

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

---

Wydział Informatyki, Elektroniki i Telekomunikacji

KATEDRA INFORMATYKI



**PRACA MAGISTERSKA**

**MARTA RYŁKO, ANNA SKIBA**

**RÓWNOLEGŁE ALGORYTMY OPTIMALIZACJI TORU  
PRZEJAZDU W NARCIARSTWIE ALPEJSKIM**

PROMOTOR:  
dr inż. Roman Dębski

Kraków 2013

## **OŚWIADCZENIE AUTORA PRACY**

OŚWIADCZAM, ŚWIADOMY ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ WYKONAŁEM OSOBIŚCIE I SAMODZIELNIE, I NIE KORZYSTAŁEM ZE ŹRÓDEŁ INNYCH NIŻ WYMNIENIONE W PRACY.

.....

PODPIS

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

---

Faculty of Computer Science, Electronics and Telecommunication

DEPARTMENT OF COMPUTER SCIENCE



MASTER OF SCIENCE THESIS

MARTA RYŁKO, ANNA SKIBA

**PARALLEL ALGORITHMS FOR SKI-LINE OPTIMISATION IN  
ALPINE SKI RACING**

SUPERVISOR:  
Roman Dębski Ph.D

Krakow 2013

Serdecznie dziękuję ...

## Spis treści

<b>1. Wstęp</b>	7
1.1. Cele pracy	8
1.2. Zawartość pracy	8
<b>2. Wprowadzenie teoretyczne</b>	9
2.1. Fizyczny model narciarza	9
2.2. Metody numeryczne rozwiązywania równań różniczkowych	9
2.3. Optymalizacja	9
2.3.1. Algorytm ewolucyjny	10
2.3.2. Hill climbing	13
2.4. Uczenie maszynowe	13
2.4.1. Uczenie się ze wzmocnieniem	14
2.5. Volunteer Computing	15
2.6. Web Workers	16
<b>3. Istniejące rozwiązania</b>	17
3.1. Platformy do Volunteer Computing	17
3.1.1. Great Internet Mersenne Prime Searchy	17
3.1.2. Distributed.net	18
3.1.3. Berkeley Open Infrastructure for Network Computing	19
3.1.4. Folding@home	19
<b>4. Proponowane rozwiązanie</b>	21
4.1. Model narciarza i środowiska	21
4.2. Opis matematyczny modelu	21
4.3. Numeryczne rozwiązanie problemu	22
4.3.1. Rozwiązanie w 3D	23
4.4. Optymalizacja toru przejazdu	24
4.4.1. Algorytm ewolucyjny	24
4.4.2. Hill climbing	25
4.5. Architektura systemu	26
<b>5. Wyniki</b>	27
5.1. Optymalizacja	27
5.1.1. Algorytm ewolucyjny	27
5.1.2. Lokalna optymalizacja	27

---

5.2.	Uczenie maszynowe .....	27
5.3.	Podsumowanie.....	27
5.4.	Optymalizacja toru przejazdu.....	27
5.5.	Architektura systemu.....	27
<b>6.</b>	<b>Podsumowanie.....</b>	<b>28</b>
6.1.	Podrozdział.....	28

# 1. Wstęp

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

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

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

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

Kluczową częścią naszego rozwiązania jest wykorzystanie algorytmu genetycznego do wybrania pewnego lokalnego optimum trasy, a następnie przeprowadzamy lokalną optymalizację celem wygładzenia znalezionej trasy. Aby przyspieszyć obliczenia, zastosowałyśmy architekturę opartą o rozproszonych klientów wykonujących obliczenia i raportujących do głównego serwera. Na podstawie zebranych danych od klientów, serwer jest w stanie dostarczyć rozwiązanie szybciej oraz można mieć większą pewność, iż jest ono jeśli nie optymalne, to bardzo bliskie optymalnego. Obliczenia wykonywane są w środowisku przeglądarek internetowych w języku JavaScript.

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

## 1.1. Cele pracy

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

## 1.2. Zawartość pracy

W rozdziale ?? przedstawiono podstawowe informacje dotyczące struktury dokumentów w  $\text{\LaTeX}$ u.  $\text{\LaTeX}$  jest językiem



## 2. Wprowadzenie teoretyczne

W rozdziale tym przedstawiono informacje .

### 2.1. Fizyczny model narciarza

### 2.2. Metody numeryczne rozwiązywania równań różniczkowych

### 2.3. Optymalizacja

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

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

#### **Optymalizacja lokalna i globalna**

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

- hill climbing
- przeszukiwanie tabu

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

### 2.3.1. Algorytm ewolucyjny

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

Podstawą całego algorytmu jest populacja osobników, z których każdy reprezentuje rozwiązanie problemu. Populacja ta zmienia się wraz z działaniem algorytmu. Ewolucja zakłada, że populacja będzie się składać z coraz lepiej przystosowanych osobników. Przystosowanie to jest obliczane za pomocą wcześniej określonej funkcji oceniającej jakość danego osobnika, czyli jak dobre jest rozwiązanie reprezentowane przez niego. Przystosowanie jest wartością liczbową obliczoną za pomocą tej funkcji przystosowania.

Funkcja przystosowania określa wartość przystosowania osobnika na podstawie jego fenotypu, który jest tworzony z genotypu. Genotyp określa zestaw cech danego osobnika i składa się z chromosomów (najczęściej z jednego). Natomiast każdy z chromosomów składa się z elementarnych jednostek - genów.

#### Schemat działa algorytmu ewolucyjnego

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

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

- reprodukcja
- operacje genetyczne
- ocena
- sukcesja

Często reprodukcję i sukcesję łączy się pod nazwą selekcja.

Reprodukcja powoduje powielenie losowo wybranych osobników z populacji. Prawdopodobieństwo wybrania osobnika do powielenia najczęściej jest proporcjonalne do jego przystosowania. Może się zdarzyć, że dany osobnik zostanie wybrany więcej niż raz, a także, że nie zostanie wybrany ani razu.

Następnie na tych kopiach przeprowadzane są operacje genetyczne powodujące zmiany w genotypie osobników. Wyróżniamy dwie podstawowe operacje:

- mutacja
- krzyżowanie

Zadaniem mutacji jest losowe zmodyfikowanie genów w genotypie.

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

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

Zakończenie działania algorytmu przeważnie opiera się na badaniu funkcji przystosowania całej populacji. Jeśli wartość przystosowania populacji nie jest zróżnicowana mówimy o stagnacji algorytmu i może być to wskazaniem

do zakończenia działania algorytmu. Czasem jednak oczekuje się aż przystosowanie to będzie wystarczająco duże, żeby stwierdzić, że znalezione rozwiązanie jest bardzo dobre. Przeważnie jednak nie znamy nawet przybliżonej wartości jakości rozwiązania, więc nie możemy stwierdzić kiedy przystosowanie jest odpowiednie i czy nie może się jeszcze znacznie poprawić.

### Kodowanie osobników

W przypadku algorytmów genetycznych, będących szczególnym przypadkiem algorytmów ewolucyjnych, do kodowania osobników stosuje się kodowanie binarne chromosomów. Pojedynczy bit reprezentuje zatem gen należący do chromosomu.

W takim przypadku mutacja wykonywana jest na każdym genie osobno z pewnym prawdopodobieństwem, jeśli do niej dochodzi, zmienia się wartość bitu na przeciwną. W krzyżowaniu wybiera się dwa osobniki rodzicielskie, których chromosomy rozcinane są na dwie części i łączone "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. Najlepiej traktować algorytmy ewolucyjne jako metaheurystykę - określony jest pewien szkic algorytmu, który można dostosowywać do konkretnego rozwiązania. W tym podrozdziale opisane są podstawowe i najbardziej popularne schematy postępowania oparte o algorytmy ewolucyjne.

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

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

Zatrzymanie algorytmu może być dokonane jeśli np.:

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

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

**Strategia (1+1)** Strategia (1+1) jest podstawową strategii ewolucyjnych. W algorytmie tym mamy doczynienia z populacją składającą się z tylko jednego osobnika posiadającego jeden chromosom. W każdej pętli algorytmu dokonuje się mutacji tego chromosomu, co powoduje powstanie nowego osobnika. Osobnik ten jest poddawany

ocenie, a następnie dokonuje się wyboru lepszego z dwóch istniejących osobników i tego pozostawia w populacji. W mutacji dodaje się do każdego genu chromosomu losową modyfikację rozkładem normalnym:

$$Y_i^t = X_i^t + \sigma \xi_{N(0,1),i} \quad (2.1)$$

Wartość  $\sigma$  będzie powodowała większe lub mniejsze zmiany w chromosomie. Jeśli chcemy przeszukać przestrzeń, powinniśmy zwiększać jej wartość, co jest pożądane zwłaszcza w początkowej fazie działania algorytmu. Natomiast, aby znaleźć jak najlepsze rozwiązanie, wiedząc że obecne rozwiązanie jest już bardzo bliskie najlepszemu, możemy zmniejszać wartość  $\sigma$  przeszukując tylko najbliższą przestrzeń.

Do wyznaczania  $\sigma$  powstał następujący algorytm zwany regułą 1/5 sukcesów:

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

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

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

W strategii tej ważne jest też kodowanie, do którego dodatkowo dołożono również chromosom przechowujący wektor  $\sigma$  zawierający wartości odchyłeń standardowych, które wykorzystuje się w trakcie mutacji.

Po wylosowaniu wartości zmiennej losowej o rozkładzie normalnym ( $\xi_{N(0,1)}$ ) dla każdego elementu wektora  $\sigma$  losujemy jeszcze jedną zmienną losową o rozkładzie normalnym ( $\xi_{N(0,1),i}$ ) i oblicza nowe wartości odchyłeń z wektora  $\sigma$ :

$$\sigma'_i = \sigma_i e^{(\tau' \xi_{N(0,1)} + \tau \xi_{N(0,1),i})} \quad (2.2)$$

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

$$\tau = \frac{K}{\sqrt{2n}} \quad (2.3)$$

$$\tau' = \frac{K}{\sqrt{2\sqrt{n}}} \quad (2.4)$$

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

$$X'_i = X_i + \sigma'_i \xi_{N(0,1),i} \quad (2.5)$$

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

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

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

**Strategia  $(\mu, \lambda)$**  Strategia  $(\mu + \lambda)$  posiada pewne wady, które postanowiono spróbować wyeliminować za pomocą nowej strategii  $(\mu, \lambda)$ . Poprzedni algorytm sprawia problemy jeśli w populacji pojawia się osobnik o wysokiej wartości przystosowania, ale posiadający zbyt duże (albo zbyt małe) wartości odchyłeń standardowych. Usunięcie takiego osobnika z populacji często nie jest procesem krótkotrwałym, gdyż wpływa on na powstające potomstwo, przekazując mu podobne do jego, nieodpowiednie wartości odchyłeń.

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.

### 2.3.2. Hill climbing

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

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

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

## 2.4. Uczenie maszynowe

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

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

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

- uczenie z nadzorem
- uczenie bez nadzoru

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

Istnieją jeszcze dwie grupy, które trudno zakwalifikować do powyższych:

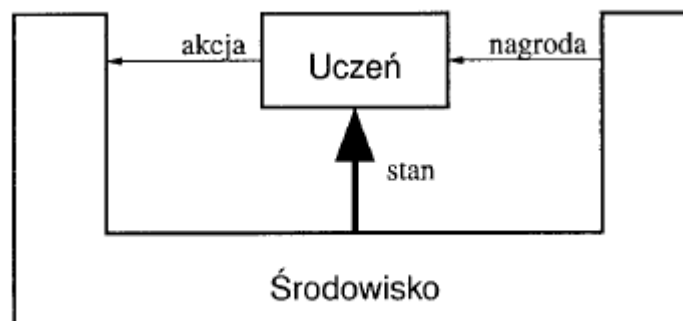
- uczenie się na podstawie zapytań
- uczenie się przez eksperymentowanie
- uczenie się ze wzmocnieniem

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

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

### 2.4.1. Uczenie się ze wzmocnieniem

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



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

Schemat algorytmu przedstawia się następująco:

Dla kolejnych kroków czasowych  $t$ :

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

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

### Strategia maksymalizacji nagród

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

W przypadku opóźnionego wzmocnienia wprowadza się współczynnik dyskontowania  $\gamma \in [0, 1]$ . Zadaniem ucznia jest zmaksymalizowanie zdyskontowanej sumy nagród:

$$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \quad (2.6)$$

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

### Zadania epizodyczne

## 2.5. Volunteer Computing

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

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

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

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

Volunteer Computing pierwotnie zakładał, że obliczenia są wykonywane na zwykłych PC-tach. Ilość komputerów tego typu jest nieporównywalnie większa do ilości wyspecjalizowanych komputerów o dużej mocy obliczeniowej i jest szacowana na ponad miliard. Dodatkowo, z przyczyn ekonomicznych, na rozwój tych maszyn producenci sprzętu przeznaczają największe fundusze więc ich moc i zdolności obliczeniowe stale rosną.

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

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

## 2.6. Web Workers

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

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

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

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



## 3. Istniejące rozwiązania

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

### 3.1. Platformy do Volunteer Computing

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

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

Zlecniodawcy obliczeń, zauważyli, że na świecie popularność zdobywają inne od komputerów osobistych urządzenia, które również dysponują nie wykorzystywaną mocą obliczeniową. Konsola Play Station 3 była pierwszym urządzeniem tego typu, włączonych do projektów wykorzystujących Volunteer Computing.

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

W tym rozdziale, zaprezentujemy przekrój obecnie najbardziej znaczących platform umożliwiających wykonywanie obliczeń w modelu Volunteer Computing.

#### 3.1.1. Great Internet Mersenne Prime Searchy

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

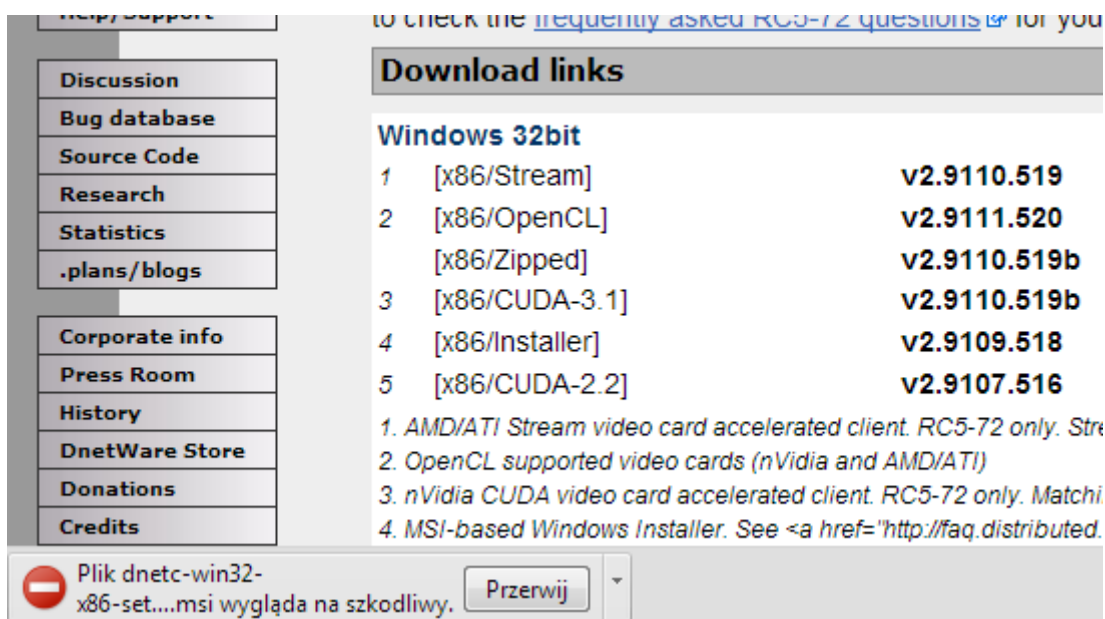
System składa się z aplikacji klienckich stworzonych dla 9 platform i konfiguracji systemów. Twórcy zapewniają, że kod programu jest wysoce zoptymalizowanym kodem assemblerowym Intel'a. Po uruchomieniu, program nawiązuje kontakt z serwerem nazwanym PrimeNet aby otrzymać część pracy.

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

### 3.1.2. Distributed.net

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

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



Ufając dostawcy, po zdecydowaniu się na zainstalowanie programu, otwiera się bardzo mało przyjazny interfejs konsolowy.

```

distributed.net client
Client Edit View Help
[Aug 17 20:13:35 UTC] Automatic processor detection found 4 processors.
[Aug 17 20:13:35 UTC] Loading crunchers with work...
[Aug 17 20:13:36 UTC] Connected to euro.v29.distributed.net:2064...
[Aug 17 20:13:36 UTC] The keyserver says: "Hai c0w! (cdy)"
[Aug 17 20:13:36 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 unit 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
[Aug 17 20:14:21 UTC] recognize the processor (tag: "100062A/")
[Aug 17 20:14:59 UTC] OGR-NG: Running micro-bench to select fastest core...
[Aug 17 20:14:59 UTC] OGR-NG: using core #3 (cj-asm-sse2).
[Aug 17 20:14:59 UTC] OGR-NG #a: Loaded 27/23-21-5-4-24-8
[Aug 17 20:14:59 UTC] OGR-NG #b: Loaded 27/23-21-5-4-24-12
[Aug 17 20:14:59 UTC] OGR-NG #c: Loaded 27/23-21-5-4-24-13
[Aug 17 20:14:59 UTC] OGR-NG #d: Loaded 27/23-21-5-4-24-15
[Aug 17 20:14:59 UTC] OGR-NG: 92 packets remain in buff-in.og2
[Aug 17 20:14:59 UTC] OGR-NG: 0 packets are in buff-out.og2
[Aug 17 20:14:59 UTC] 4 crunchers ('a'-'d') have been started.
[Aug 17 20:14:59 UTC] OGR-NG #a:27/23-21-5-4-24-8 [0]
[Aug 17 20:15:00 UTC] RC5-72: using core #0 (SES 1-pipe).
[Aug 17 20:15:19 UTC] RC5-72: Benchmark for core #0 (SES 1-pipe)
[Aug 17 20:15:19 UTC] 0.00:00:16.38 [4,813,691 keys/sec]
[Aug 17 20:15:19 UTC] RC5-72: using core #1 (SES 2-pipe).

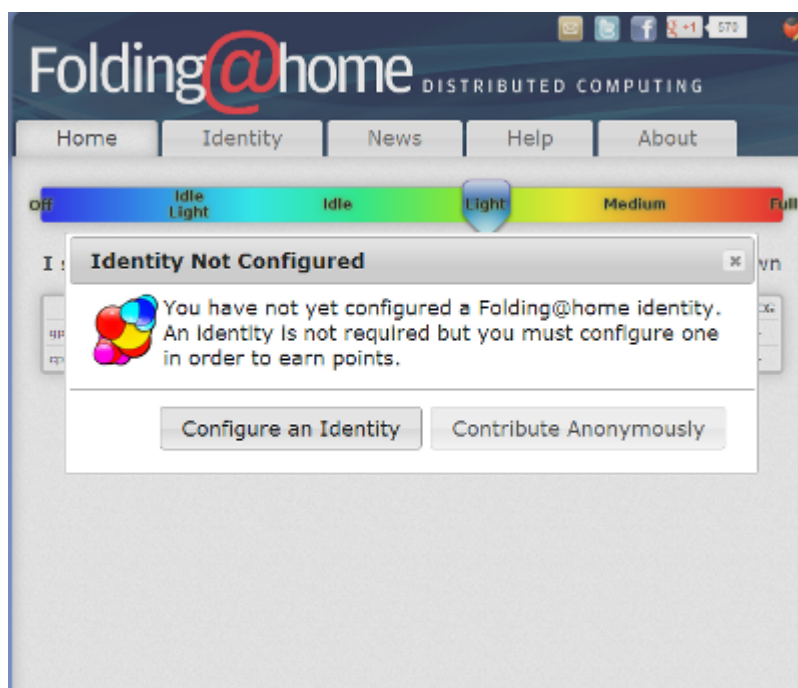
```

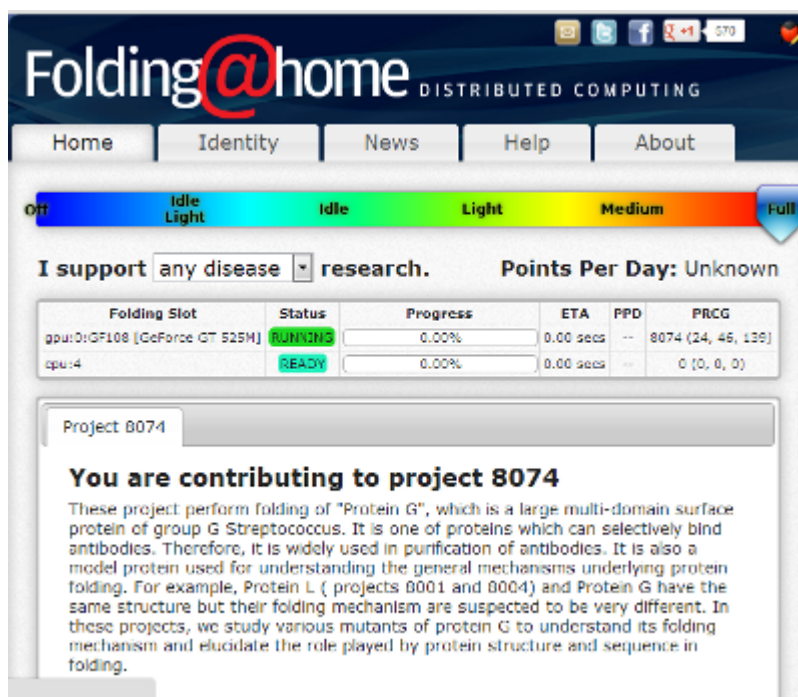
Nasz wniosek jest taki, że przestarzałe i nieatrakcyjne systemy do Volunteer Computing nie mają racji bytu w dzisiejszym świecie i prędzej czy później wyginą na rzecz bardziej przyjaznych rozwiązań.

### 3.1.3. Berkeley Open Infrastructure for Network Computing

### 3.1.4. Folding@home

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





Folding@home jest jednym z największych systemów komputerowych na świecie. Jego szybkość szacuje się na około 14 petaFLOPSów na sekundę, więcej niż wszystkich projektów, które są przetwarzane na wywodzącej się z Berkeley platformie BOINC. W 2007 został wpisany do książki rekordów Guinnessa jako system rozproszonych obliczeń o największej mocy obliczeniowej. Stan na Czerwiec 2013 wskazuje prawie pół miliona aktywnych CPU oraz prawie 25 tysięcy aktywnych GPU.


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

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

Share This Link

Share: On your own timeline

Write something...



**Folding@home**  
<http://folding.stanford.edu/>  
What if you could share your unused computer power to help find a cure? In just 5 minutes you can join the world's largest distributed super computer and get us closer sooner. Watch the video.

1 of 1

Choose a Thumbnail

☐ No Thumbnail

Friends

Share Link

Cancel

## 4. Proponowane rozwiązanie

W rozdziale tym przedstawiono informacje .

### 4.1. Model narciarza i środowiska

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

### 4.2. Opis matematyczny modelu

$m$  - masa

$g$  - współczynnik grawitacji

$\alpha$  - kąt nachylenia powierzchni stoku do powierzchni ziemi

Siła grawitacji działająca na obiekt o masie  $m$ :

$$Q = mg \quad (4.1)$$

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

$$Q_a = mg \sin \alpha \quad (4.2)$$

jest to siła ściągająca narciarza w dół stoku.

Składowa siły grawitacji prostopadła do powierzchni stoku:

$$F_n = mg \cos \alpha \quad (4.3)$$

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

$$F_f = \mu F_n = \mu mg \cos \alpha \quad (4.4)$$

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

$$F_d = k_1 \dot{x} + k_2 \dot{x}^2 \quad (4.5)$$

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

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

gdzie  $B$ :

$$B = 4 \frac{m}{s} \quad (4.7)$$

Ograniczenie to zostało zaczerpnięte z pracy [1] (Aerodynamic-Drag) i opisane w rozdziale [1].

Współczynnik  $k_1$  ....

$$k_1 = ... \quad (4.8)$$

Współczynnik  $k_2$  został opisany w [1].

$$k_2 = \frac{1}{2} C \rho A \quad (4.9)$$

gdzie:

$C$  - współczynnik oporu

$\rho$  - gęstość powietrza

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

### 4.3. Numeryczne rozwiązanie problemu

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

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

Wyrażmy teraz przyspieszenie jako drugą pochodną położenia:

$$a = \ddot{x} \quad (4.11)$$

Podstawiając do równania:

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

Zatem po podzieleniu przez  $m$ :

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

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

$$v = \dot{x} \quad (4.14)$$

Otrzymujemy następujące równania:

$$\begin{cases} \dot{v} = g \sin \alpha - \mu g \cos \alpha - \frac{k_1}{m} v - \frac{k_2}{m} v^2 \\ v = \dot{x} \end{cases} \quad (4.15)$$

Pamiętamy przy tym, że:

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

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

### 4.3.1. Rozwiązanie w 3D

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

Założmy, że narciarz porusza się tylko po liniach prostych.

Rozpatrzmy teraz jak będzie wyglądał układ sił działających na narciarza:

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

$$g \sin \alpha * \sinus \quad (4.17)$$

gdzie  $\sinus$  to:

Zatem nasze równanie ruchu będzie wyglądało następująco:

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

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

$$F_n = g \cos \alpha \quad (4.19)$$

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

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

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

$$\begin{cases} v_x = \dot{x}_x \\ v_y = \dot{x}_y \\ \dot{x} = \sqrt{v_x^2 + v_y^2} \end{cases} \quad (4.21)$$



otrzymujemy:

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

$$\begin{cases} v_x = \dot{x}_x \\ v_y = \dot{x}_y \\ \dot{v}_x = g \sin \alpha - (\dot{x}_x^2 + \dot{x}_y^2)^{\frac{1}{2}} \kappa \dot{x}_y \operatorname{sgn}(\dot{\varphi}) - g \sin \alpha \frac{\dot{x}_y^2}{\dot{x}_x^2 + \dot{x}_y^2} - \frac{(F_f + F_d)}{m} \frac{\dot{x}_x}{(\dot{x}_x^2 + \dot{x}_y^2)^{\frac{1}{2}}} \\ \dot{v}_y = (\dot{x}_x^2 + \dot{x}_y^2)^{\frac{1}{2}} \kappa \dot{x}_x \operatorname{sgn}(\dot{\varphi}) + g \sin \alpha \frac{\dot{x}_x}{\dot{x}_x^2 + \dot{x}_y^2} - \frac{(F_f + F_d)}{m} \frac{\dot{x}_y}{(\dot{x}_x^2 + \dot{x}_y^2)^{\frac{1}{2}}} \end{cases}$$

## 4.4. Optymalizacja toru przejazdu

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

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

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

### 4.4.1. Algorytm ewolucyjny

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

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

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

Zastosowany algorytm opiera się na strategii  $(\mu + \lambda)$  opisanej w rozdziale 2. Jako początkową populację wybieramy losowe osobniki - poziome wartości punktów są ograniczone jedynie przez wartości poziome położenia dwóch najbliższych bramek. Jest to kierowane koniecznością zadbania o szybsze znalezienie rozwiązania - zbyt duże odległości można z góry odrzucić opierając się na doświadczeniach z rzeczywistej jazdy narciarza po slalomie. Wielkość populacji bazowej  $\mu$  jest jednym z parametrów programu, ale najczęściej wartość ta wynosi 30. Wartość parametru  $\lambda$  także jest parametrem, jednak w testach przeważnie użyto wielkości 100.

Szkielet algorytmu zgodny jest z zastosowaną strategią, po wylosowaniu z istniejącej populacji populacji tymczasowej o wielkości  $\lambda$ , dokonuje się na jej osobnikach operacji genetycznych, najpierw krzyżowania, a następnie mutacji na osobnikach otrzymanych z krzyżowania. Kolejnym krokiem jest ocenienie nowych osobników i wybranie spośród nich oraz populacji początkowej osobników o najlepszym przystosowaniu i to one stanowią nową populację bazową.

### Krzyżowanie

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

### Mutacja

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

### Warunek zakończenia

Warunek zakończenia algorytmu zawsze sprawia wiele problemów. Nie jest łatwo zdecydować na jakiej podstawie zatrzymywać jego działanie. Często korzysta się z informacji o rozrzucie przystosowania w populacji - obliczamy go na podstawie różnicy pomiędzy najlepszym i najgorszym osobnikiem. Jeśli rozrzut ten jest niewielki może oznaczać stagnację algorytmu. Niekoniecznie świadczy to o znalezieniu rozwiązania, ale w połączeniu z dodatkowymi mechanizmami może być skuteczną metodą na decyzję o zakończeniu optymalizacji.

W naszym rozwiązaniu bierzemy zatem również pod uwagę taki wskaźnik jak poprawa najlepszego obecnego rozwiązania. Jeśli przez określoną liczbę iteracji, najczęściej kilka lub kilkanaście najlepsze rozwiązanie nie poprawia się w ogóle, a populacja jest bardzo mało zróżnicowana to jest to znak, że znalezione rozwiązanie powinno być wystarczająco bliskie najlepszemu. Oczywiście sterując liczbą iteracji przez które sprawdzamy zmiany oraz wielkością rozrzutu populacji możemy znajdować lepsze lub gorsze rozwiązania kosztem wydłużenia lub skrócenia czasu działania algorytmu.

### 4.4.2. Hill climbing

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

ności nie możemy mieć nigdy.

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

## 4.5. Architektura systemu

## **5. Wyniki**

W rozdziale tym przedstawiono informacje .

### **5.1. Optymalizacja**

#### **5.1.1. Algorytm ewolucyjny**

#### **5.1.2. Lokalna optymalizacja**

### **5.2. Uczenie maszynowe**

### **5.3. Podsumowanie**

### **5.4. Optymalizacja toru przejazdu**

### **5.5. Architektura systemu**

## **6. Podsumowanie**

W rozdziale tym przedstawiono informacje .

### **6.1. Podrozdział**