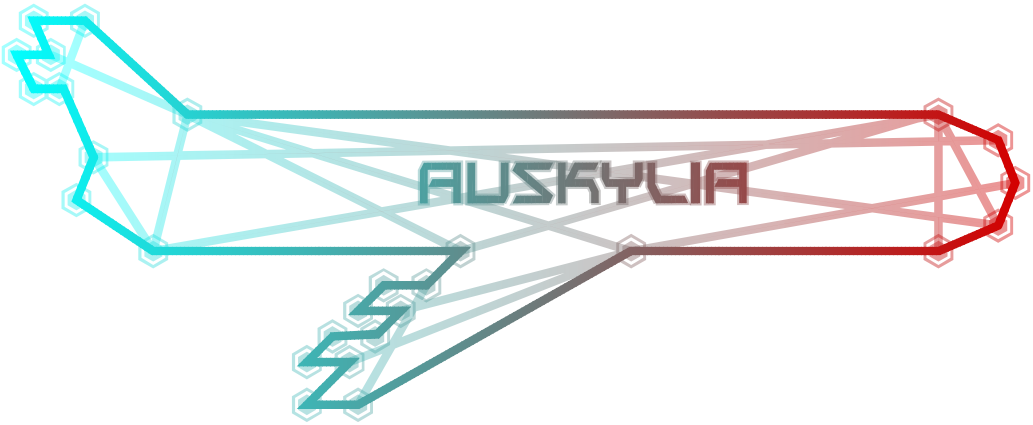


Opis problemu

Celem projektu było stworzenie programu obliczającego najlepszą drogę z Polski do Australii. Podstawową własnością programu miał być zastosowany algorytm ewolucyjny. Algorytm miał za zadanie brać pod uwagę nasze preferencje. Użytkownik ma możliwość zdefiniowania, jak bardzo zależy mu na:

- **Jak najmniejszej cenie za przelot.** Jest to sumaryczna ilość opłat za kolejne przeloty, celem jest jej zmniejszenie.
- **Bezpieczeństwie podróży.** Ważne jest, aby unikać połączeń obsługiwanych przez stare samoloty, albo przechodzących nad niebezpiecznymi terytoriami. Celujemy w maksymalizację tego współczynnika.
- **Wygodzie podróży.** Na to składają się takie rzeczy, jak wygoda siedzenia, serwowane jedzenie i udogodnienia.
- **Jak najmniejszym czasie przelotu.** Wiadomo, że ważne jest, aby nie spędzać całych dni na siedzeniu na fotelu w czasie lotu, nawet jeśli taki przelot jest bardziej opłacalny.

Opis architektury

Aby uniknąć problemów z wewnętrzną komunikacją aplikacji i aby zwiększyć czytelność kodu, zdecydowaliśmy się na zastosowanie architektury Model View Controller. Jest ona powszechnie używana i gwarantuje względną wolność od błędów. Dodatkowo wszyscy używaliśmy jej już wcześniej w projektach z innych przedmiotów i dobrze się sprawdziła.

Język C++ w standardzie C++11 jest bardzo łatwy w użyciu i pisaniu. Każdy z nas zna go bardzo dobrze. W dodatku jest wystarczająco szybki na aplikację z intensywnymi obliczeniami matematycznymi. Użycie języków interpretowanych, jak Python, lub działających na maszynie wirtualnej, jak Java spowodowałoby znacznie więcej błędów i trudniejszą implementację i kilkukrotnie dłuższe oczekiwanie na wynik, nawet jeśli program byłby bardziej przenośny.

Było wiele wątpliwości co do wybrania biblioteki interfejsu graficznego. Nie istnieje wspólna biblioteka z którą każdy z nas miałby wcześniej styczność. Wiele rozwiązań takich, jak QT wymaga także, aby główna pętla była po stronie interfejsu i jeden cały wątek odpowiadał za GUI i sterował aplikacją. W ten sposób dochodzi problem asynchronicznej komunikacji z resztą systemu. Obliczenia muszą działać w innym wątku równolegle, aby była możliwość sprawdzenia postępu obliczeń i zatrzymania wykonywania kalkulacji. Implementacja takiej komunikacji bez zjawiska wyścigów jest skomplikowana.

Rozwiązaniem tego problemu jest interfejs w HTML uruchamiany w przeglądarce. Każdy bowiem bardziej, lub mniej kojarzy tworzenie stron internetowych i wie, że jest to dość proste w użyciu. Jednocześnie umożliwia to obsługę programu z innego komputera, niż uruchomiona aplikacja. Rozwiązanie w postaci interfejsu HTML w przeglądarce jest stosowane w wielu aplikacjach. Przykładem są Metasploit – program do automatycznego przeprowadzania testów penetracyjnych, Freenet – zdecentralizowana sieć anonimizująca ma panel kontrolny przeglądarkowy (Tor już np. nie), lub CUPS – system zarządzający drukowaniem i obsługą sterowników drukarek w Unixach.

Kontroler działa, jak serwer. Używa gniazd, aby odbierać i wysyłać dane do przeglądarki. Kompiluje się w jedno razem z modelem odpowiedzialnym za obliczenia. Ma zapisaną listę dozwolonych plików, które pobiera z dysku i wysyła do przeglądarki, aby wyświetlić interfejs. Ten sposób uniemożliwia ataki typu „path traversal", czyli dodanie do nagłówka pakietu „../" przeskakującego o poziom w dół.

Plan projektu

Idea była, aby każdy z nas zajął się inną częścią architektury MVC. Tomasz Jakubczyk zajął się kontrolerem, Andrzej Roguski modelem, a Radosław Świątkiewicz widokiem.

Każdy również musiał mieć na oku miejsce, gdzie moduły łączą się ze sobą i zmodyfikować czyjś kod w razie czego. Projekt był tworzony przy systemie kontroli wersji Git, dzięki temu nie było problemów z działaniem kodu.

Każda część aplikacji rozwijała się w tym samym czasie z naciskiem na widok, aby można było zweryfikować działanie swojego fragmentu. Przez początkowy okres system działał i zwracał tymczasowe na sztywno wpisane dane.

Opis algorytmu

Model wczytuje listę skrótów IATA lotnisk po kolei z pliku, przyporządkowując kolejne numery. Dla każdego lotniska losowana jest jego pozycja na płaszczyźnie. Następnie algorytm generuje losowe połączenia między lotniskami, o losowych współczynnikach.

Obliczając odległość wierzchołków od siebie i mając daną prędkość samolotu na odcinku, można wyliczyć czas, jaki zajmie samolotowi przebycie trasy. Każde połączenie ma współczynnik komfortu i bezpieczeństwa w skali od 1 do 10. Również prędkość samolotu na tym połączeniu jest określona liczbą od 1 do 10. Koszt jest proporcjonalny do charakterystyki linii i jej długości. Jego wartość to średnia komfortu i szybkości razy dystans.

Dane do obliczeń to wierzchołki początkowy i końcowy, oraz parametry zadane przez użytkownika w postaci wag. Rozwiązaniem jest jedna ze ścieżek o początku i końcu podanym przez użytkownika.

Osobnikami algorytmu ewolucyjnego są ścieżki o określonym początku i końcu. Ich cechami, które krzyżujemy i mutujemy są kolejne wierzchołki.

Krzyżowanie odbywa się techniką jednopunktową. Jeśli mamy dwie ścieżki do krzyżowania, dzielimy każdą z nich w losowym miejscu i zamieniamy miejscami po jednej części. Pierwsze dziecko ma górną połowę żeńską, a dolną męską, natomiast drugie dziecko odwrotnie - górną męską, a dolną żeńską. Odpowiednikiem tego jest lotus podziału w algorytmach generycznych operujących na ciągach bitów.

Mutacja przebiega po wierzchołkach. Dla każdego owego jest pewna szansa na zmutowanie, może polegać ona na:

- ➔ Usunięciu wierzchołka i połączeniu sąsiadujących krawędzią.
- ➔ Dodaniu nowego losowego wierzchołka obok mutującego.
- ➔ Zamianie wierzchołka na inny, losowy wraz z połączeniami.

Funkcja przystosowania ocenia jakość ścieżki względem zadanych parametrów. Jest to suma uśrednionych wagowo parametrów każdej krawędzi. Najpierw wyliczamy średnią ważoną z parametrów krawędzi, gdzie wagi są liczbami podanymi w widoku. Potem dodajemy wszystkie średnie z krawędzi do siebie. Działa to w ten sposób:

- ➔ Obliczenie dystansu między początkiem ścieżki, a końcem w linii prostej.
- ➔ Pod a podstawiamy dystans podzielony przez sumaryczny czas podróży.
- ➔ Pod b podstawiamy sumaryczny koszt podróży podzielony przez dystans.
- ➔ Pod c podstawiamy średni komfort podróży.
- ➔ Pod d podstawiamy średnie bezpieczeństwo podróży.

- ➔ Przemnożenie a, b, c i d przez współczynniki podane przez użytkownika.
- ➔ Zsumowanie przemnożonych wartości daje ocenę ścieżki.

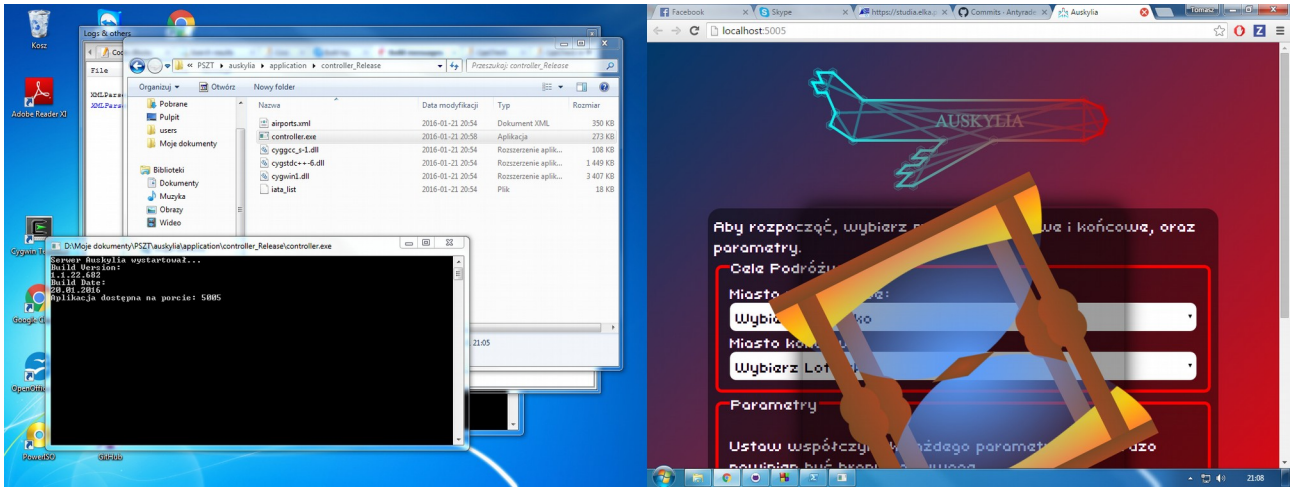
Po urodzeniu potomków i dołączeniu ich do zbioru, jest on sortowany względem funkcji przystosowania. Gorsza połowa jest odrzucana. Następne krzyżowanie odbywa się kolejno parami (0,1), (2,3), (4,5)...

Instrukcja uruchomienia aplikacji

Na Linuxie wystarczy wejść w katalog **application** i uruchomić program make. Następnie uruchomić plik wykonywalny **auskylia** Program przyjmuje jeden argument, to jest numer portu na którym ma zostać uruchomiony serwer. Domyślnie jest to 5005.

Na Windowsie należy uruchomić plik EXE w folderze controller_Release\controller.exe. Powinno wyświetlić się okno wiersza polecenia. Można też uruchomić program bezpośrednio z wiersza polecenia, a jako argument opcjonalnie wpisać numer portu.

Wystarczy teraz wejść przeglądarką na stronę http://localhost:5005, lub podobną, jeśli zmieniono numer portu, mamy tam dostęp do interfejsu użytkownika.



Wygląd i obsługa aplikacji

Zarys interfejsu

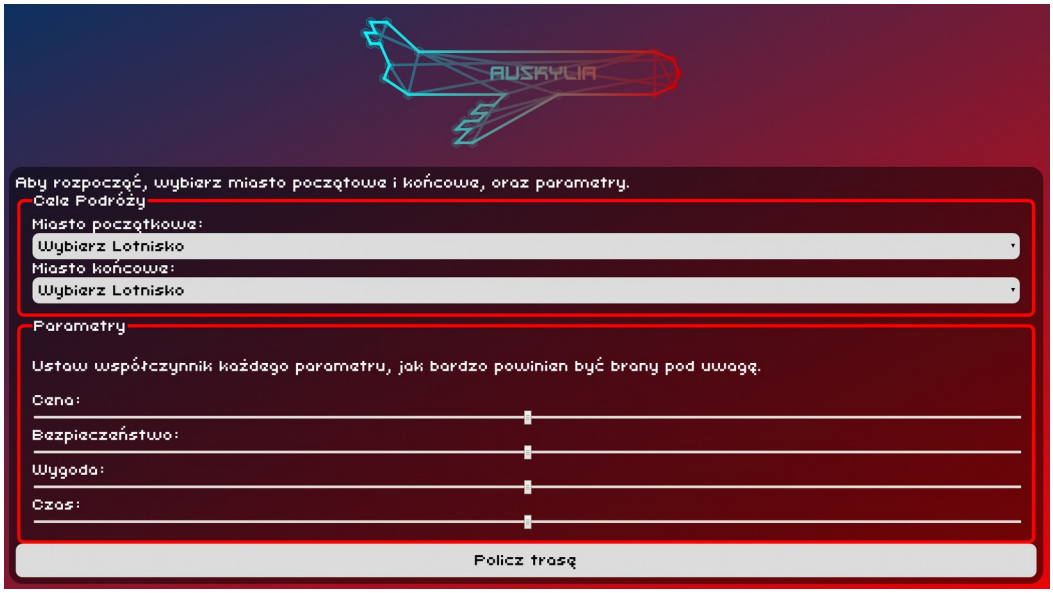
Główny interfejs jest stroną internetową. Za jego pomocą następuje komunikacja z resztą aplikacji, działającą jak mały serwer WWW.

Wymagania przeglądarki

Interfejs wymaga od przeglądarki obsługi najnowszych standardów HTML5 i CSS3. Najnowsze wersje Chrome/Chromium, Firefoxa i Opery bez problemu poradzą sobie z zadaniem. Jeśli przeglądarka nie posiada niektórych funkcjonalności, wpłynie to jedynie na wygląd, a nie na działanie programu. Ze względu na użytą wersję jQuery, aplikacja nie zadziała na Internet Explorer 8 i starszych.

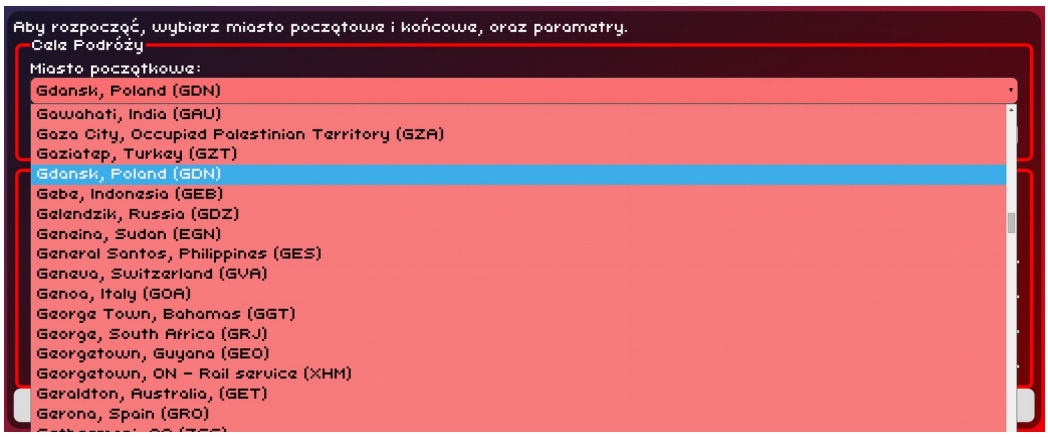
Uruchomienie

Domyślnie aplikacja działa na porcie 5005, aby uzyskać dostęp do interfejsu, należy wpisać http://localhost:5005 w pasek. Po uruchomieniu i wczytaniu, strona zacznie ściągać listę wszystkich lotnisk na świecie. Trwa to zazwyczaj kilka sekund. W tym czasie wyświetlane jest okno ładowania, a interfejs jest zablokowany. Po wczytaniu lotnisk, możemy podać dane do obliczeń. Interfejs wygląda wtedy następująco.



Wprowadzanie danych

Za pomocą dwóch selektorów możemy wybrać lotniska początkowe i końcowe. Standardowo przy tego typu kontrolkach, wpisanie litery na klawiaturze odsyła nas do pierwszego miasta na tę literę.



Po wyborze miast, możemy użyć suwaków poniżej do wprowadzenia parametrów znajdowania drogi. Są to:

- **Cena za bilet.** Powinna być jak najmniejsza. Przesuwając suwak w lewo zmniejszamy wpływ tego parametru na obliczenia – nie obchodzi nas, czy zapłacimy dużo, czy mało. Przesuwając w prawo zwiększamy wpływ parametru, chodzi nam o to, żeby cena przelotów była jak najmniejsza kosztem innych własności.
- **Bezpieczeństwo podróży.** Bywają połączenia nad niebezpiecznymi rejonami i starymi maszynami. Suwak z lewej strony zmniejszy nam wpływ tego parametru, suwak z prawej wybierze nam najnowsze i najbezpieczniejsze połączenia.
- **Wygoda.** Niektórzy nie obejdą się bez pierwszej klasy i obiadu posypanego płatkami złota. Przesuwając trzeci suwak w prawo wybierzemy tylko najwygodniejsze połączenia rejsowe. Suwak z lewej zagwarantuje nam miejsce w luku bagażowym.
- **Czas podróży.** Są historie ludzi spędzających na lotnisku całe dnie. Jeśli chcemy do celu dolecieć jak najszybciej, warto ustawić ten ostatni suwak z prawej strony. Ustawienie go z lewej zmniejszy nam wpływ czasu na kalkulację i może zwiększyć ogólny czas podróży.

Domyślnie wartości są ustawione po równo na 50%.

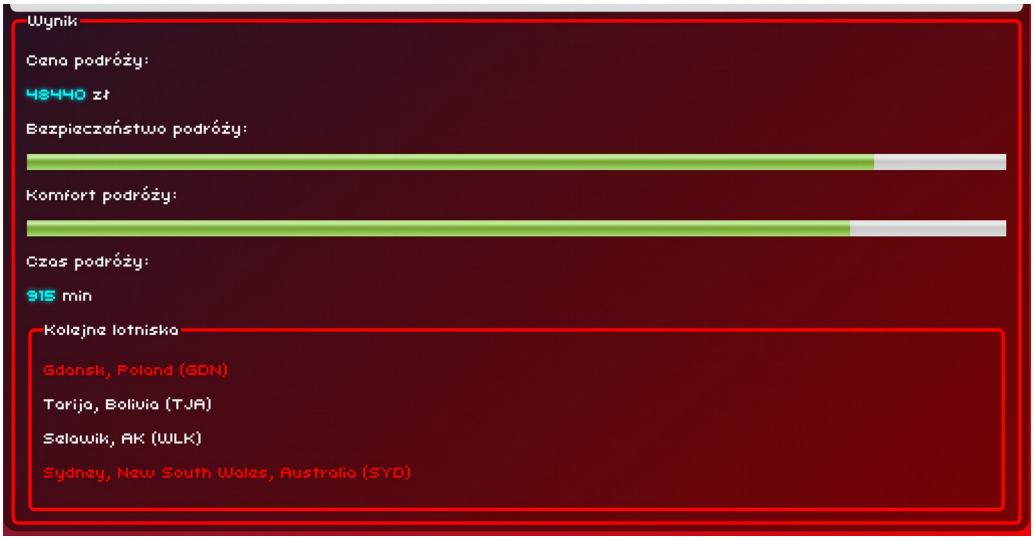
Jeśli jesteśmy zadowoleni z danych, możemy wysłać je do aplikacji i rozpocząć obliczanie ścieżki. Naciśnięcie ostatniego, dużego przycisku zablokuje interfejs na czas wysyłania danych i dostaniemy możliwość monitorowania postępu.

Oczekiwanie na wynik

W czasie oczekiwania na wynik, pokaże się pasek postępu. Zazwyczaj aplikacja działa na tyle szybko, że ów pasek od razu wypełni się do 100%. Poniżej paska znajduje się opcja przerwania obliczeń. Po przerwaniu dostaniemy błąd serwera o celowym niedokończeniu obliczania wyniku. Po uzyskaniu 100% pokaże się wynik.

Wyświetlenie wyniku

Program po obliczeniu wyświetli wynik pod przyciskiem i schowa okno postępu. Aby przeliczyć nową trasę, wystarczy ponownie ustawić wartości i załączyć obliczenia. Ono wyniku składa się z 5 pól. Na niego składają się obliczona cena przelotu, bezpieczeństwo, komfort, czas i lista lotnisk.

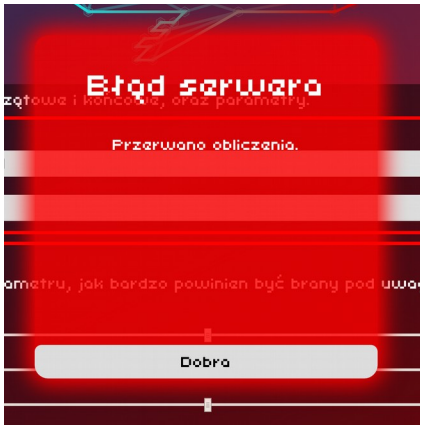


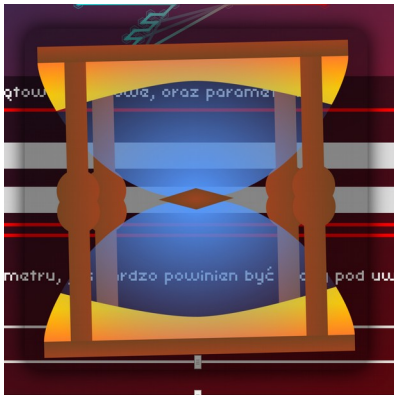
- ➔ Cena podróży to sumaryczna wartość przelotów na wszystkich krawędziach. Tyle musimy przygotować funduszy na całą podróż.
- ➔ Pasek bezpieczeństwa to średnia z bezpieczeństw każdego z przelotów. Jeśli jeden przelot jest bardzo bezpieczny, a drugi mało, to wynik wskaże wartość pomiędzy.
- ➔ Komfort również jest średnią z każdej krawędzi. Komfortowe samoloty podwyższają wartość, a ekonomiczne obniżają.
- ➔ Czas podróży jest sumą wszystkich czasów na ścieżce, bez brania pod uwagę oczekiwania na lotniskach.

Następnie jest lista kolejnych lotnisk przez które przechodzi nasz lot. Pierwsze i ostatnie są zaznaczone kolorem.

Okna i stany programu

Program w czasie wysyłania i odbierania danych wyświetla okno i animacje klepsydry. Jeśli serwer zwróci błąd, lub nie uda się wysłać/odebrać danych, wyświetli się powiadomienie o błędzie.





Wszystkie grafiki są w formacie wektorowym i zostały stworzone na potrzeby tego projektu. Animacje pojawiania się i znikania okien dostarczyło jQuery, a animacja obracania klepsydry jest natywna dla CSS3. Gradient tła również jest osiągnięty przez CSS, a nie przez grafikę.

Wnioski i podsumowanie

Najważniejszy wniosek, jaki wyciągnęliśmy z projektu, jest taki, że gdybyśmy mieli robić go jeszcze raz, to zrobilibyśmy go inaczej. Przede wszystkim należy zmienić sposób modelowania ścieżek. Zamiast ciągu kolejnych wierzchołków ścieżki, lepsze byłoby zastosowanie zapewne kodowania bitowego do zapisania kolejności oraz obecności wierzchołków w ścieżce. Jednym słowem algorytm całkowicie generyczny. Rozwiązało by to wiele problemów, takich jak konieczność sprawdzania poprawności ścieżki w celu uniknięcia cykli lub zmienna liczba cech osobników.

Pierwotna koncepcja zakładała zastosowanie krzyżowania równomiernego (dla odpowiednio znormalizowanych ścieżek o różnych długościach - z losowo wstawionymi genami "pustymi").

Pomysł ten, poza problematyczną implementacją, jak łatwo przewidzieć nie sprawdził się zbyt dobrze. Jako, że dopasowanie do optimum każdej cechy zależało nie tylko od wierzchołka na danej pozycji, ale też i tych na sąsiednich pozycjach, krzyżowanie dwóch dobrych ścieżek dawało rezultaty zupełnie losowe.

Znaczącą poprawę wprowadziła zmiana krzyżowania na jednopunktowe (częściowo rozwiązując przy okazji problem ścieżek różnej długości). Dzięki temu dwie dobre ścieżki generują potomka, u którego złą może być tylko jedna krawędź. Pozwoliło to też prosto generować potomków o różnych długościach.

Innym problemem okazały się powtarzające się ścieżki oraz bardzo krótkie ścieżki. Bardzo szybko prowadziły one do zapełnienia populacji jednakowymi osobnikami, zabijając dalsze działanie algorytmu. Zastąpienie potomków bardzo krótkich ścieżek oraz powtarzających się osobników nowo losowanymi ścieżkami rozwiązało problem.

Warty uwagi jest także problem generowania grafu. Parametry połączeń generowane zupełnie losowo prowadziły do powstania grafu, gdzie - uśredniając - wszystkie punkty były od siebie mniej więcej równoodległe, co prowadziło do matematycznej niemożliwości. W efekcie uzyskiwane populacje zawsze składały się z bardzo krótkich ścieżek.

Sytuację poprawiło losowanie pozycji punktów na pomocniczej płaszczyźnie - dzięki temu jeśli A jest blisko B, a daleko C, B również będzie daleko od C, co oznacza, że ścieżka złożona z punktów leżących blisko odcinka łączącego punkt początkowy z końcowym nie będzie wiele dłuższa od tegoż odcinka.

Inną kwestią godną wspomnienia jest funkcja przystosowania. Bardzo wyraźnie można było zaobserwować wpływ odpowiedniego dobrania funkcji na jakość algorytmu.

W kwestii widoku okazało się, że zabawa w cukierkowe grafiki zamęczy na śmierć każde urządzenie mobilne i starsze komputery. Jednak coraz więcej stron ociekających w animacje pojawia się w internecie. Może następnym razem należałoby do tego podejść bardziej po inżyniersku? W idealnym świecie wynik

byłby wypisywany na standardowe wyjście, a potem użytkownik mógłby robić z nim co chce. Za pomocą ANSI Escape Codes można ustawiać kursor w dowolnej pozycji i „rysować" paski ładowania itp. Przecież tak działają prawie wszystkie programy Unixowe. Dla wymagających jest biblioteka ncurses wyświetlająca pseudointerfejs w postaci grafiki z różnych kolorowych znaków na terminalu i sterowalna strzałkami. Jeśli terminal znajduje się w oknie, to potrafi nawet rejestrować kliknięcia myszą. Programy prezentujące w ten sposób dane nie ustępują w funkcjonalności okienkowym. Istnieje nawet przeglądarka internetowa tego typu. Dla problemu naszej aplikacji takie rozwiązanie było by całkowicie wystarczające.