Politechnika Łódzka

Instytut Informatyki

PRACA DYPLOMOWA MAGISTERSKA

Optymalizacja struktury sieci drogowej

Optimization of structures of road networks

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej
Promotor: dr hab. inż. Aneta Poniszewska-Marańda
Kopromotor: mgr inż. Łukasz Chomątek
Dyplomant: inż. Michał Siatkowski
Nr albumu: 186865
Kierunek: Informatyka
Specjalność: Sztuczna Inteligencja i Inżynieria Oprogramowania

Łódź 20.01.2015

Instytut Informatyki

90-924 Łódź, ul. Wólczańska 215, budynek B9

tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: office@ics.p.lodz.pl

Spis treści

Rozdział 1

Wstęp.

1.1 Problematyka i zakres pracy.

Niniejsza praca obejmuje zagadnienia z zakresu inżynierii oprogramowania i sztucznej inteligencji. Głównym jej celem jest stworzenie aplikacji optymalizującej strukturę sieci drogowej.

Problemy komunikacji w dzisiejszych miastach są wszystkim znane. Zatory drogowe i korki w godzinach szczytu są chlebem powszednim. Pomimo wielu prób i sposobów, wciąż nie istnieje metoda jednoznacznie rozwiązująca tę kwestię. Bezspornie, dotyczy to wszystkich miast na świecie. Z teoretycznego punktu widzenia, jedynym rozwiązaniem jest komunikacja publiczna. Oczywistym jest jednak, że nigdy nie doprowadzimy do sytuacji, gdy wszyscy mieszkańcy zrezygnują ze swoich pojazdów. Dodatkowo, wiele osób i usług wymaga oddzielnej formy transportu. W obliczu tych faktów miasta decydują się na rozwój swojej infrastruktury drogowej. Budowa nowych tras oraz poszerzanie starych przynosi nadzieję mniejszych zatorów, a co za tym idzie, szybszego przejazdu do celu. Niestety, historia pokazuje, że takie inwestycje nie zawsze przynoszą oczekiwane korzyści.

Teorii próbujących wytłumaczyć te zjawiska, jak również dowodów, które je popierają lub obalają jest wiele. Jedną z najpopularniejszych oraz taką która została wykorzystana w niektórych miastach na świecie jest paradoks Braessa[20]. Jest to twierdzenie matematyczne orzekające, że w pewnym modelu ruchu drogowego, czasy podróży pojazdów mogą ulec wydłużeniu po dodaniu do sieci drogowej nowego połączenia. Ma ono również zastosowanie w przypadku sieci komputerowych oraz istnieją jego analogie dla doświadczeń fizycznych.

Celem pracy jest opracowanie metody, która dla danej struktury sieci drogowej, zmodyfikuje ją wykorzystując prawidłowość z powyższego paradoksu. W efekcie poprzez zamknięcie wybranych ulic w danej sieci drogowej, średni czas podróży powinien ulec skróceniu.

1.2 Metoda badawcza i cel pracy.

Studia literaturowe.

Badania rozpoczęliśmy od poszukiwania źródeł traktujących o opisywanym problemie. Paradoks Braessa został sformułowany w 1970 roku i był od tego czasu wykorzystywany przy planowaniu przestrzeni i infrastruktury wielu miast, np:

- Korea, Seul, likwidacja m.in. estakad Cheonggyecheon,
- Niemcy, Stuttgart, likwidacja dróg zbudowanych w latach 60,
- USA, Nowy Jork, czasowe zamknięcie ulicy 42,
- USA, Winnipeg.[21]

Propozycja rozwiązania problemu.

Oczywistym rozwiązaniem problemu komunikacji mogłoby być stworzenie idealnej sieci odpowiadającej potrzebom danego miasta. Rozbudowa lub modyfikacja tej infrastruktury jest jednak kosztowna i czasochłonna. Z tego powodu postanowiliśmy sprawdzić rozwiązanie zaproponowane przez Braessa. Ponieważ istnieją prace negujące lub podważające paradoks[2] , zdecydowaliśmy by przy potwierdzaniu wyniku optymalizacji nie kierować się wyłącznie założeniami zawartymi w twierdzeniu.

Opis zastosowania algorytmów genetycznych.

Ponieważ nie znaleźliśmy żadnych przesłanek wykazujących jednoznaczną ocenę co do słuszności zamknięcia danej ulicy, zdecydowaliśmy się na losowe przeszukiwanie przestrzeni rozwiązań. Jednym z rozwiązań w przypadku takich poszukiwań są algorytmy genetyczne, które zostały wykorzystane w pracy.

Przedstawienie oceny optymalizacji.

Paradoks Braessa zakłada dość oczywiste potwierdzenie swojej wiarygodności. Dlatego, w celu potwierdzenia jego słuszności, zdecydowaliśmy się na zastosowanie zewnętrznego systemu oceny. Taką rolę spełniają systemy symulacji. System, który wybraliśmy działa zupełnie oddzielnie od metody twierdzenia, symulując rzeczywisty ruch na danej sieci drogowej. Wynik symulacji jest jednoznaczną wartością liczbową, przedstawiającą średni czas przejazdów wszystkich agentów biorących udział w danym scenariuszu. Zakładając stały zestaw agentów dla zmieniających się w wyżej opisany sposób sieci, dążymy oczywiście do minimalizacji średniego czasu przejazdu.

Ocena możliwości wdrożenia proponowanych rozwiązań.

Paradoks Braessa nie jest jedynym twierdzeniem traktującym o problemach komunikacyjnych miast. Wiele teorii jest opartych głownie na socjologicznych lub psychologicznych założeniach[1]. Są jednak niemniej ważne. Biorąc pod uwagę złożoność problemu, wynik otrzymany podczas eksperymentu nie może być dowodem, ani decydującym głosem w decyzjach dotyczących ustalaniu rzeczywistej sieci drogowej miasta.

1.3 Przegląd literatury w dziedzinie.

By przybliżyć temat problemów komunikacyjnych i rozwiązania zaproponowanego przez Braessa polecamy pracę magisterską *Leslie Arthura Keith Bloy*[1]. W pracy zostały opisane również inne twierdzenia dotyczące ruchu drogowego. Publikacja *Reducing the Effects of the Braess Paradox with Information Manipulation*[4] jest bardzo dobrym uzupełnieniem tematu o interesującą nas kwestię symulacji wieloagentowych. Prezentuje ona różnice w wynikach dla przypadku losowego wyboru drogi przez agentów oraz tej wybranej przy użyciu inteligencji kolektywnej[2].

Zbiorowa praca napisana na potrzeby międzynarodowej konferencji dotyczącej technologii symulacji[4] jest pozycją, która opisuje podobny do podejmowanego przez nas problem. Mianowicie skupia się na modyfikacjach drogi poprzez zmiany dostępności jej składowych.

[1] np. paradoks Downsa Thomsona[22] czy prawo Lewisa Mogridge'a[23]

[2] ang. Collective Intelligence (COIN)

W zupełnie oddzielnej tematyce, algorytmów genetycznych, polecamy pozycję, która jest wstępem do tej tematyki[6]. Pozycja zawiera bogaty zestaw problemów przykładowych wraz z opisem ich rozwiązania. Ponadto dogłębnie opisuje ona każdy aspekt działania zastosowanych algorytmów.

1.4 Układ pracy.

Tematem pracy jest optymalizacja struktury sieci drogowej, zaś za główny cel przyjęto stworzenie rozwiązania, które wykorzystując algorytmy genetyczne odnajdzie najlepsze rozwiązanie.

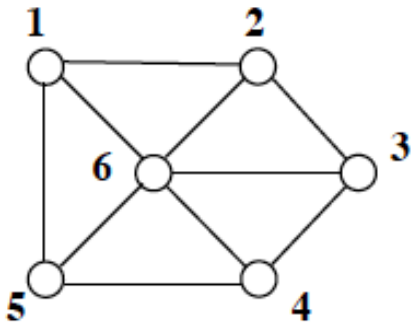
Rozdział 1 zawiera wstęp i cele pracy. W rozdziale 2 przybliżamy istotę optymalizacji struktur sieci drogowych oraz opisujemy teoretycznie, wykorzystane przeze nas rozwiązania. W rozdziale 3 wymieniamy technologie, w których wykonaliśmy aplikację oraz przybliżamy zewnętrzne biblioteki, o które oparliśmy swoje rozwiązanie. Głównym rozdziałem pracy jest rozdział 4, w którym przedstawiamy wykonaną aplikację oraz wyniki optymalizacji przykładowej sieci. W rozdziale podsumowującym 5, zawarliśmy wnioski oraz możliwości rozwoju pracy. Po rozdziale 5, podsumowującym, załączamy spis rysunków, tabel, listingów i bibliografie, w tej kolejności.

Rozdział 2

Optymalizacja struktury sieci drogowej.

2.1 Podstawowe definicje.

Definicja 1. ***Grafem (nieskierowanym)*** *nazywamy parę zbiorów* (V, E)*. Elementy zbioru* V *nazywają się wierzchołkami, natomiast elementy zbioru* E *nazywają się krawędziami. Każda krawędź jest parą wierzchołków, tzn.* $E \subseteq u, v : u, v \in V$*[9].*

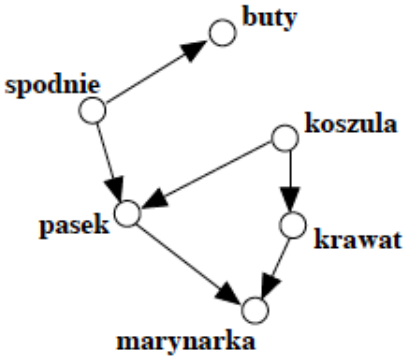


Rysunek 2.1: Przykładowy graf nieskierowany.

Przykładowy graf nieskierowany może zostać opisany przez zbiory:

$V = \{1, 2, 3, 4, 5, 6\}$ $E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{6, 5\}, \{6, 1\}, \{2, 6\}, \{3, 6\}, \{4, 6\}\}$

Definicja 2. ***Grafem skierowanym*** *nazywamy taki graf, w którym każda krawędź ma zdefiniowany początek i koniec, tzn. żeby pary były uporządkowane. Wtedy* $E \subseteq V \times V = u, v : u, v \in V$*. Krawędź* (u, v) *najłatwiej wyobrazić sobie jako strzałkę od u do v, dlatego często będziemy oznaczać ją przez* $u \rightarrow v$*[9].*

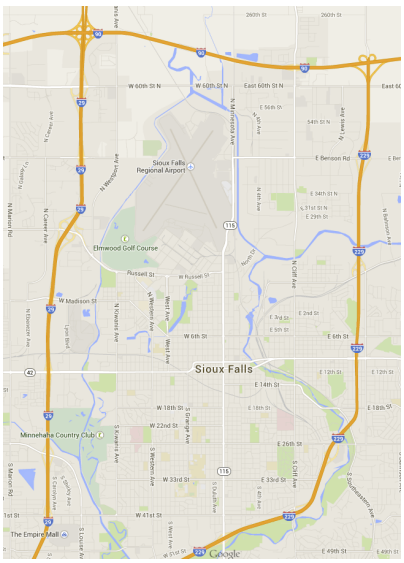


Rysunek 2.2: Przykładowy graf skierowany.

Przykładowy graf skierowany może zostać opisany przez zbiory:
$V = \{spodnie, buty, pasek, koszula, krawat, marynarka\}$
$E = \{\{spodnie \rightarrow buty\}, \{spodnie \rightarrow pasek\}, \{pasek \rightarrow koszula\}, \{koszula \rightarrow krawat\}, \{pasek \rightarrow marynarka\}, \{krawat \rightarrow marynarka\}\}$

2.2 Sieć drogowa w postaci grafu

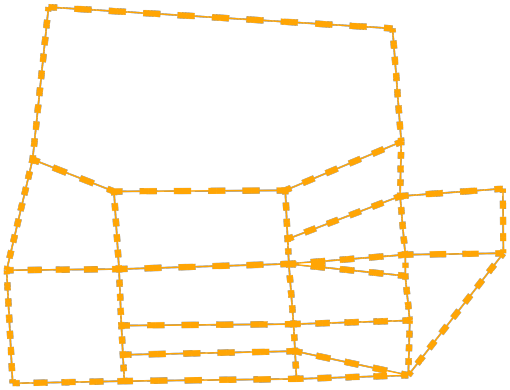
W przypadku sieci drogowej mamy oczywiście do czynienia z abstrakcyjną strukturą sieci. Przez sieć drogową rozumiemy bowiem układ dróg lub ulic np. w mieście. Podczas swoich badań posługuję się zawsze pewnym fragmentem większej sieci. Najlepiej te dane zobrazuje poniższy przykład.



Rysunek 2.3: Fragment sieci drogowej miasta Sioux Falls, Południowa Dakota.

Posługując się powyższymi definicjami, tworząc graf z pewnej sieci drogowej, przyjmujemy, że zbiorem V, wierzchołków są skrzyżowania, natomiast zbiór krawędzi E odnosi się do ulic pomiędzy tymi skrzyżowaniami. W pracy posługujemy się zawsze grafem skierowanym. W związku z tym, w przypadku ulic dwukierunkowych tworzone są pary krawędzi z odpowiednimi kierunkami, nawet jeśli ulice nie są rozłączne w rzeczywistości.

W celu potwierdzenia wiarygodności powyższego przykładu prezentujemy graf nałożony na mapę miasta[8].



Rysunek 2.4: Siec drogowa miasta Sioux Falls w postaci grafu.

Rysunek 2.5: Graf z dopasowaną geometrią.

2.3 Paradoks Braessa.

Jak już wcześniej wspomnieliśmy, jest to twierdzenie matematyczne orzekające, że w pewnym modelu ruchu drogowego czasy podróży pojazdów mogą ulec wydłużeniu po dodaniu do sieci drogowej nowego połączenia. Autorem twierdzenia jest niemiecki matematyk Dietrich Braess[20]. Paradoks działa w oparciu o model ruchu drogowego, który ma następujące cechy:

1. Sieć drogowa składa się ze skończenie wielu węzłów i łączących je odcinków dróg
2. Po sieci porusza się skończenie wiele pojazdów, każdy z nich ma wyznaczony węzeł startowy i węzeł docelowy

3. Odcinki dróg mają przypisane sobie czasy przejazdu, przy czym czasy te mogą zależeć od liczby pokonujących dany odcinek pojazdów.

4. Układ sieci drogowej i czasy przejazdu poszczególnych odcinków są znane pojazdom

5. Celem pojazdów jest przejazd przez sieć z węzłów początkowych do docelowych po trasie złożonej z odcinków drogowych tak, by zminimalizować łączny czas ich pokonania

6. Decyzje o wyborze tras pojazdy podejmują indywidualnie i niezależnie od siebie

Paradoks przedstawiamy w oparciu o przykład z oryginalnego artykułu Dietricha Braessa[5].

2.3.1 Przykładowy, wyjściowy układ drogowy.

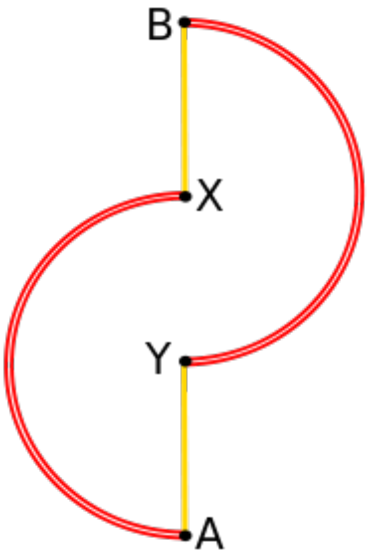
Sieć drogowa i auta

Przykład sytuacji, w której ujawnia się paradoks Braessa jest skonstruowany z czterech miast A, B, X i Y. Są one połączone odcinkami drogowymi jak na rysunku i z następującymi czasami przejazdu, przy czym p oznacza gęstość ruchu w tysiącach aut.

Autostrady:
AX, $t_{AX}(p) = 50 + p$ min
YB, $t_{YB}(p) = 50 + p$ min

Drogi lokalne:
AY, $t_{AY}(p) = 10p$ min
XB, $t_{XB}(p) = 10p$ min

Aut jest 6000 i wszystkie mają za zadanie przejechać trasę z A do B.



Rysunek 2.6: Wyjściowy układ drogowy

Analiza równowagi Nasha

Każde auto musi zdecydować się na wybór trasy albo AXB albo AYB.

Równowaga Nasha to taka sytuacja, w której każdy z samochodów spowoduje wydłużenie swojego czasu jazdy, zmieniając decyzję co do wyboru trasy przy niezmienionych decyzjach pozostałych aut.

Jeśli p i q to liczby aut w tysiącach pokonujących odpowiednio trasy AXB i AYB, otrzymujmy równania:

$$\begin{aligned} p + q &= 6 \\ t_{AX}(p) + t_{XB}(p) &= t_{AY}(q) + t_{YB}(q) \\ 50 + p + 10p &= 10q + 50 + q \end{aligned}$$

rozwiązaniem jest $p = q = 3$. Przy tej gęstości ruchu pokonanie obu dostępnych tras zabiera $50 + 3 + 30 = 83$ minuty.

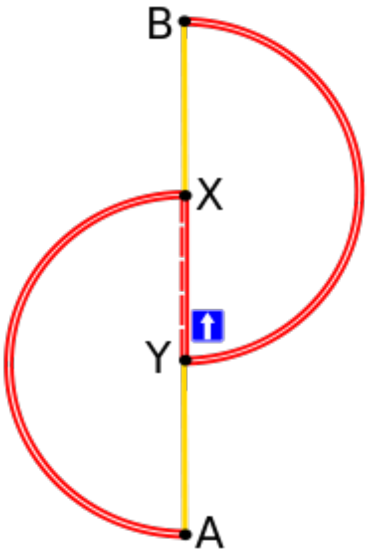
2.3.2 Przykładowy, uzupełniony układ drogowy.

Sieć drogowa i auta

Do wyjściowego układu drogowego dodana zostaje autostrada:

Autostrady:
YX, $t_{YX}(p) = 10 + p$ min

Aut jest nadal 6000 i wszystkie mają za zadanie przejechać trasę z A do B.



Rysunek 2.7: Uzupełniony układ drogowy

Analiza równowagi Nasha

Jeśli p, q i r to liczby aut w tysiącach pokonujących odpowiednio trasy AXB, AYB i $AYXB$, otrzymujmy równania:

$$p + q + r = 6$$
$$t_{AX}(p) + t_{XB}(p + r) = t_{AY}(q + r) + t_{YB}(q) = t_{AY}(q + r) + t_{YX}(r) + t_{XB}(p + r)$$
$$50 + p + 10(p + r) = 10(q + r) + 50 + q = 10(q + r) + 10 + r + 10(p + r)$$

rozwiązaniem jest $p = q = r = 2$. Czas przejazdu każdej z tych dróg wynosi wówczas $50 + 2 + 10(2 + 2) = 92$ minuty.

2.3.3 Wyjaśnienie intuicyjne.

Wąskim gardłem systemu są drogi lokalne, na których czas przejazdu bardzo szybko wzrasta wraz z intensywnością ruchu. Po pojawieniu się dodatkowej drogi dostępna staje się nowa trasa, prowadząca oprócz nowego skrótu YX tylko drogami lokalnymi.

Z perspektywy całości systemu nowy odcinek drogowy odciąża ruch na autostradach, gdzie jest to mało odczuwalne, a w zamian jeszcze bardziej zagęszcza ruch na drogach lokalnych, powodując wydłużenie czasu podróży.

2.4 Graf skierowany silnie spójny.

Na potrzeby symulatora, sieć drogowa przedstawiona w postaci grafu skierowanego musi spełniać warunek silnej spójności.

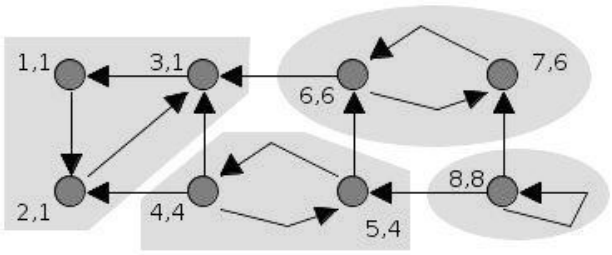
Definicja 3. ***Grafem skierowanym silnie spójnym*** *nazywamy graf skierowany w którym jest możliwe dotarcie do każdego wierzchołka zaczynając z dowolnego innego poprzez dowolną ilość krawędzi. Wszystkie wierzchołki w grafie skierowanym silnie spójnym muszą zatem posiadać przynajmniej jedną krawędź wchodzącą i wychodzącą[24].*

W sprawdzaniu grafu pod względem spójności, wykorzystujemy algorytm Tarjana do znajdowania składowych silnie spójnych.

Definicja 4. *Podstawowym założeniem* ***algorytmu Tarjana*** *jest przeszukiwanie grafu w głąb zaczynając od dowolnego wierzchołka wybranego w sposób arbitralny. Tak jak*

w przypadku klasycznego przeszukiwania w głąb, każdy sąsiadujący wierzchołek po odwiedzeniu zostaje oznaczony i algorytm nigdy ponownie go nie odwiedza. Dzięki temu, tworzymy kolekcję przeszukanych drzew, która jest drzewem rozpinającym grafu. Składowe silnie spójne są następnie odnajdowane jako poddrzewa, a korzenie tych poddrzew są nazywane korzeniami składowych silnie spójnych. Każdy wierzchołek grafu może być wybrany na korzeń składowej silnie spójnej jeśli zostanie wybrany jako pierwszy wierzchołek podczas przeszukiwania w głąb.

Definicja 5. ***Składowa silnie spójna*** *(ang. strongly connected component) jest maksymalnym pod grafem, w którym istnieją ścieżki pomiędzy każdymi dwoma wierzchołkami. Jeśli pod graf ten obejmuje wszystkie wierzchołki grafu, to mówimy, że dany graf skierowany jest silnie spójny (ang. strongly connected digraph). W grafach nieskierowanych każdy graf spójny jest również silnie spójny.*



Rysunek 2.8: Przykładowy graf z zaznaczonymi składowymi silnie spójnymi.

W efekcie, zawsze przed rozpoczęciem symulacji sprawdzamy, czy graf jest grafem skierowanym silnie spójnym lub dokładniej mówiąc, czy posiada tylko jedną składową silnie spójną.

2.5 Klasyczny algorytm genetyczny.

Idea algorytmu genetycznego została zaczerpnięta z nauk przyrodniczych opisujących zjawiska doboru naturalnego i dziedziczenia. Mechanizmy te polegają na przetrwaniu osobników najlepiej dostosowanych w danym środowisku, podczas gdy osobniki gorzej przystosowane sa eliminowane. Z kolei te osobniki, które przetrwają - przekazują informacje genetyczną swoim potomkom. Krzyżowanie informacji genetycznej otrzymanej od „rodziców” prowadzi do sytuacji, w której kolejne pokolenia sa przeciętnie coraz lepiej dostosowane do warunków środowiska; mamy tu więc do czynienia ze swoistym procesem optymalizacji.

2.5.1 Podstawowe pojecia algorytmów genetycznych

Definicja 6. ***Populacją*** *nazywamy zbiór osobników o określonej liczebności.*

Definicja 7. ***Osobnikami*** *populacji w algorytmach genetycznych są zakodowane w postaci chromosomów zbiory parametrów zadania, czyli rozwiązania, określone też jako punkty przestrzeni poszukiwań. Osobniki czasami nazywa się organizmami.*

Definicja 8. ***Chromosomy*** *to inaczej łańcuchy lub ciągi kodowe, to uporządkowane ciągi genów.*

Definicja 9. ***Gen*** *– nazywany też cechą, znakiem, detektorem – stanowi pojedynczy element genotypu, w szczególności chromosomu. Genotyp, czyli struktura, to zespół chromosomów danego osobnika. Zatem osobnikami populacji mogą być genotypy albo pojedyncze chromosomy.*

Definicja 10. ***Fenotyp*** *jest zestawem wartości odpowiadających danemu genotypowi, czyli zdekodowana struktura, a wiec zbiorem parametrów zadania (rozwiązaniem, punkt przestrzeni poszukiwań).*

Definicja 11. ***Allel*** *to wartość danego genu, określona jako wartość cechy lub wariant cechy.*

Definicja 12. ***Locus*** *to pozycja - wskazuje miejsce położenia danego genu w łańcuchu, czyli chromosomie.*

Definicja 13. ***Funkcja przystosowania nazywana tez funkcją dopasowania lub funkcja oceny.*** *Stanowi ona miarę przystosowania (dopasowania) danego osobnika w populacji.*

2.5.2 Algorytm klasycznego algorytmu genetycznego.

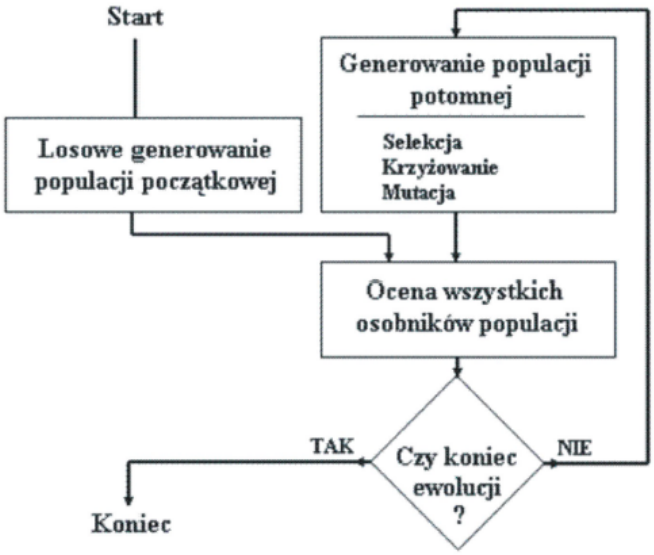
Na podstawowy (klasyczny) algorytm genetyczny, nazywany także elementarnym lub prostym algorytmem genetycznym, składają się kroki:

1. inicjacja czyli wybór początkowej populacji chromosomów,
2. ocena przystosowania chromosomów w populacji,
3. sprawdzenie warunku zatrzymania,
4. selekcja chromosomów,
5. zastosowanie operatorów genetycznych,

6. utworzenie nowej populacji,

7. wyprowadzenie najlepszego chromosomu.

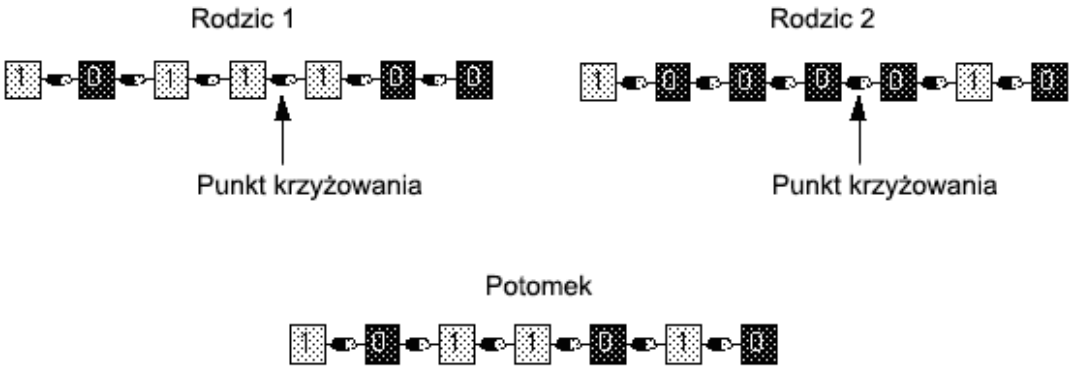
Najłatwiej wyobrazić sobie powyższe kroki analizując je na schemacie:



Rysunek 2.9: Ogólny schemat algorytmu genetycznego.

2.5.3 Operacje klasyczne, krzyżowanie.

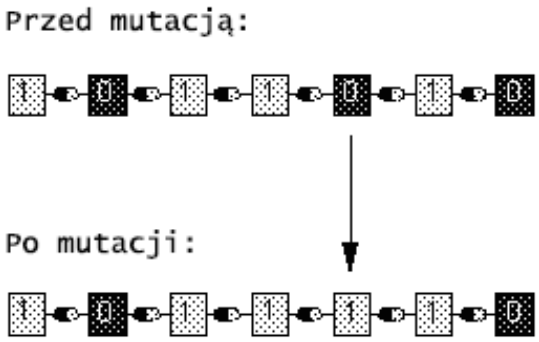
Operacja krzyżowania jest podstawową operacją algorytmów genetycznych służącą do rekombinacji materiału genetycznego. Operacja opiera się na dwóch chromosomach, których części materiału genetycznego zostają wymieszane w celu otworzenia nowego chromosomu. Podstawowa operacja krzyżowania opiera się o jeden punkt krzyżowania i została przedstawiona na poglądowym schemacie 2.10 [11].



Rysunek 2.10: Ogólny schemat operacji krzyżowania.

2.5.4 Operacje klasyczne, mutacja.

Proces rekombinacji przez krzyżowanie nie byłby w stanie odkryć całej przestrzeni poszukiwań, jeżeli dana kombinacja nie byłaby obecna w sekcjach populacji. To mogłoby prowadzić do błędnego wyniku. Operacja mutacji pozwala na wprowadzenie nowych struktur genetycznych w obecnej populacji. Dokonuje tego poprzez losową zmianę dowolnego genu w chromosomie[11].



Rysunek 2.11: Ogólny schemat operacji mutacji.

2.6 Użycie grafów w algorytmie genetycznym.

W swojej pracy posługuję się przede wszystkim grafami. W tym wypadku klasyczna odmiana algorytmu genetycznego musiałaby zostać zmodyfikowana na potrzeby wykorzystania grafów jako chromosomów. Dodatkowo wymagałoby to zastosowania nowych sposobów krzyżowania i mutacji jednostek.

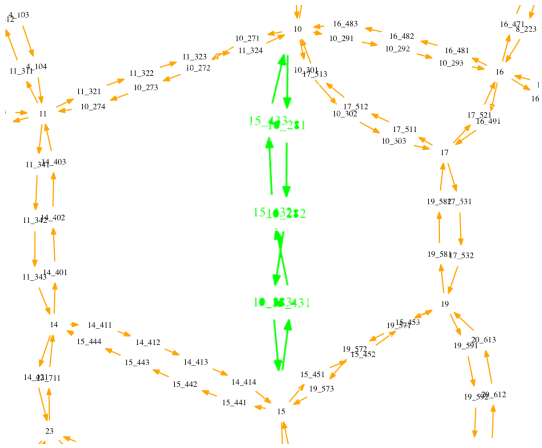
Zdecydowaliśmy, że zamiast przystosowywać algorytm genetyczny do nowej struktury, przystosujemy strukturę do algorytmu. Ponieważ naszym zadaniem jest zdecydowanie o zamknięciu lub nie, danej ulicy (krawędzi grafu) postanowiłem opisać tę strukturę jako listę opisującą ten stan. Zgodnie z tą myślą, sieć drogowa (graf) jest tłumaczona na tablicę elementów przyjmujących wartości:

- 1 (prawda) - dla ulicy (węzła), który jest przejezdny,
- 0 (fałsz) - dla ulicy (węzła) zamkniętego dla ruchu.

Tablica tworzona w ten sposób może być modyfikowana przez algorytm genetyczny, jak również bez problemu możemy z niej odtworzyć stan obecny sieci drogowej. Za przykład podajemy sytuację przedstawioną na rysunkach 2.12 i 2.13.

1	1	0	0	0	0	0

Rysunek 2.12: Fragment sieci w postaci tablicy binarnej

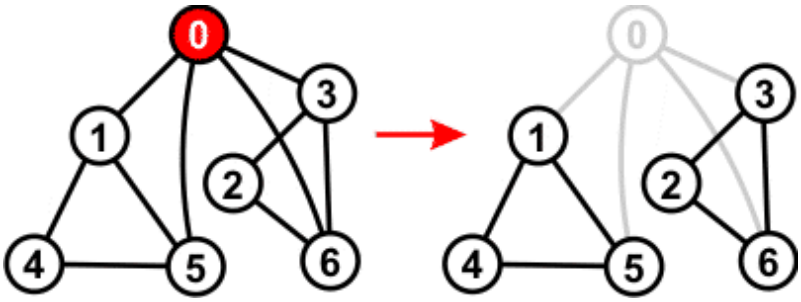


Rysunek 2.13: Fragment sieci w postaci grafu

Przedstawiony na rysunku 2.12 fragment sieci zawiera kolejne elementy zerowe, które w tłumaczeniu na przykładowy graf 2.13, są zaznaczone kolorem zielonym. W tym przypadku krawędzie te zostaną wyłączone z ruchu.

2.7 Punkty artykulacji grafu.

Punktem artykulacji[1] jest wierzchołek, którego usunięcie z grafu spowoduje zwiększenie liczby spójnych składowych. Na rysunku 2.14, punktem artykulacji jest wierzchołek o numerze 0.



Rysunek 2.14: Przykład grafu z zaznaczonym punktem artykulacji.

Algorytm poszukiwania punktów artykulacji wykorzystuje przeszukiwanie grafu w głąb. Podczas nierekurencyjnego przejścia, wierzchołek zostaje oznaczony jako punkt artykulacji, wtedy i tylko wtedy, jeśli jest on korzeniem posiada więcej niż dwóch synów. A wierzchołek v nie będący korzeniem drzewa DFS jest punktem artykulacji wtedy i tylko wtedy, gdy przynajmniej dla jednego jego syna nie istnieje krawędź wtórna $\{u, w\}$, taka że wierzchołek u jest potomkiem v i w jest przodkiem v.

[1] ang. articulation point lub cut vertex

Punkty te są wykorzystywane, by podczas losowych mutacji i krzyżowań w algorytmie genetycznym nie doszło do stworzenia grafu innego niż grafu skierowanego silnie spójnego. Krok sprawdzenia jest wykonywany podczas każdorazowego zamykania drogi, zatem za każdym razem gdy zostaje wprowadzona nowa wartość , , 0″ w tablicy.

Rozdział 3

Technologie i metody użyte.

3.1 Symulator transportu.

MATSim jest środowiskiem[1] do implementacji szerokiej skali symulacji transportu opartych na agentach. Składa się z wielu modułów, które mogą zostać połączone lub używane oddzielnie. Moduły można zastępować własnymi implementacjami w celu przetestowania pojedynczych aspektów pracy[12].

Rysunek 3.1: Logo symulatora transportu MATSim

Oczywiście MATSim nie jest jedynym dostępnym symulatorem transportu. Podobnych rozwiązań jest wiele. Wybór motywowaliśmy jednak przede wszystkim dostępnością materiałów szkoleniowych i przykładowymi scenariuszami. MATSim jest żywym projektem oparty o licencję open source[2]. Dysponuje szerokim zasobem przykładów i gotowych scenariuszy, w dodatku udostępniając prace wielu osób na swoim repozytorium.

[1] ang. framework, pl. szkielet do budowy aplikacji
[2] pol. otwarte oprogramowanie

3.2 Algorytmy genetyczne.

Projekt The Apache Commons jest tworzony przez Apache Software Foundation. Głównym celem projektu jest stworzenie wolnego oprogramowania do wielokrotnego użytku w języku Java. Projekt podzielony jest na trzy główne części: proper, sandbox, i dormant[13].

W przypadku głównej części Apache Commons Proper wyróżniamy wiele modułów różniących się funkcjonalnością i celem. W przypadku Apache Commons Math jest to moduł zapewniający rozwiązywania problemów głównie matematycznych i statystycznych. Znajduje się w nim implementacja podstawowej formy algorytmu genetycznego, którą rozszerzamy w pracy.

Rysunek 3.2: Logo biblioteki Apache Commons Math

3.3 Obsługa grafów.

Ponieważ moduł do wizualizacji rozwiązań MATSim nie spełniał naszych oczekiwań zarówno pod względem wydajności jak i stabilności zdecydowaliśmy się na stworzenie własnej implementacji. W tym celu wykorzystałem dojrzały projekt NetworkX. Jest to biblioteka wykonana w języku Python stworzona z myślą o grafach i sieciach. Jest ona dostarczana jako darmowe oprogramowanie na licencji BSD new[14].

Rysunek 3.3: Logo biblioteki NetworkX

3.4 Technologie i metodologie programistyczne.

Użyte przez nas technologie są poniekąd wymuszone przez języki w jakich zostały stworzone wykorzystywane pomocnicze rozwiązania. W przypadku symulatora MATSim jest to Java. Zdecydowaliśmy się natomiast na wykorzystanie Pythona głównie ze względu na przeważającą przewagę biblioteki NetworkX nad bibliotekami obsługującymi

grafy w języku Java. Nie jest to wyjątek, gdyż ogólnie rozwiązania dostarczane dla Python'a, szczególnie w przypadku problemów akademickich, są dużo lepsze. Poniżej zestaw narzędzi, które wykorzystaliśmy w procesie tworzenia aplikacji.

Rysunek 3.4: Logo Java.

Rysunek 3.5: Logo IDE Eclipse.

Rysunek 3.6: Logo Python.

Rysunek 3.7: Logo PyDev.

Loga pochodzą ze stron dostawców[15][16][17][18].

3.5 Badane miasto przykładowe: Siouxfalls

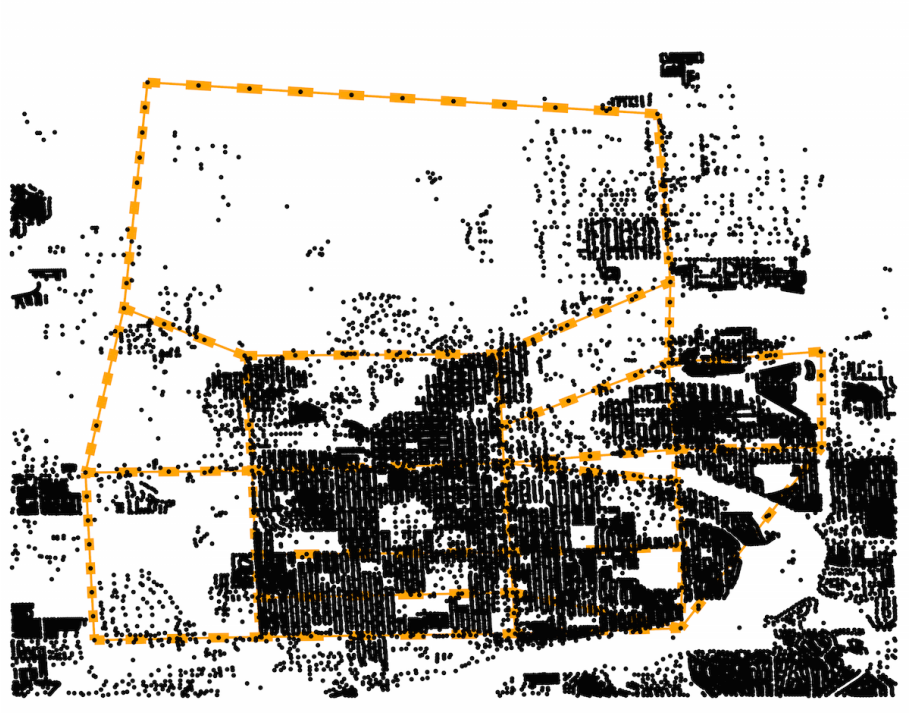
Główną zaletą korzystania z MATSima, o której już wcześniej wspomniałem, jest dość pokaźny zbiór danych przykładowych. Jednym z nich jest materiał zaprezentowany przez twórców aplikacji, który udostępnili w 2013 roku na zgromadzeniu użytkowników platformy[8]. Przykład ten dotyczy miasta Sioux Falls w Południowej Dakocie. Domyślnie scenariusz składa się z:

- dwóch grup zapotrzebowania bez charakterystyk socjodemograficznych:
  - 68094 agentów z samochodem oraz korzystających z transportu publicznego,
  - 40877 agentów posiadających samochód.
- dostosowanej sieci drogowa miasta Sioux Falls,
- transportu publicznego razem z rozkładem jazdy,
- przykładowych miejsc zamieszkania, pracy i rozrywki.

Po zapoznaniu się z przykładem, dostarcza on nawet za dużo danych, które by mnie interesowały. Dostosowałem go zatem do swoich potrzeb poprzez:

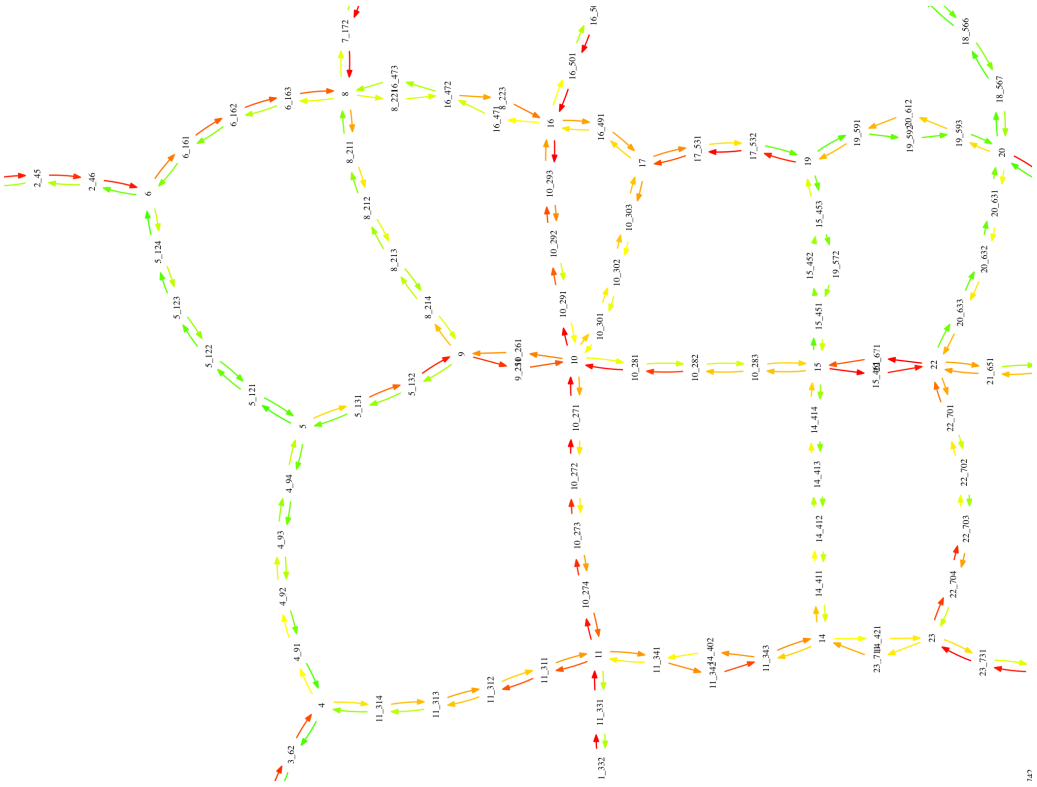
- usunięcie transportu publicznego,
- wyposażenie każdego agenta w swój samochód.

Powyższe modyfikacje znacznie wzmogły ruch w mieście (co było dla mnie pozytywnym efektem) i skróciły obliczenia związane z symulacjami. Poniżej prezentuję rozkład miejsc pracy, domostw i innych zakładów nałożonych na graf sieci drogowej miasta.

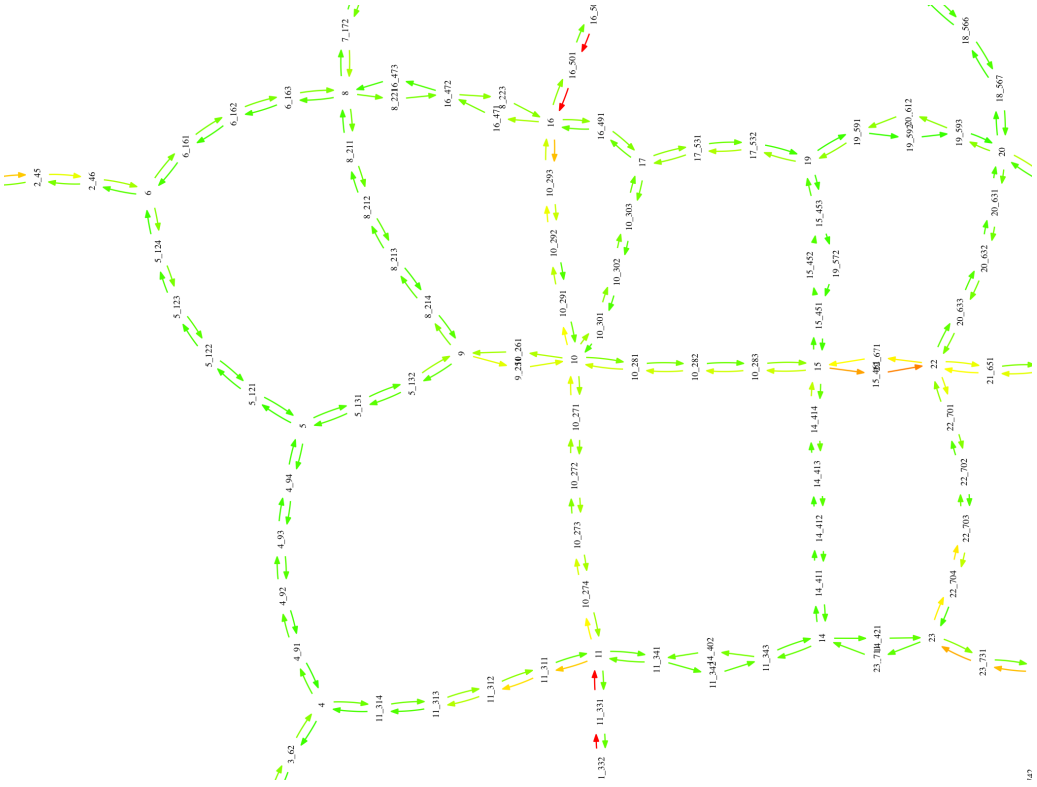


Rysunek 3.8: Rozkład budynków na grafie miasta Sioux Falls.

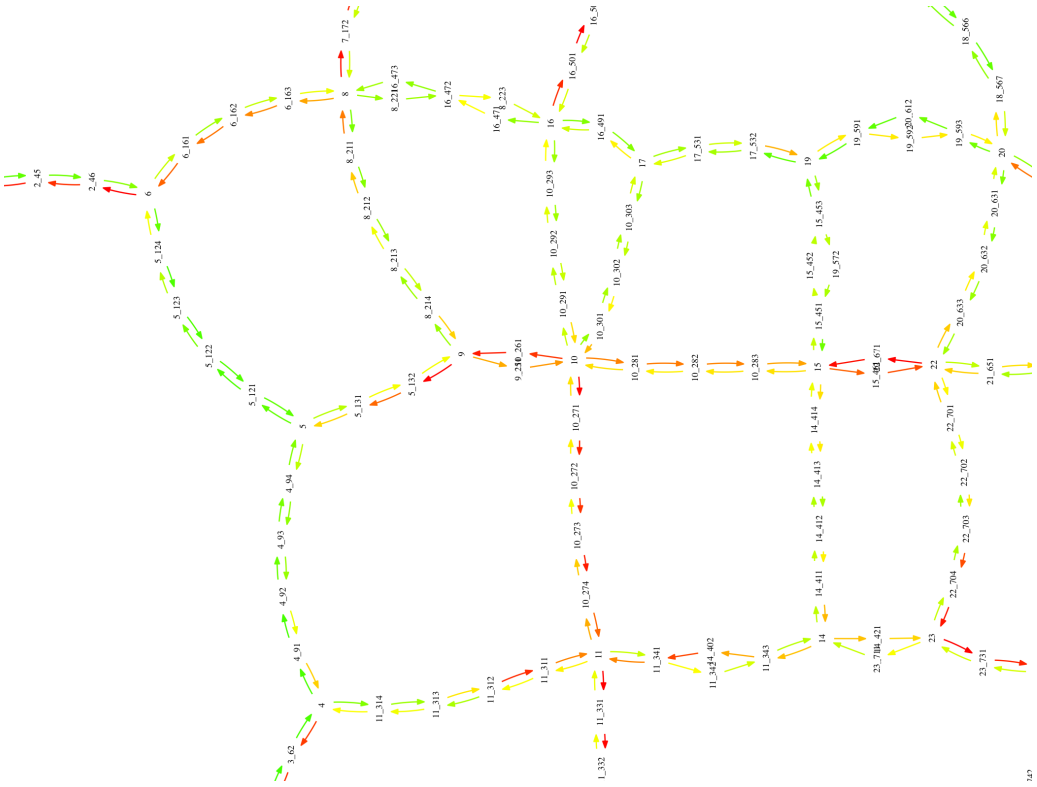
Prezentuję również ruch drogowy w najbardziej intensywnych godzinach dnia, tj. godzinach szczytu. Ruch jest przedstawiony jako natężenie ruchu na każdej krawędzi (ulicy), poprzez odniesienie go do możliwości przepustowości każdego węzła. Kolor zielony oznacza małe obciążenie, żółty średnie i czerwony - wysokie natężenie ruchu w tym miejscu. Obrazki są obrócone o 90° w lewo, dla zwiększenia ich czytelności i rozmiaru.



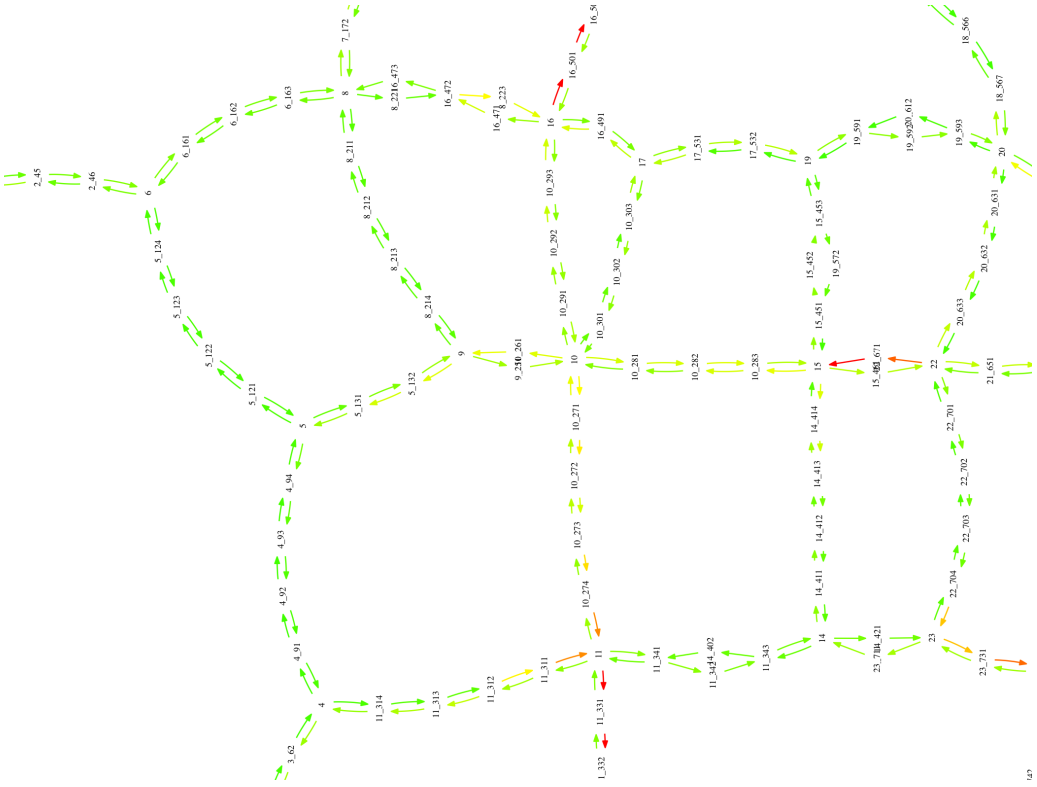
Rysunek 3.9: Natężenie ruchu w mieście Sioux Falls w godzinach 6.00-7.00.



Rysunek 3.10: Natężenie ruchu w mieście Sioux Falls w godzinach 7.00-8.00.



Rysunek 3.11: Natężenie ruchu w mieście Sioux Falls w godzinach 16.00-17.00.



Rysunek 3.12: Natężenie ruchu w mieście Sioux Falls w godzinach 17.00-18.00.

Rozdział 4

Opis projektu.

W tym rozdziale opiszę strukturę swojego projektu z opisem jego uruchomienia. Ponieważ przeprowadzane przez projekt obliczenia wymagają wiele czasu, zdecydowałem się na pominięcie interfejsu użytkownika przy projektowaniu aplikacji. Całość jest obsługiwana przez plik konfiguracyjny, który zostaje wczytany na początku operacji i za jego pomocą kontrolujemy przebieg obliczeń. Pomimo dwóch osobnych technologii, w których wykonałem projekt, są one od siebie zależne. Całość jest obsługiwana przez projekt w Java, skrypty w Pythonie jedynie wspierają niektóre procesy.

4.1 Dane wejściowe.

Poniżej przedstawiam przykładowy plik konfiguracyjny dla projektu z opisem jego funkcji.

Listing 4.1: Plik konfiguracyjny projektu

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
   <project>
      <name>siouxfalls-test</name>
      <output-dir>../output/</output-dir>
      <threads>4</threads>
      <log-level>INFO</log-level>
      <python-path>/usr/local/bin/python</python-path>
      <python-main>../python/ms/call_center.py</python-main>
   </project>
   <scenario>
      <config>../scenarios/siouxfalls/config.xml</config>
      <network>../scenarios/siouxfalls/network.xml</network>
      <population>../scenarios/siouxfalls/population.xml</population>
      <facilities>../scenarios/siouxfalls/facilities.xml</facilities>
      <iterations>1</iterations>
```

```
   </scenario>
   <genetics>
      <population-size>4</population-size>
      <max-generations>3</max-generations>
      <elitism-rate>0.15</elitism-rate>
      <crossover-rate>1</crossover-rate>
      <mutation-rate>0.1</mutation-rate>
      <tournament-arity>2</tournament-arity>
   </genetics>
</config>
```

Jak widać konfiguracja dzieli się na trzy podstawowe grupy:

- project,
- scenario,
- genetics.

Grupa *project* odpowiada za główne ustawienia całego projektu, w *scenario* znajdziemy ustawienia dotyczące symulacji przeprowadzanej przez MATSim, natomiast sekcja *genetics* zawiera ustawienia dotyczące algorytmu genetycznego.

Wyjaśnienie opcji *project*:

- name - nazwa projektu, używana jako katalog wyjściowy
- output dir - katalog, gdzie zapisujemy wyniki
- threads - ilość wątków, które mają być użyte podczas obliczeń
- log level - poziom logowania Log4J[1]
- python path - ścieżka instalacji Pythona
- python main - folder ze skryptami pomocniczymi

Wyjaśnienie opcji *scenario*:

- config - scieżka dostępu pliku konfiguracyjnego scenariusz MATSim
- network - scieżka dostępu pliku z siecią wejściową scenariusza
- population - scieżka dostępu pliku z populacja scenariusza
- facilities - scieżka dostępu pliku z budynkami scenariusza

[1] zewnętrzna biblioteka do logowania, dostępne poziomy: DEBUG, INFO, WARN, ERROR i FATAL

- iterations - ilość iteracji symulacji MATSima

Wyjaśnienie opcji *genetics*:

- population size - rozmiar populacji
- max generations - ilość testowanych generacji (warunek stopu)
- elitism rate - ułamek najlepszych chromosomów populacji biorący udział w kolejnej iteracji
- crossover rate - szansa na krzyżowanie osobników
- mutation rate - szansa na mutacje osobników
- tournament rate - ilość osobników biorących udział w turnieju

4.2 Wdrożenie.

Ze względu na duże wymagania sprzętowe obliczeń, a zarazem ograniczoną przenośność rozwiązania z powodu skorzystania z Pythona, zdecydowałem się na usprawnienie rozwiązania. Wykorzystując system Linux Trisquel stworzyłem maszynę wirtualną spełniającą wszystkie wymagania do uruchomienia aplikacji. Dzięki temu mogłem wykorzystać inne maszyny oprócz tej, na której tworzyłem rozwiązanie, bez problemu przystosowania czy instalowania dużej ilości zewnętrznego oprogramowania i bibliotek[19].

Rysunek 4.1: Logo systemu Linux Trisquel.

4.3 Wyniki.

Rozdział 5

Podsumowanie.

5.1 Dyskusja wyników.

5.2 Perspektywy dalszych badań w dziedzinie.

5.3 Struktura projektu.

Spis rysunków

Spis tablic

Spis listingów

Bibliografia

[1] Leslie Arthur Keith Bloy, *An investigation into Braess' paradox*, 02/2007

[2] Rric Pas and Shari Principio *Braess' paradox: Some new insights*, April 1996

[3] Wataru Nanya, Hiroshi Kitada, Azusa Hara, Yukiko Wakita, Tatsuhiro Tamaki, and Eisuke Kita *Road Network Optimization for Increasing Traffic Flow* Int. Conference on Simulation Technology, JSST 2013.

[4] Ana L. C. Bazzan and Franziska Klügl *Reducing the Effects of the Braess Paradox with Information Manipulation*

[5] Dietrich Braess. *Über ein Paradoxon aus der Verkehrsplanung. „Unternehmensforschung"*. 12, s. 258–268, 1968 (niem.).

[6] Mitchell Melanie *An Introduction to Genetic Algorithms* First MIT Press paperback edition, 1998

[7] M. Rieser, C. Dobler, T. Dubernet, D. Grether, A. Horni, G. Lammel, R. Waraich, M. Zilske, Kay W. Axhausen, Kai Nagel *MATSim User Guide* updated September 12, 2014

[8] A. Chakirov *Enriched Sioux Falls Scenario with Dynamic Demand* MATSim User Meeting, Zurich/Singapore, June 2013.

[9] Łukasz Kowalik *Algorytmy i struktury danych, grafy* Wykład, 2003.

[10] Marta Grzanek *Sztuczna inteligencja, Klasyczny algorytm genetyczny* Wykład, 2007.

[11] Dilvan de Abreu Moreira *Agents: A Distributed Client/Server System for Leaf Cell Generation* Thesis, 1995

[12] http://matsim.org

[13] http://commons.apache.org/proper/commonsmath

[14] http://networkx.github.io

[15] http://www.java.com/pl/

[16] https://eclipse.org

[17] http://pl.python.org

[18] http://pydev.org

[19] https://trisquel.info

[20] http://pl.wikipedia.org/wiki/Paradoks_Braessa

[21] http://urbnews.pl/paradoksbraessa/

[22] http://pl.wikipedia.org/wiki/Paradoks_DownsaThomsona

[23] http://pl.wikipedia.org/wiki/Prawo_Lewisa Mogridge'a

[24] http://mathworld.wolfram.com/StronglyConnectedDigraph.html

Abstract

Yet to come...