać obliczenia, by były wykonywane na innych węzłach obliczeniowych.

Dla porównania przeanalizowano jak wpłyneły zmiany parametrów μ i λ na uzyskiwane wyniki na podstawie danych z poprzednich eksperymentów. Uwzględniona jest także średnia liczba symulacji koniecznych do osiągnięcia wyniku dla każdej pary ($\mu + \lambda$).

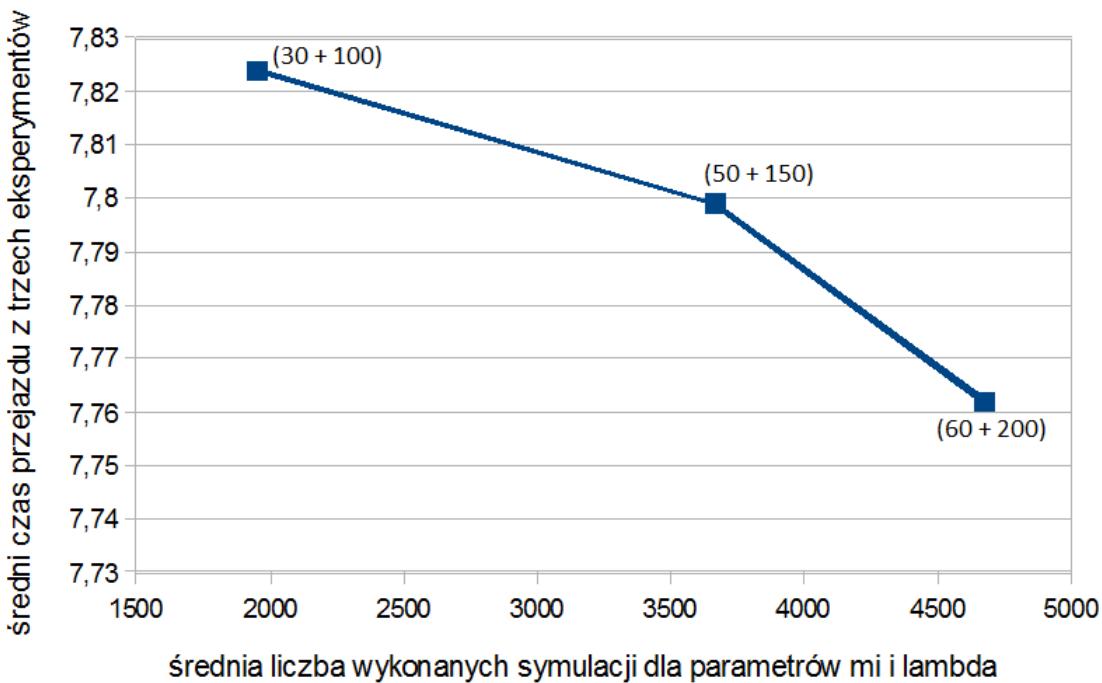
Wykres 6.26 pokazuje jak zmienił się średni czas uzyskanych wyników odpowiednio dla kolejnych wartości parametrów μ i λ . Widać, że choć poprawa jest niewielka, bo o około 0.05 s, to jednak widoczny jest trend wskazujący, że zwiększenie populacji wpływa pozytywnie na znajdowane rozwiązanie. Poprawa jest lepsza w drugiej części wykresu, biorąc pod uwagę jak zmienia się liczba symulacji koniecznych do osiągnięcia wyniku, w porównaniu z poprawą uzyskanego średniego czasu przejazdu.

Gęstość siatki

Aby uzyskać dokładniejsze rozwiązania 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 liczbę rozpatrywanych punktów rozwiązania.

W eksperymencie wartości parametrów wynoszą dla warunku zakończenia:

- dla zróżnicowania populacji przyjęto wartość 0.01, czyli 1%,
- zmiana najlepszego osobnika jest uznawana za wystarczająco dużą jeśli wyniesie przynajmniej 0.05 s,
- liczba iteracji, przez które czekamy na zmianę najlepszego osobnika - 10.



Rysunek 6.26: Średni czas przejazdu z wykonanych prób dla różnych parametrów μ i λ

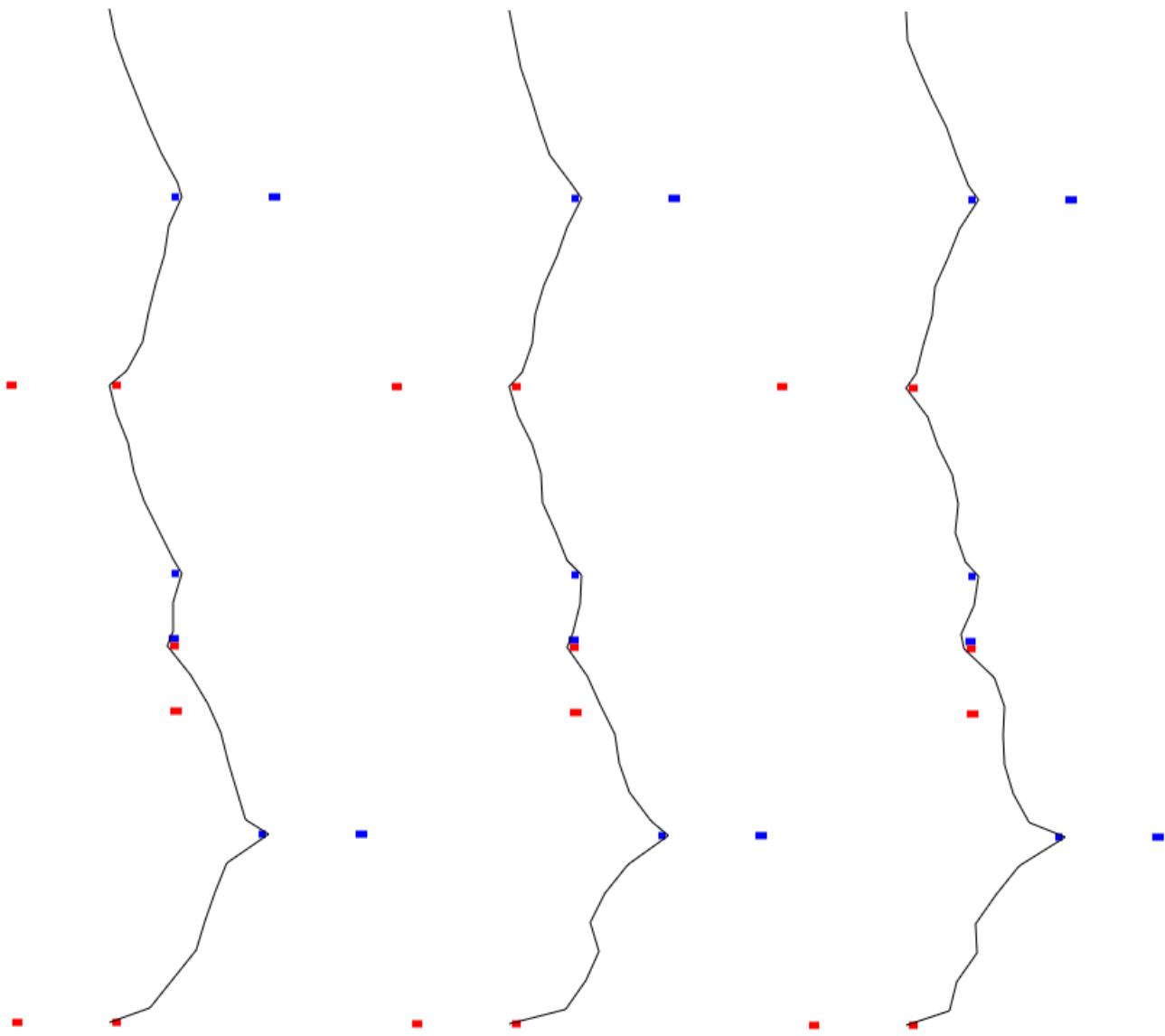
Natomiast parametry μ i λ przyjmijmy 30 i 100.

Na rysunkach 6.28 oraz 6.27 przedstawione są wyniki z trzech przeprowadzonych prób.



Rysunek 6.27: Wykresy dla kolejnych prób eksperymentu dla zwiększonej gęstości siatki punktów rozwiązania

Czas wykonania wydłużył się do 100-150 s, a liczba 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 - obliczona trasa jest zbyt "kanciasta". Wydaje się, że taki sposób optymalizacji nie umożliwia nam znalezienia rzeczywiście najszybszej trasy. Zachodzi zatem konieczność wprowadzenia mechanizmu, który wyeliminuje zbyt ostre zmiany kierunku np. poprzez uwzględnienie konieczności zmniejszenia prędkości w miejscach o zbyt dużym kącie załamania. Szczegółowo mechanizm ten opisany jest w sekcji 5.5 na stronie 40.



Rysunek 6.28: Wyniki dla kolejnych prób eksperymentu dla zwiększonej gęstości siatki punktów rozwiązania

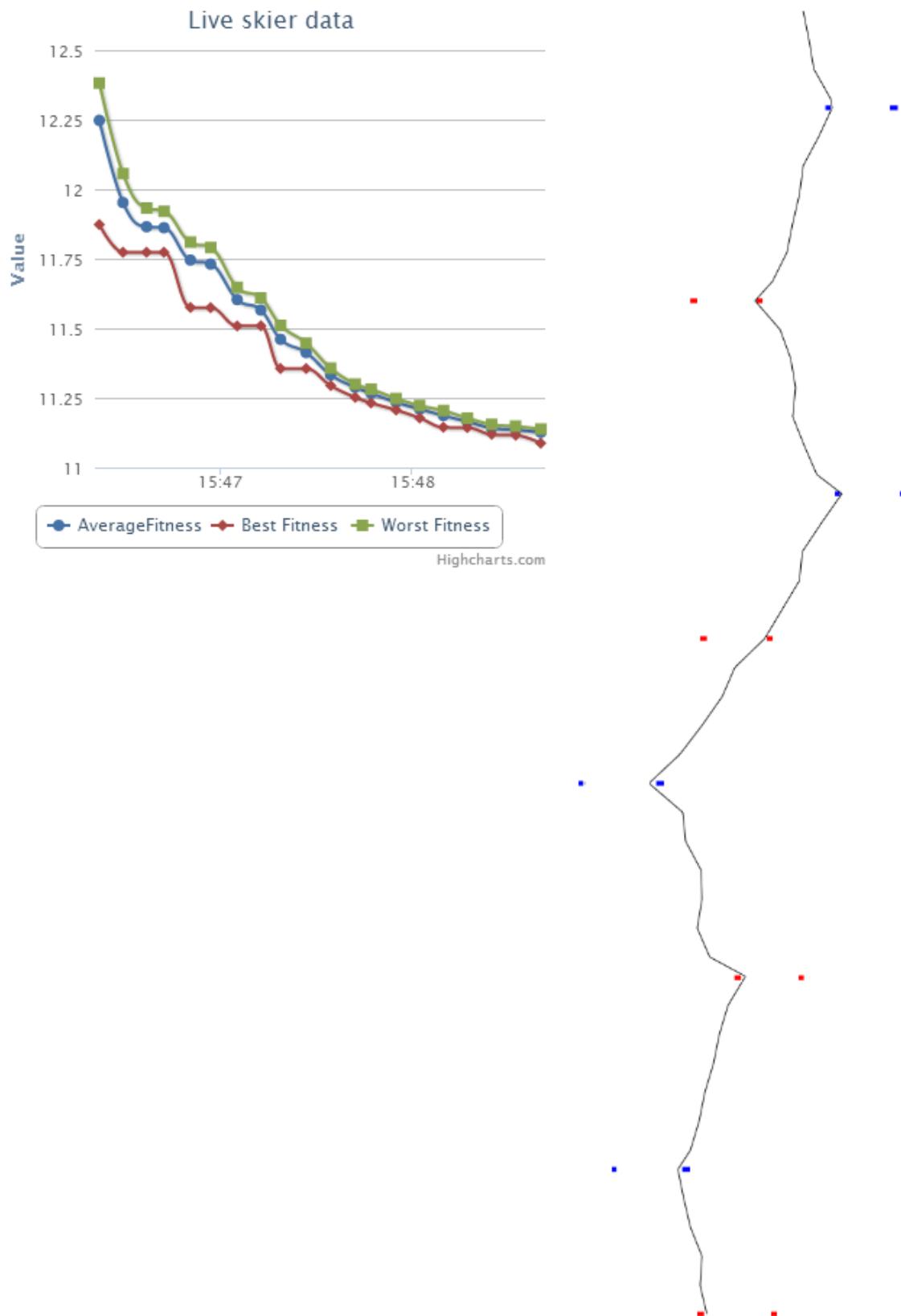
Długa trasa

Dotychczas rozpatrywany slalom był jedynie fragmentem standardowego slalomu, czas przejazdu wynosił mniej niż 8 s. Pozostaje pytanie, czy dla większej liczby 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.

W tym eksperymencie sprawdzone zostało jakie wyniki otrzymamy dla dłuższego slalomu: (19,10), (11,30), (20,50), (12,65), (0,80), (10,100), (3,120), (6,135)). Parametry μ i λ pozostały ustawione na 30 i 100. Parametry zatrzymania algorytmu zostały ustawione tak, 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,
- liczba iteracji, przez które czekamy na zmianę najlepszego osobnika - 10.

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



Rysunek 6.29: Uzyskany wynik wraz z wykresem dla eksperymentu na dłuższej trasie

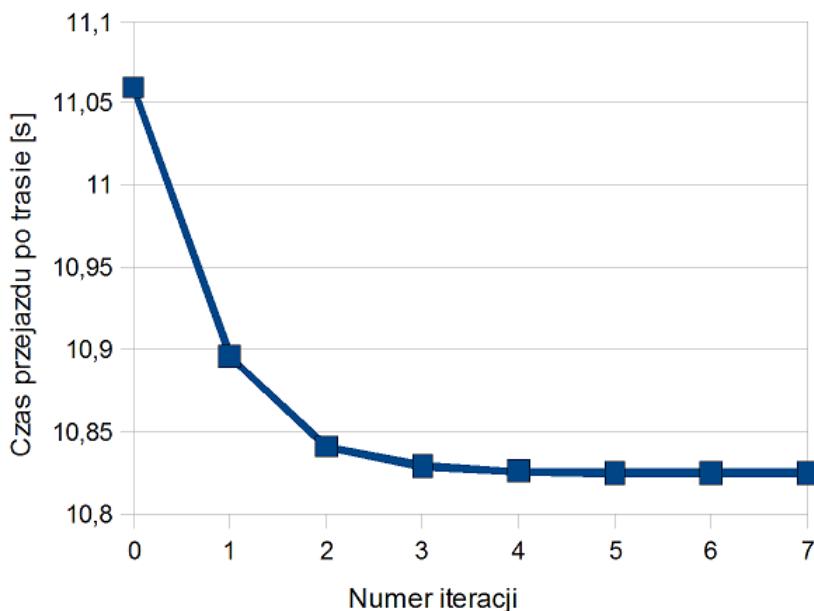
Otrzymany wynik to 11.087 s. Obliczenia trwały 170 s, a algorytm potrzebował 23 iteracje do zakończenia obliczeń. Znów można powiedzieć, że wynik jest dobry, jednak na pewno mógłby być lepszy - istnieje wiele ostrych zakrętów, zwłaszcza w okolicach bramek, rozwiązań nie jest idealne.

Warto zauważyc, że wydłużenie trasy nie spowodowało natomiast większych problemów jeśli chodzi o znalezienie trasy, mimo zastosowanych warunków zakończenia i zwiększenia liczby punktów z około 30 na prawie 50.

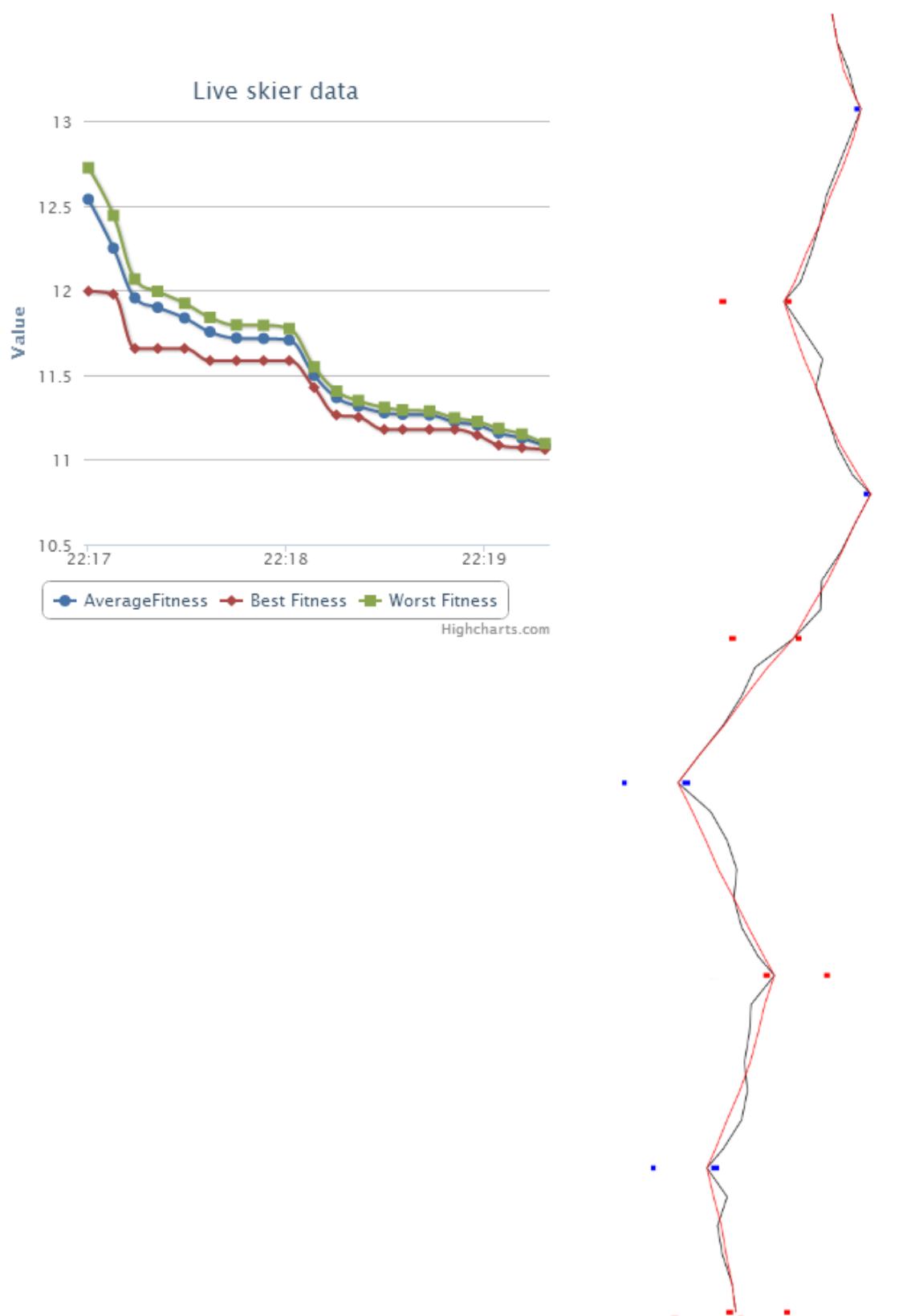
6.2.2. Lokalna optymalizacja

Algorytm ewolucyjny ma problem ze znalezieniem lepszych rozwiązań, zwłaszcza kiedy dotychczas znalezione rozwiązanie daje już dosyć dobry wynik, ale wciąż można gołym okiem zobaczyć miejsca, które negatywnie wpływają na jakość wyniku. Skoro algorytm nie radzi sobie będąc blisko rozwiązania, można spróbować zastosować algorytm memetyczny - bazując na rozwiązaniu znalezionym przez algorytm ewolucyjny, dokonujemy dalszej optymalizacji wykorzystując algorytm lokalnej optymalizacji. 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.

Eksperyment 1. Bazując na eksperymencie “Długa trasa” w sekcji 6.2.1 porównano jak poprawi się otrzymany wynik dodając na koniec obliczeń optymalizację lokalną algorymem Hill climbing. Wszystkie parametry oraz trasa przejazdu pozostały takie same jak w eksperymencie, do którego się odwołujemy.



Rysunek 6.30: Wykres zmian czasu przejazdu w kolejnych iteracjach algorytmu HC



Rysunek 6.31: Porównanie wyników ES oraz HC wraz z wykresem dla eksperymentu na dłuższej trasie

Na wykresie 6.31 znajduje się porównanie wyniku algorytmu ewolucyjnego oraz lokalnej optymalizacji. Czarna linia trasy jest rezultatem otrzymanym po działaniu samego algorytmu ewolucyjnego, czas przejazdu po

takiej trasie wynosi 11.06 s, a więc jest to czas zbliżony do otrzymanego w eksperymencie bazowym. Algorytm ewolucyjny działał przez 21 iteracji.

Czerwona linia stanowi trasę uzyskaną po zastosowaniu algorytmu optymalizacji lokalnej na trasie znalezionej przez algorytm ewolucyjny. Na wykresie 6.30 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.

Eksperiment 2. W eksperymencie pierwszym sprawdziliśmy działanie algorytmu lokalnego dla domyślnie ustalonych parametrów. W zastosowanym algorytmie Hill Climbing istnieje możliwość zmiany dwóch parametrów:

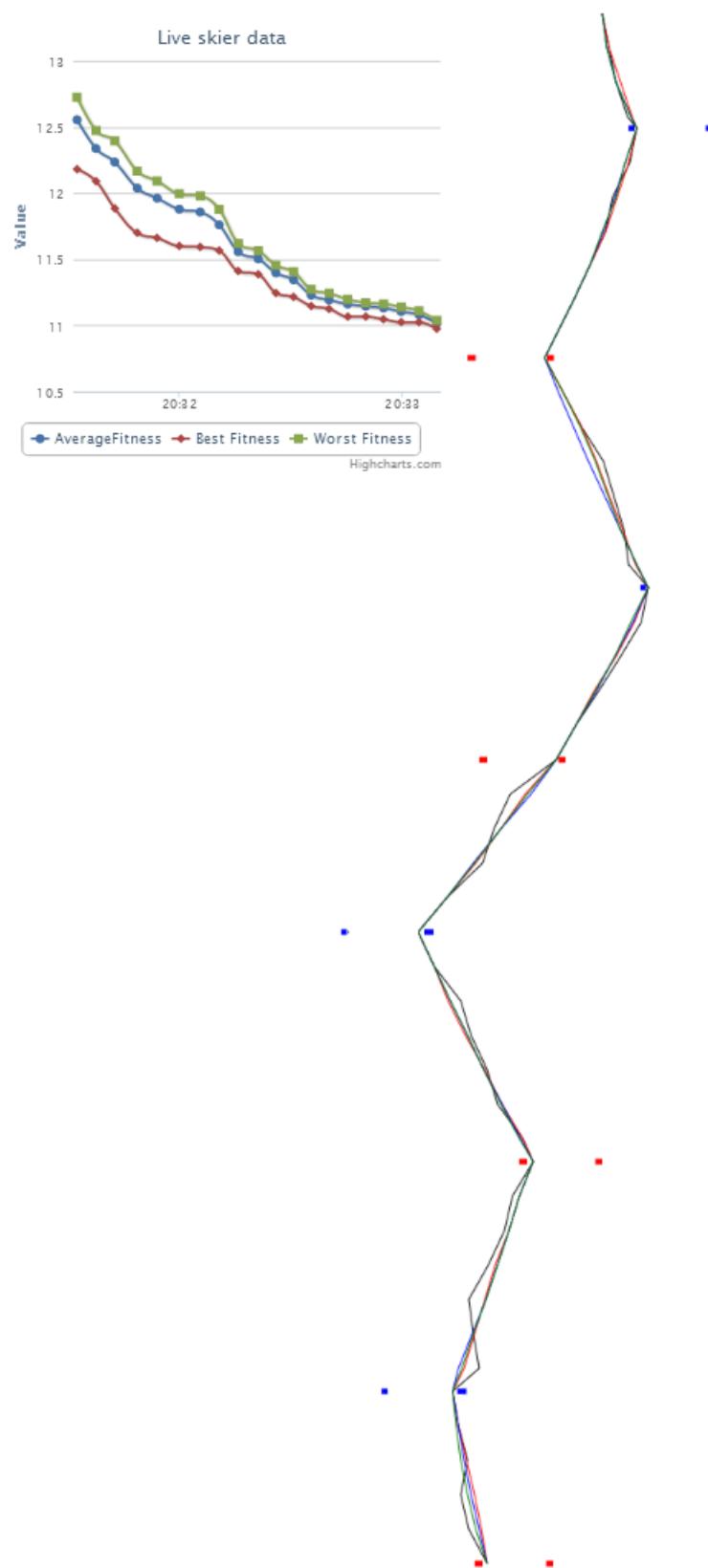
- rozmiaru kroku,
- tzw. przyspieszenia (ang. *acceleration*).

Szczegółowy opis parametrów można znaleźć w podrozdziale 3.4. W domyślnym ustawieniu wartości tych parametrów wynoszą odpowiednio 0.5 (odpowiadająca 0.5 metra 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 uruchomiony zostanie trzy razy algorytm lokalnej optymalizacji, ale z zestawami parametrów przedstawionymi w tabeli 6.9.

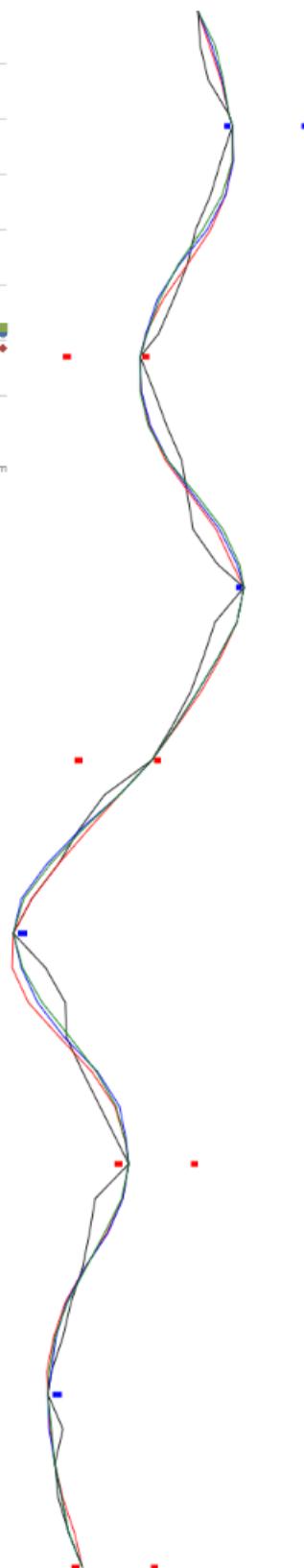
Jak zostanie pokazane w następnym eksperymencie, stosowanie mechanizmu karania poprawia otrzymywane wyniki, więc ten sam eksperiment wykonamy także z zastosowaniem tego mechanizmu.

Tablica 6.9: Wartości parametrów algorytmu Hill climbing w kolejnych eksperymentach

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



Rysunek 6.32: Wyniki kolejnych prób eksperymentu z różnymi wartościami parametrów algorytmu HC



Rysunek 6.33: Wyniki kolejnych prób eksperymentu z różnymi wartościami parametrów algorytmu HC z zastosowaniem PF

Rysunki 6.32 i 6.33 pokazuje otrzymane trasy dla różnych algorytmów i parametrów:

- trasa czarna - trasa uzyskana w wyniki działania algorytmu ewolucyjnego,
- 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. Małe zróżnicowanie wynika z tego, że początkowa trasa, którą należało zoptymalizować była w każdym przypadku taka sama. Algorytm Hill Climbing jest algorymem deterministycznym, ale zmiana parametrów tego algorytmu wpływa na otrzymywany wynik końcowy. W przypadku pierwszym, bez zastosowania mechanizmu karania 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.

Przeanalizujmy teraz dokładniej otrzymane wyniki zebrane w tabelach 6.10 oraz 6.11.

Czas działania algorytmu ewolucyjnego w pierwszym przypadku to 181 s, w eksperymencie z karaniem 253 s. Czasy uzyskane w wyniku działania algorytmu ewolucyjnego w każdej z prób wynoszą odpowiednio 10.974 s oraz 10.895 s.

Tablica 6.10: Wyniki dla różnych parametrów HC bez mechanizmu karania

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

Tablica 6.11: Wyniki dla różnych parametrów HC z PF

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

W pierwszej części eksperymentu najlepszy czas uzyskany 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.

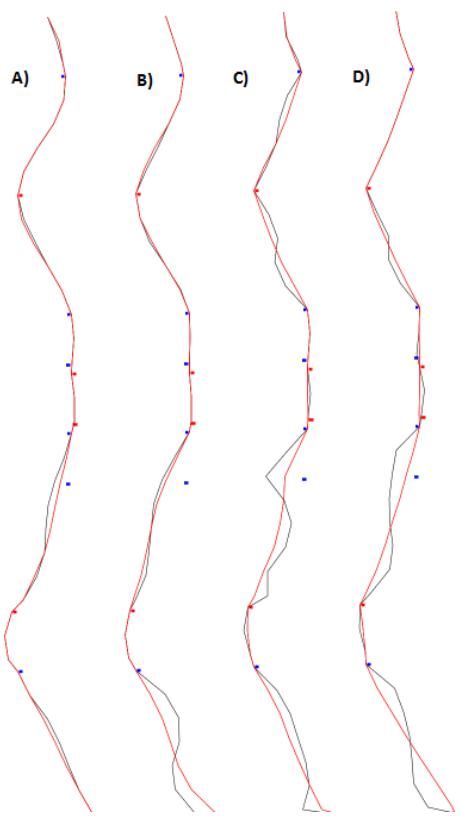
Drugi przypadek jest trudniejszy do analizy, ponieważ wprowadzenie karania powoduje, że nie jesteśmy w stanie przenieść uzyskiwanych ocen do rzeczywistości. Według nich, znów najlepszy okazuje się być zestaw parametrów z drugiego przypadku, kiedy jednak sprawdzimy w tabeli uzyskiwane czasy, widzimy, że są one gorsze niż najlepszy uzyskany wynik z algorytmu ewolucyjnego, 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 przedstawienie nart w odpowiednim kierunku, co jest niemożliwe bez utraty prędkości.

6.2.3. Strategie karania

Testowane zostały wszystkie opisane w rozdziale 5.5 na stronie 40 strategie karania. Testy miały na celu zoorientowanie się czy wszystkie strategie będą przede wszystkim prowadzić do poprawnego wyniku - tj. wizualnie poprawnej i płynnej trasy. Przy okazji sprawdzany był wpływ kolejnych strategii karania na szybkość zbieżności algorytmu ewolucyjnego oraz lokalnej optymalizacji, a co za tym idzie do najszybszego znajdowania wyników. Dla każdej testowanej strategii, ta sama strategia karania była użyta w algorytmie ewolucyjnym, jak i w lokalnej optymalizacji.

Poprawność strategii



Rysunek 6.34: Tor A przedstawia trasy obliczone przy użyciu strategii sumowania pochodnych, Tor B przy strategii zmniejszania wektorów, Tor C strategii karania za przekrawędziowanie. Dla torów zarówno A, B i C - czarny tor to według wprowadzonych wcześniej oznaczeń ES + PF, czerwony tor to ES + HC + PF. Tor D przedstawia trasy bez zastosowania żadnej strategii karania - czarny tor to ES, czerwony tor to ES + HC

Rysunek 6.34 przedstawia znalezione tory przejazdu przy zastosowaniu każdej z tych strategii. Jak widać, można dostrzec znaczącą różnicę w torach znalezionych przez algorytm ewolucyjny (czarna linia) w zależności od wyboru strategii. W przypadku lokalnej optymalizacji różnice są już mniejsze, jednak pierwsze dwie strategie wypadają w tym przypadku lepiej niż pozostałe, zwłaszcza na pierwszym odcinku trasy.

Szybkość obliczeń

Tabela 6.12 na stronie 84 pokazuje średni czas uzyskania wyniku dla każdej ze strategii jak również dla braku zastosowania strategii. Najszybszą zbieżność mają metody, które prowadzą do najlepszych jakościowo rozwiązań (C i D z rysunku 5.6). Jak widać najszybszą zbieżność algorytmu ewolucyjnego uzyskuje się dla karania poprzez zliczanie ilości przekrawędziowań. Różnice pomiędzy średnimi szybkościami są znaczne, bo nawet czterokrotne.

Zastosowanie strategii A prowadzi do uzyskania bardzo dobrego jakościowo wyniku, dobrego do tego stopnia, że lokalna optymalizacja nie wprowadza znacznej poprawy. Strategia B, nie dość, że prowadzi do bardzo dobrego wyniku jakościowego, to zbiega się dość szybko w porównaniu ze strategią A - o ponad 40% szybciej.

Tablica 6.12: Porównanie długości trwania obliczeń instancji problemu dla użytych różnych strategii karania. Obliczenia prowadzone są na maszynie o następujących parametrach: Procesor Intel Core i5-2410M, 2.3 GHz, 8 GB pamięci RAM, System Windows 7 Home Premium, zminimalizowane obciążenie procesora przez inne procesy, obliczenia prowadzi jedna zakładka przeglądarki Chrome

	Strategia 1 - sumowanie pochodnych	Strategia 2 - proporcjonalne zmniejszanie długości wektora	Strategia 3 - zliczanie przekrawędziowania	Strategia 4 - brak karania
Czas rozwiązania kolejnego problemu w sekundach	91.272	48.241	18.798	31.264
	66.032	44.404	17.888	37.16
	117.033	54.362	20.441	27.404
	87.8	59.479	19.971	18.545
	68.612	42.383	21.276	25.054
	Średni czas rozwiązania	86.1498	49.7738	19.6748
				27.8854

Wybór strategii

Rozważając wszystkie strategie, zostało przyjęte, że najlepszym modelem jest model ze strategią proporcjonalnego zmniejszania długości wektora prędkości w zależności od kątów pomiędzy kolejnymi odcinkami łamanej - czyli Strategia B. Czas obliczeń dla tej strategii nie jest ani największy ani najmniejszy. Ta właśnie strategia, jako najlepiej obrazująca rzeczywistość i pozwalająca na otrzymywanie najlepszych wyników, została wybrana jako mechanizm karania dla innych eksperymentów, chyba, że jest to zaznaczone inaczej.

6.3. Volunteer Computing

Aby zweryfikować zaproponowane rozwiązanie architektoniczne zaplanowany został szereg eksperymentów mających na celu zbadanie:

- wydajności poszczególnych przeglądarek dla intensywnych obliczeń (Chrome, Firefox, Internet Explorer) oraz znalezienie maksymalnej prędkości obliczeń na jednej maszynie, poprzez znalezienie optymalnej konfiguracji - liczby uruchomionych na raz kart przeglądarki
- skuteczności rekrutowania ochotników do obliczeń typu Volunteer Computing w przeglądarce poprzez internetowe media społecznościowe

6.3.1. 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,
- rozwiązywana jest ta sama instancja problemu (ta sama konfiguracja slalomu),
- dana przeglądarka ma uruchomioną tylko jednąinstancję (liczbę niezależnych okien),
- instancja uruchomionej przeglądarki ma otwartą tylko określona 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 w sumie, licząc wszystkie zakładki przeglądarki, liczba obliczonych rozwiązań problemu (A) wynosi co najmniej dziesięć,
- 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,
- wyliczane jest tempo prowadzonych obliczeń - liczbę sekund potrzebnych na rozwiązanie jednego problemu:

$$P = \frac{T_{MAX}}{A} [\frac{s}{rozwiązanie}]$$

Tablica 6.13: Niezależnie uruchomione obliczenia w każdej z przeglądarek, prowadzone w jednej zakładce.

	Chrome	Firefox	IE
Numer zakładki	0	0	0
Czas rozwiązania kolejnego problemu [s]	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 - Liczba obliczonych rozwiązań	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 6.35 na stronie 88. 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 obliczeń prowadzonych w wielu kartach przeglądarki jednocześnie. Dla każdej z przeglądarek, zdecydowanie wydajniej jest uruchamiać obliczenia w dwóch zamiast w jednej zakładce jednocześnie. Równolegle 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 wyniki 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 liczba zakładek, to między 6 a 8. Przy 9 zakładkach, tempo istotnie spadło - bo prawie o 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 całkowite tempo spada. Uruchomienie obliczeń w 9 zakładkach równocześnie okazało się na testowej maszynie niemożliwe, gdyż powodowało awaryjne zakończenie działania programu 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. Obserwacje te ilustrują zrzuty ekranu 6.36 i 6.37 na stronie 89 z managera zadań systemu podczas prowadzenia obliczeń oraz wyniki w tabelach 6.14, 6.15 i 6.16 na stronie 87 na których widać, że czas rozwiązywania problemu jest zbliżony dla każdej z kart.

Tablica 6.14: Uruchomienie obliczeń w sześciu zakładkach przeglądarki Chrome równocześnie.

Chrome						
Numer zakładki	0	1	2	3	4	5
Czas rozwiązania kolejnego problemu [s]	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 - Liczba obliczonych rozwiązań	24					
P - Tempo [s/rozwiązanie]	27.9					

Tablica 6.15: Uruchomienie obliczeń w sześciu zakładkach przeglądarki Firefox równocześnie.

Firefox						
Numer zakładki	0	1	2	3	4	5
Czas rozwiązania kolejnego problemu [s]	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 - Liczba obliczonych rozwiązań	14					
P - Tempo [s/rozwiązanie]	153.6					

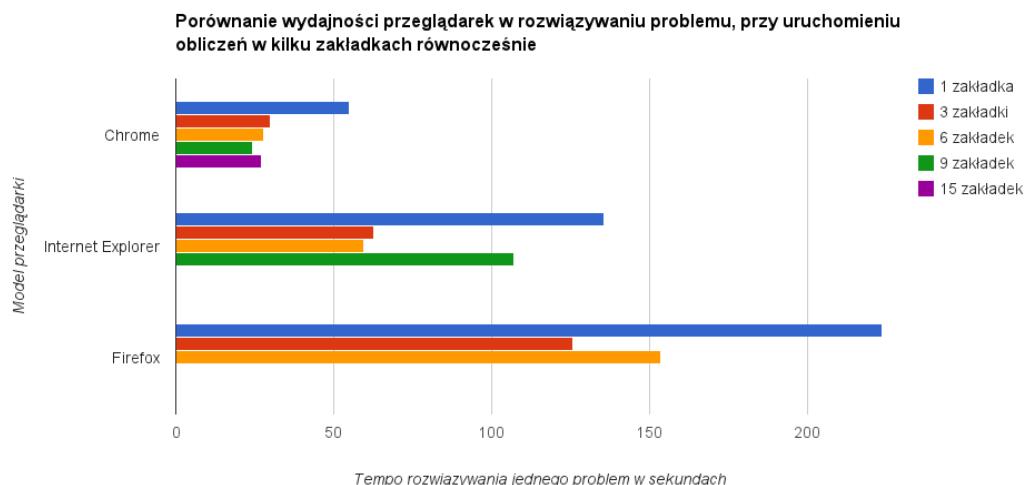
6.3.2. Skuteczność rekrutowania ochotników do obliczeń w przeglądarkowym modelu Volunteer Computing

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:

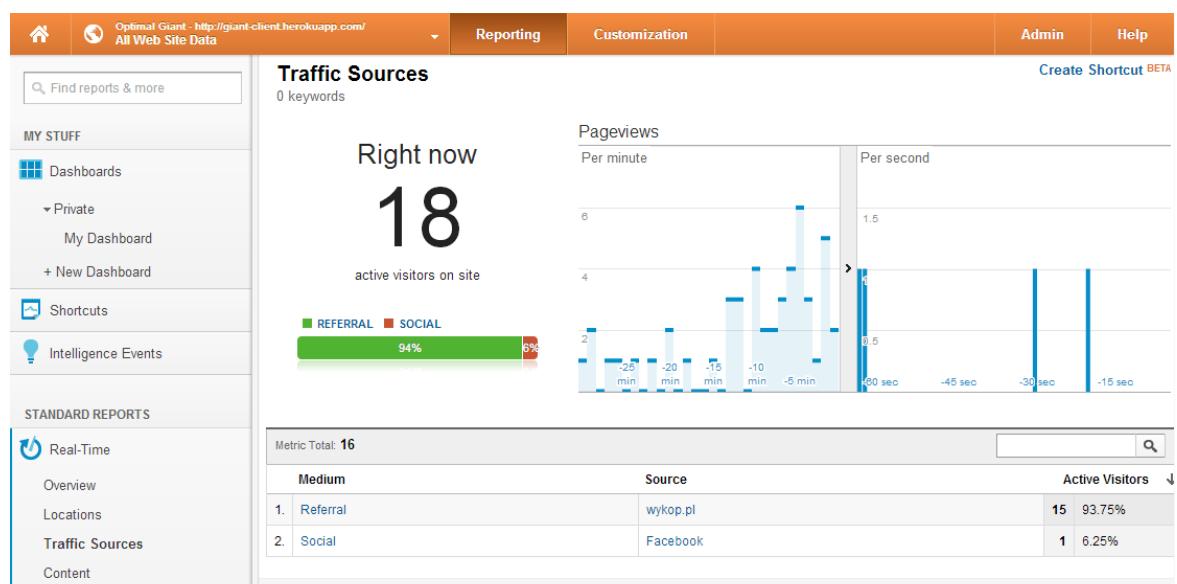
- na prywatnej tablicy w serwisie Facebook (520 znajomych),
- na tablicy Studenckiego Klubu Narciarskiego FIRN AGH w serwisie Facebook (180 obserwujących),
- w serwisie Wykop.pl,
- z prywatnego konta w serwisie Twitter (120 obserwujących).

Tablica 6.16: Uruchomienie obliczeń w sześciu zakładkach przeglądarki Internet Explorer równocześnie

Internet Explorer						
Numer zakładki	0	1	2	3	4	5
Czas rozwiązania kolejnego problemu [s]	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 - Liczba obliczonych rozwiązań				21		
P - Tempo [s/rozwiązań]				59.7		

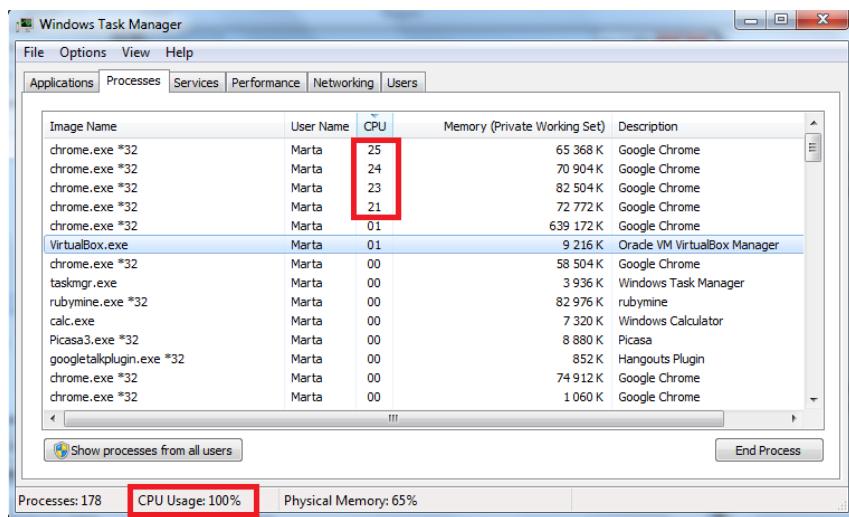


Rysunek 6.35: Porównanie wydajności różnych modeli przeglądarek w rozwiązywaniu problemu, przy uruchomieniu obliczeń w kilku zakładkach równocześnie.

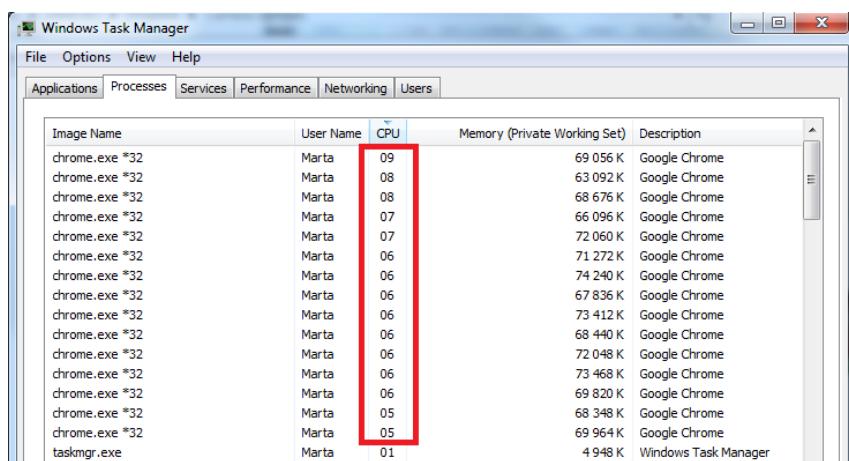


M. Rylko, A. Skiba *Optymalizacja toru przejazdu w narciarstwie alpejskim przy wykorzystaniu Volunteer Computing*

Rysunek 6.38: Zrzut ekranu z panelu Google Analytics pokazujący aktualną wówczas liczbę kontrybuujących w obliczeniach ochotników oraz źródła z jakich trafiły na stronę



Rysunek 6.36: Podział zużycia procesora dla czterech zakładek przeglądarki Chrome prowadzącej obliczenia.



Rysunek 6.37: Podział zużycia procesora dla piętnastu zakładek przeglądarki Chrome prowadzącej obliczenia.

Przez sześć godzin 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 ($6 \text{ h} = 21600 \text{ s}$; Tempo $23.9 \text{ s} / 1 \text{ rozwiązanie}$) Ten sam efekt udało się uzyskać, dzięki zaprzegnięciu do obliczeń ochronników, w niecałe 2 godziny. Nie jest to wielkie przyspieszenie, 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, prosili o wyjaśnienie dlaczego kon-

figuracja 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.

6.4. Wnioski

W tej sekcji zostaną wymienione wady i zalety zaproponowanego rozwiązania opierającego się na modelu Volunteer Computing udostępnianego za pomocą przeglądarek internetowych (*Web browser based volunteer computing*).

Zalety

- łatwa dostępność,
- brak konieczności instalacji dodatkowego oprogramowania po stronie ochotników,
- brak bariery wejścia do używania nowego oprogramowania - wszystko dzieje się w znany 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 zaproponowanego rozwiązania do analogicznych obliczeń. Jest to przede wszystkim propagowanie wiedzy o problemach naukowych - łatwy dostęp i atrakcyjna wizualizacja, która pozwala na wyjaśnienie 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 zadanej liczbie zakładek przeglądarki),
- mało wydajna implementacja obliczeń - implementacja w języku JavaScript,
- niewielka ilość bibliotek numerycznych i naukowych do obliczeń w środowisku JavaScript.

Zaproponowana przez nas platforma nie jest w tym momencie konkurencyjna dla obecnie funkcjonujących systemów ze względu na wydajność i ograniczoną konfigurowalność. Jednak jej zdecydowanie pozytywnym aspektem, jest mała bariera wejścia dla ochotników chcących brać udział w obliczeniach, jak również sam ułatwiony proces rekrutowania ochotników do obliczeń. Powyższe wpływa na to, iż mimo że wydajność pojedynczego węzła jest znaczco mniejsza w stosunku do węzłów obliczeniowych w tradycyjnych systemach opartych o instalowane na komputerach aplikacje, potencjał na stworzenie dużej sieci jest bardzo duży.

Zaproponowana koncepcja jest ciekawym, komplementarnym do tradycyjnego podejścia rozwiązaniem. Dzięki łatwej dostępności i braku bariery wejścia może mieć dodatkowe zastosowanie chociażby do promowania koncepcji Volunteer Computing.

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 4.4 na stronie 27. Dzięki niej możemy nie tylko wspomagać projekt swoimi zasobami, ale również na bieżąco śledzić wizualizowane w 3D postępy obliczeń. Największym problemem jest właśnie bariera wejścia. Proponujemy zatem, by stworzyć prosty prototyp, który będzie symułował przystąpienie do obliczeń za pośrednictwem przeglądarki internetowej oraz wykonywał i wizualizował - najprostsze choćby operacje w jakikolwiek sposób związane z projektem fightAIDS@home. Osoba, która trafi na stronę internetową z takim "lekkim" klientem Volunteer Computing odczuje - choćby przez to, że usłyszy, że procesor jej PC-ta zaczął intensywnie pracować - jak to jest wspomagać projekt swoimi zasobami w szczytnym celu. Taka osoba powinna dostać bardzo szybko informację zwrotną, dotyczącą 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 wyobrazić, 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.

Bibliografia

- [1] M. Andrew and D. Brodie. Optimisation of performance in alpine ski racing with fusion motion capture, 2009.
- [2] J. Arabas. *Wykłady z algorytmów ewolucyjnych*. Wydawnictwo WNT, Warszawa, 2004.
- [3] Caroline Barelle. *Sport Aerodynamics: on the Relevance of Aerodynamic Force Modelling versus Wind Tunnel Testing*. InTech, 2011.
- [4] Hans-Georg Beyer. Evolution strategies. *Scholarpedia*, 2(8):1965, 2007.
- [5] Xianshun Chen, Yew-Soon Ong, Meng-Hiot Lim, and Kay Chen Tan. A multi-facet survey on memetic computation. *IEEE Trans. Evolutionary Computation*, 15(5):591–607, 2011.
- [6] the European Space Agenc Chris Talbot. The science of ski waxes, January 2003.
- [7] Richard Dawkins. *The selfish gene*. Oxford University Press, Oxford New York, 1989.
- [8] Roman Dębski. Gradient-based algorithms in the brachistochrone problem having a black-box represented mathematical model. *Journal of Telecommunications and Information Technology*, 2014.
- [9] Roman Dębski. Simulation-based high-performance algorithms for alpine ski racer's trajectory optimization in heterogeneous computer systems. *International Journal of Applied Mathematics and Computer Science*, 2014.
- [10] Alex Freire. The brachistochrone problem, 2008.
- [11] Web Hypertext Application Technology Working Group. Web workers - specification, September 2013.
- [12] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.
- [13] Bloomberg Olga Kharif. Using smartphones to cure diseases while you sleep, August 2013.
- [14] Frank Porter. *Calculus of Variations*, 2011.
- [15] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Technical University of Berlin, Department of Process Engineering, 1971.
- [16] Ingo Rechenberg. *Evolutionsstrategie; Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart-Bad Cannstatt, 1973.
- [17] S. Russell and P. Norvig. *Artificial Intelligence. A Modern Approach*. Prentice Hall, New Jersey, 2003.
- [18] H.-P. Schwefel. Kybernetische evolution als strategie der exprimentellen forschung in der strömungstechnik. diploma thesis, Technical University of Berlin, 1965.

- [19] H.-P. Schwefel. *Evolutionsstrategie und numerische optimierung*. 1975.
- [20] H.-P. Schwefel. *Numerische Optimierung von ComputerModellen mittels der Evolutionsstrategie*. Birkhäuser, Basle, Switzerland, 1977.
- [21] H.-P. Schwefel. *Numerical optimization of computer models*. Wiley, Chichester New York, 1981.
- [22] H.-P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.
- [23] Strona Wiki tworzona przez wielu autorów. Volunteer computing, October 2012.