AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA AUTOMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa magisterska

Aplikacja znajdująca najkrótszą drogę w supermarkecie
An application finding the shortest path in a supermarket

Autor: *Krzysztof Olszowy*
Kierunek studiów: *Automatyka i Robotyka*
Opiekun pracy: *dr hab. Adrian Horzyk*

Kraków, *2017*

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Podpis dyplomanta ……

Składam serdeczne podziękowania wszystkim osobom,

które przyczyniły się do powstania poniższej pracy

dyplomowej. W szczególności dziękuję

Panu dr hab. Adrianowi Horzykowi

za pomoc, cenne uwagi merytoryczne

i udzielone wsparcie.

Spis treści

Wstęp

Inspiracją do napisania poniższej pracy dyplomowej są codzienne zakupy przeprowadzane w dużych krakowskich supermarketach. W ostatnich latach można było zauważyć rosnącą liczbę powstawania tych wielkich zachodnich sklepów, które poszerzają u nas swoje rynki zbytu. Kłopot stanowią zwłaszcza miejsca, gdzie tanie supermarkety doprowadziły do bankructwa konkurencje w postaci małych osiedlowych sklepików. W takich przypadkach mieszkańcy muszą nawet po małe zakupy wybrać się do supermarketu i szukać kilku produktów na setkach metrów kwadratowych powierzchni. Poruszanie się po tak dużej przestrzeni sklepowej może być uciążliwe, zwłaszcza gdy nie znamy sklepu i musimy szukać każdego z produktów. Jednak większość mieszkańców miasta decyduje się na zakupy w supermarketach dopiero na potrzeby dużych zakupów. W przypadku braku dobrej organizacji przy takich zakupach możemy utknąć na kilka godzin z koszykiem w ręku.

W wymienionych przypadkach z pomocą przychodzi aplikacja zaprezentowana w poniższej pracy. Jest ona przydatna zwłaszcza, że to właśnie czas jest tak istotną wartością we współczesnym świecie. Aplikacja ma na celu znalezienie najkrótszej drogi potrzebnej na wykonanie zakupów w supermarkecie. Znalezienie tej drogi jest równe z optymalizacją czasu zakupów. Przytoczone zagadnienie jest problemem komiwojażera. Przy większej liczbie danych wejściowych nie ma optymalnego rozwiązania tego problemu. Dobrym pomysłem będzie zastosowanie kilku algorytmów o różnej złożoności obliczeniowej i porównanie wyników. Do zaimplementowania aplikacji użyto języka programowania JAVA.

Materiały użyte do pracy magisterskiej gromadzone były w różny sposób. Zarówno na stronach internetowych jak i w książkach znajduje się wiele informacji na temat samego problemu komiwojażera, a także sposobów jego rozwiązania. Bardzo przydatna okazała się wiedza zdobyta na zajęciach z metod optymalizacji podczas studiów. Większość opisanych w tej pracy metod została już omówiona przez profesorów AGH. Samych algorytmów do rozwiązania problemu komiwojażera jest mnóstwo i nie sposób wszystkich przetestować, czy choćby opisać w jednej pracy magisterskiej.

Praca dyplomowa podzielona jest na cztery główne rozdziały. Pierwsze dwa opisują w sposób teoretyczny problem komiwojażera i kilka ciekawych sposobów jego rozwiązania. Kolejne dwa rozdziały przybliżają metody wykorzystane w tworzonej aplikacji. Omówiony jest sposób realizacji projektu od pseudokodu, aż po graficzne rozwiązania.

W pierwszym rozdziale przybliżony jest sam problem komiwojażera. Jako że jest to popularne zagadnienie wielu naukowców zmagało się już z jego rozwiązaniem. Opisano tutaj efekty tych starań. Przedstawiono także, jakie cechy posiada typowy problem komiwojażera oraz przybliżono spektrum jego zastosowań. Na koniec rozdziału poruszono temat złożoności obliczeniowej zagadnienia.

W rozdziale drugim przedstawiono kilka najbardziej znanych metod rozwiązania problemu komiwojażera. Dokonano podziału na metody heurystyczne, aproksymacyjne i inteligentne. Zaproponowano także algorytmy potrzebne do obliczenia końcowego rozwiązania, celem których było ustalenie najkrótszej drogi między dwoma produktami w sklepie.

Trzeci rozdział przybliża działanie samej aplikacji. W pierwszej części opisano zaprojektowany model sklepu oraz sposób poruszania się klienta. W drugiej części została szczegółowo opisana zasada działania każdego z wykorzystanych algorytmów wraz z zaprezentowanym pseudokodem.

W ostatnim rozdziale przeprowadzono testy wszystkich zaimplementowanych metod. Przedstawiono listę produktów wraz z ich umiejscowieniem w sklepie. Metody testowane są pod wieloma względami, między innymi pod względem czasu potrzebnego na obliczenie wyniku. Finalnie porównano wyniki wykorzystanych algorytmów zarówno liczbowo jak i graficznie.

Na zakończenie pracy dyplomowej autor wyciąga wnioski z napisanej pracy i stworzonej aplikacji. Przeanalizowano, które elementy pracy wykonano prawidłowo, gdzie popełniono błędy, oraz w którym etapie pracy należy dokonać zmian, aby usprawnić ten projekt na przyszłość.

1. Problem komiwojażera

1.1. Zagadnienie na przestrzeni lat

Problem komiwojażera (TSP – ang. Travelling Salesman Problem) to zagadnienie, które ma na celu optymalizację przebytej drogi pod względem odległości, czasu lub kosztów przejazdu. Rozwiązaniem problemu może być odpowiedź na pytanie: Jaka jest najkrótsza droga, która odwiedza każde miasto dokładnie raz i wraca do punktu wyjścia? W tym przypadku musimy znać dokładną ilość miast i odległości między każdą ich parą.

Problem podróży sprzedawców po miastach był rozważany już początkiem XIX wieku przez irlandzkiego matematyka Williama Rowana Hamiltona i brytyjskiego matematyka Thomasa Kirkmana. W poradnikach dla podróżujących sprzedawców obejmujących wycieczki po Niemczech i Szwajcarii wspomniano o złożoności problemu. Jednak problem został zdefiniowany dopiero w 1930 roku przez austriackiego matematyka Karla Mangera. Uważał on, iż jest to algorytm „brute-force", czyli algorytm polegający na sprawdzeniu wszystkich możliwych rozwiązań problemu i wybraniu tego najbardziej optymalnego. Karl Mangera zauważył, że problem komiwojażera można rozwiązać przez skończenie wiele prób. Natomiast algorytm, który mówi o poruszaniu się do następnego najbliższego miasta nie oznacza najkrótszej drogi. [1]

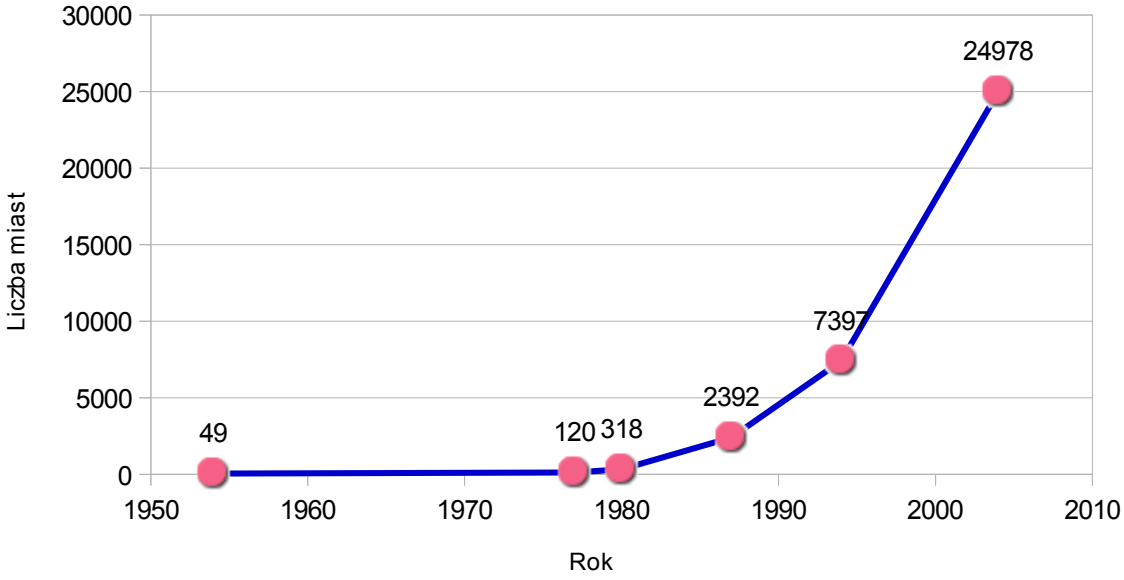
W latach pięćdziesiątych i sześćdziesiątych problem komiwojażera stał się coraz bardziej popularny w kręgach naukowych. Zaproponowano nawet nagrodę za rozwiązanie problemu. Duży wkład w tej dziedzinie wznieśli George Dantzig, Delbert Fulkerson i Selmer Johnson. Określili oni problem jako programowanie liniowe i rozwiązali przykład z 49 miastami, opracowaną przez siebie metodą cięcia. Metodę tą później nazwano metodą podziału i ograniczeń. Polega ona na ograniczeniu wartości rozwiązań, które uzyskujemy z potomnych węzłów minimalnego drzewa rozpinającego (MST – ang. Minimum Spanning Tree). Możemy dzięki temu oddzielić obszary o obiecującym rozwiązaniu od innych obszarów. [2]

W kolejnych dziesięcioleciach problem był badany przez wielu naukowców. Przełom przyniosło dopiero nowe podejście, w którym zamiast poszukiwania optymalnego rezultatu tworzono rozwiązanie, którego długość jest w sposób wystarczający ograniczona przez wielokrotność optymalnej długości, a tym samym tworzy niższe granice dla problemu. Taką metodą było utworzenie minimalnego drzewa rozpinającego, czyli takiego drzewa, które zawiera wszystkie wierzchołki badanego grafu o najmniejszej sumie wag krawędzi.

Należało podwoić wszystkie krawędzie MST co skutkowało tym, że długość optymalnej trasy była co najwyżej dwukrotnością wag MST. Wykorzystując to nowatorskie podejście w 1976 roku Christofides stworzył algorytm zwany jego imieniem, którego rozwiązanie było w najgorszym przypadku 1,5 razy dłuższe niż optymalne. [1]

W 1977 roku Martin Groetshel znalazł optymalną trasę składającą się ze 120 miast ówczesnych Zachodnich Niemiec. Wykorzystał on metodę płaszczyzn tnących. Dziesięć lat później wraz z Olafem Holladem powiększył tę liczbę do 666 miast znajdujących się już na całej półkuli ziemskiej. Jednak tego samego roku, ten rekord został mu odebrany przez Padberga i Rinaldiego, którzy jako pierwsi przekroczyli liczbę 1000, korzystając z bazy punktów firmy Tektronics Incorporated.

W latach dziewięćdziesiątych czwórka naukowców Applegate, Bixby, Chvatal i Cook opracowała program Concorde. Był on używany w wielu najnowszych rozwiązaniach związanych z problemem komiwojażera. W 2004 roku dzięki niemu padł rekord dla optymalnej trasy komiwojażera wynoszący 24 978 szwedzkich miast. Naukowcy wykorzystali metodę simplex CPLEX. Przez blisko pół roku 96 komputerów, po 2 procesory każdy, prowadziło obliczenia w celu znalezienia optymalnego rozwiązania. Trasa miała długość około 72 500 km. Interesujący jest fakt, że rozwiązanie gorsze od optymalnego zaledwie o 0.033% udało się otrzymać już w ciągu 30 minut. Rozwój algorytmów optymalizujących trasę komiwojażera przedstawiony został na rysunku 1.



Rys. 1. Rozwój algorytmów optymalizujących trasę komiwojażera [3]

Największym, jak dotąd przykładem rozwiązania TSP jest przeprowadzona przez grupę badawczą Concorde trasa składająca się z 85 900 punktów. Obliczenia zostały opisane w książce pt. „The Traveling Salesman Problem: A Computational Study”. Zestaw punktów został zebrany z mikrochipa LaserLogic, a problem rozwiązany w 2006 roku przedstawia ścieżkę łączącą układy scalone. [3]

1.2. Rozwinięcie problemu

Problem komiwojażera ma wiele zastosowań we współczesnych dziedzinach takich jak logistyka, planowanie czy produkcja mikrochipów. Po lekkiej modyfikacji jest wykorzystywany na przykład do sekwencjonowania DNA. W tych przypadkach koncepcyjne miasto reprezentują klienci, punkty lutowania czy fragmenty DNA. TSP używane jest także w astronomii. Astronomowie obserwując wiele gwiazd chcą zminimalizować czas spędzony na teleskopie podczas przemieszczania się między źródłami. Szerokie spektrum zastosowań sprawiło, że ten problem jest tak popularny.

TSP polega na odnalezieniu minimalnego cyklu Hamiltona. Jest to cykl, w którym każdy wierzchołek w grafie poza wyjściowym odwiedzamy dokładnie raz. Cykl Hamiltona dla TSP zachodzi w pełnym grafie ważonym, czyli takim zbiorze wierzchołków, gdzie dla każdej ich pary istnieje krawędź je łącząca o określonej wadze. Rozwiązaniem jest taki cykl, w którym suma wag krawędzi będzie minimalna.

Rozróżniamy symetryczny i asymetryczny problem komiwojażera. Symetryczny to taki, w którym odległości między dwoma badanymi miastami są zawsze takie same niezależnie od wierzchołka początkowego. Analogicznie w asymetrycznym TSP odległość z miasta A do B jest różna niż odległość z B do A, co dodatkowo komplikuje problem.

Co łatwo zauważyć, kluczową rolę w problemie komiwojażera pełni odległość. Z tego względu TSP dzielimy na problem metryczny i niemetryczny. Jeśli długości krawędzi w cyklu Hamiltona spełniają nierówność trójkąta, jest to problem metryczny. Bezpośrednie połączenie dwóch miast, a właściwie odległość między nimi będzie w każdym przypadku mniejsza lub równa od odległości wykorzystującej do połączenia tych miast dodatkowy wierzchołek. Ten przypadek ma miejsce tylko przy minimalizacji samej drogi. Jeśli pod uwagę weźmiemy np. koszty przejazdu, może okazać się, że bezpośrednie połączenie samolotem jest droższe, niż przejazd pociągiem, którego trasa przebiega przez dodatkowe miasto.

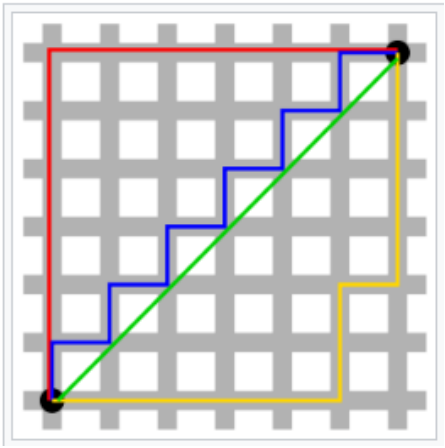
Na podstawie powyższego wyróżniamy metrykę euklidesową, czyli bezpośrednia odległość między miastami. W przestrzeni R^n metryka ta wyrażona jest wzorem:

$$d_e(\mathbf{x}, \mathbf{y}) = \sqrt{(y_1 - x_1)^2 + \cdots + (y_n - x_n)^2}$$

Z kolei suma wartości bezwzględnej różnic między dwoma wierzchołkami nosi nazwę metryki miejskiej. W tym przypadku krawędzie łączące miasta będą tylko w pionie i poziomie, co bardzo przyda nam się w tej pracy magisterskiej. Dla dwóch miast metryka wynosi:

$$d_m(\mathbf{x}, \mathbf{y}) = |x_1 - y_1| + |x_2 - y_2|.$$

Metryka miejska zwana metryką Manhattan swoją nazwę wywodzi od nowojorskich taksówek poruszających się po przecznicach miasta. Wykorzystywana jest głównie przy minimalizacji czasu przejazdu lub ograniczeń w poruszaniu się między punktami po linii prostej. Jej porównanie z metryką euklidesową obrazuje rysunek 2. [4]



Rys. 2. Porównanie metryki euklidesowej (kolor zielony) z metryką miejską (pozostałe kolory) [4]

1.3. Złożoność obliczeniowa

Niestety dla problemu komiwojażera nie istnieje ogólny i równocześnie efektywny algorytm dający optymalne rozwiązanie. TSP jest problemem NP-trudnym. Istnieje cała klasa problemów o czasowej złożoności większej od wielomianowej. Z tego względu dla problemów NP utworzono precyzyjniejsze podzbiory umożliwiające bardziej szczegółowe określenie problemu.

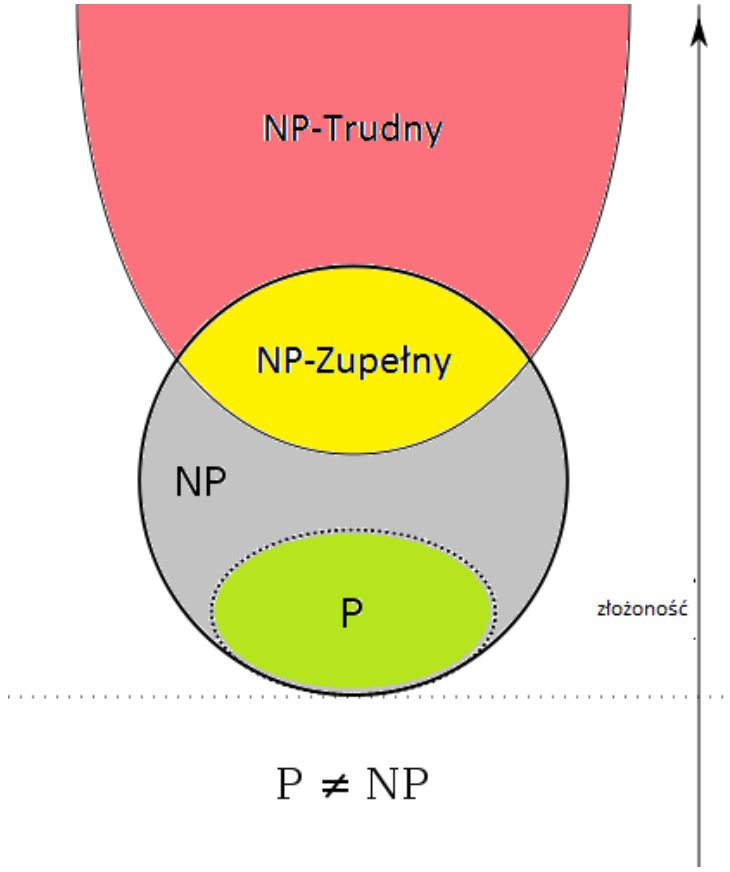
Dla wyjaśnienia omówimy sobie najważniejsze klasy, których zależność ilustruje rysunek 3, pod warunkiem, że P i NP nie są tą samą klasą:

- problem P – problem decyzyjny, dla którego jesteśmy w stanie znaleźć rozwiązanie w czasie wielomianowym;

- problem NP – problem decyzyjny, w którym znamy rozwiązanie problemu, jeśli w czasie wielomianowym jesteśmy w stanie sprawdzić jego poprawność;

- problem NP-zupełny – problem decyzyjny, dla którego nie jesteśmy w stanie znaleźć rozwiązania w czasie wielomianowym;

- problem NP-trudny – problem obliczeniowy, dla którego nie istnieje rozwiązanie ze złożonością wielomianową, a sprawdzenie rozwiązania zagadnienia jest równie trudne co każdego innego problemu NP; [5]



Rys. 3. Zależność problemów P, NP, NP-zupełnych i NP-trudnych

Załóżmy, że chcemy rozwiązać TSP sprawdzając wszystkie możliwe rozwiązania. Przyjmując jako liczbę wszystkich miast *n* dla drugiego miasta mamy (*n*-1) możliwości, a dla trzeciego (*n*-2). Sumując dla trzech miast mamy (*n*-1)*(*n*-2) różnych kombinacji, a dla czterech miast (*n*-1)*(*n*-2)*(*n*-3). Rozwiązanie TSP tą metodą wymagałoby rozpatrzenia, aż (*n*-1)!/2 kombinacji. Widać, że z każdym kolejnym miastem ilość kombinacji, a więc również czas potrzebny na obliczenia rośnie o silnię. Dla kilku miast można sprawdzić jeszcze wszystkie przypadki, ale już dla kilkunastu jest to zbyt czasochłonne. Powiedzmy tylko, że dla 14 miast dostajemy ponad 3 miliardy kombinacji. Przy większej liczbie miast zadanie to staje się niewykonalne, nawet przy możliwościach dzisiejszych procesorów. Na szczęście mamy do dyspozycji o wiele efektywniejsze algorytmy. [6]

2. Metody rozwiązania problemu

W związku z brakiem praktyczności algorytmu dokładnego powstało wiele innych algorytmów przybliżonych. Począwszy od algorytmów heurystycznych, poprzez aproksymacyjne, aż po inteligentne, jak algorytm wykorzystujący inteligencje rojową. Współczesne metody potrafią w dość krótkim czasie dla tysięcy miast znaleźć rozwiązanie odbiegające od optymalnego zaledwie o kilka procent.

2.1. Heurystyka

Heurystyka polega na znalezieniu rozwiązania jak najbardziej zbliżonego do optymalnego. Stosowana jest najczęściej, gdy algorytm dokładny jest nieznany, na przykład przy wykrywaniu wirusów lub prognozie pogody. Heurystyka ma zastosowanie również w przypadku, gdy algorytmy dokładne są nieopłacalne. Dzieje się tak głównie, gdy chcemy skrócić czas działania programu, a rozwiązanie odbiegające od optymalnego o kilka procent jest dla nas satysfakcjonujące.

Metody heurystyczne można podzielić na dwie podgrupy: metody heurystyki konstruktywnej i ulepszającej. Algorytmy należące do pierwszej grupy budowane są stopniowo i nie modyfikują już wcześniej otrzymanego rozwiązania. Te drugie zaś ulepszają dostępne rozwiązania, aby otrzymać jak najbardziej zadowalający rezultat.

2.1.1. Algorytm zachłanny

Algorytm zachłanny polega na składaniu lokalnie optymalnych wyborów ścieżki z nadzieją na znalezienie globalnego optimum. W przypadku problemu komiwojażera metoda polega na wybraniu nieodwiedzonego jeszcze miasta znajdującego się w najbliższej odległości od naszej aktualnej pozycji. W większości przypadków ta chciwa strategia nie znajduje maksimum globalnego, ale zwraca najbliższe lokalne maksimum. Rozwiązanie to, w przeciwieństwie do algorytmu dokładnego kończy się w rozsądnej liczbie kroków. Algorytm zachłanny możemy scharakteryzować jako krótkowzroczny, ponieważ nie przewiduje więcej niż jednego kroku w przód. Jest to wada różniąca tę metodę od programowania dynamicznego, która w przeciwieństwie do algorytmu zachłannego podejmuje decyzje w oparciu o wszystkie wybory podjęte we wcześniejszych etapach i ponownie rozważa ścieżkę. Algorytm zachłanny w większości przypadków nie jest w stanie znaleźć globalnego rozwiązania optymalnego, ponieważ zazwyczaj nie działa wyczerpująco na wszystkich danych.

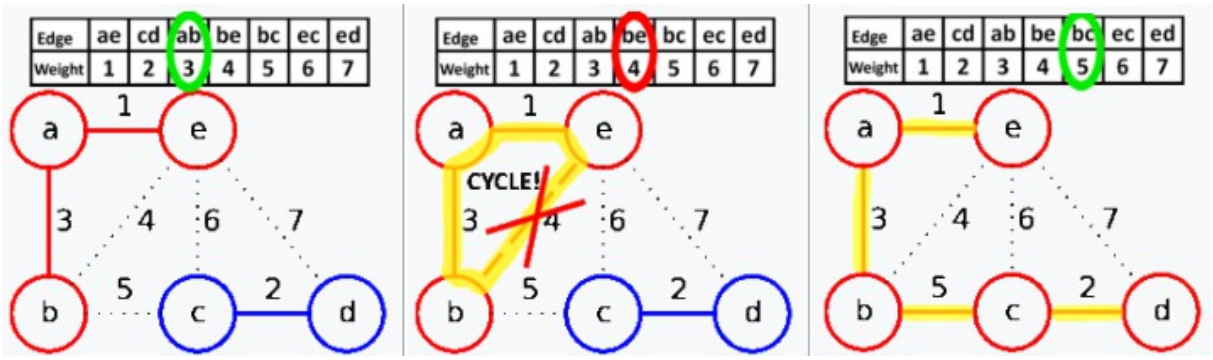
Metoda ta może podejmować decyzje zbyt wcześnie, co uniemożliwia jej znalezienie jak najlepszego ogólnego rozwiązania w kontekście całego problemu. Jeżeli rozpatrujemy przypadek, w którym algorytm zachłanny jest w stanie znaleźć rozwiązanie optymalne i możemy to udowodnić zazwyczaj metoda ta staje się głównym wyborem, ponieważ jest szybsza niż inne metody optymalizacji. Najbardziej znanymi przykładami algorytmów zachłannych są algorytm Kruskala i algorytm Prima.

Algorytm Kruskala znajduje minimalne drzewo rozpinające, znajdujące się w grafie ważonym. Drzewo to składa się z krawędzi, których waga jest możliwie jak najmniejsza. Algorytm Kruskala można opisać w poniższych krokach:

---

1. Utwórz zbiór wierzchołków F gdzie każdy wierzchołek jest osobnym drzewem.
2. Utwórz zbiór krawędzi S posortowany rosnąco.
3. Ze zbioru S usuń krawędź o najmniejszej wadze.
4. Jeżeli wybrana krawędź łączy dwa osobne drzewa, dodaj ją do zbioru F aby połączyła je w jedno drzewo.
5. Wracaj do kroku 3 dopóki S nie jest zbiorem pustym lub F nie tworzy drzewa rozpinającego.

Po zakończeniu algorytmu, jeżeli wykres jest połączony to zbiór wierzchołków F tworzy minimalne drzewo rozpinające. Zasadę działania tej metody algorytmu zachłannego prezentuje rysunek 4.



Rys. 4. Działanie algorytmu Kruskala przedstawione w 3 krokach [7]

Algorytm Prima, w przeciwieństwie do metody Kruskala nie łączy kilku drzew w jedno. Działa on przez rozbudowę stałego drzewa o jeden wierzchołek w jednej iteracji. Krawędź prowadzona jest z dowolnego wierzchołka istniejącego już drzewa, na każdym kroku dodając minimalną krawędź z drzewa do nowego wierzchołka grafu. [8]

Działanie algorytmu Prima możemy podsumować w kilku poniższych krokach:

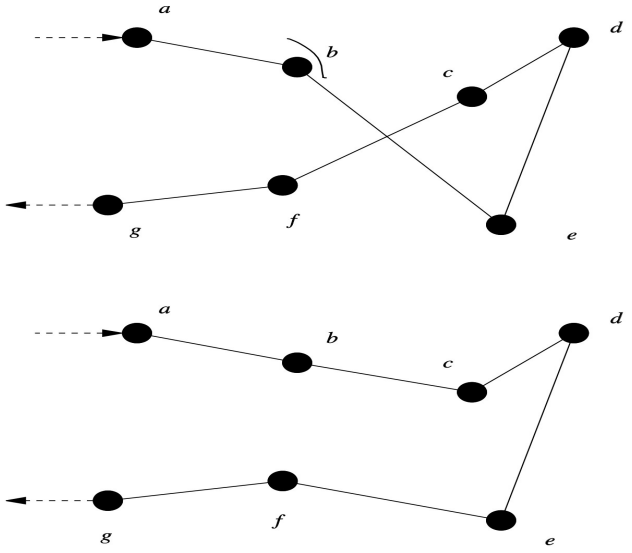
---

1. Zainicjuj drzewo z pojedynczym wierzchołkiem, dowolnie wybranym z grafu.
2. Znajdź krawędź nie należącą do drzewa o minimalnej wadze, której początkiem jest wierzchołek należący już do drzewa.
3. Dodaj wybraną krawędź i wierzchołek końcowy do drzewa.
4. Wracaj do kroku 2 dopóki drzewo nie zawiera wszystkich wierzchołków grafu.

2.1.2. Wymiana parami

Metoda zwana po angielsku „2-opt", której głównym założeniem jest znaleźć dwie przecinające się krawędzie w drzewie rozpinającym i zamienić je w taki sposób, aby się nie przecinały. Metoda ta modyfikuje już istniejące rozwiązanie, na przykład algorytm zachłanny zmniejszając jego koszt. Zaletą tej metody jest znaczna poprawa jakości rozwiązania w stosunku do czasu obliczeniowego. Przykład zastosowania tej metody pokazuje rysunek 5.

Istnieje też metoda „3-opt" bądź „V-opt" przeznaczona dla większej liczby połączeń. Polega ona na usunięciu 3 lub więcej przecinających się krawędzi w ścieżce i ponownym ich połączeniu w sposób optymalny. „V-opt" w przeciwieństwie do pozostałych jest adaptacyjny i w każdym kroku decyduje ile krawędzi między wierzchołkami trzeba zamienić, aby znaleźć krótszą ścieżkę. Najpopularniejszą heurystyką wykorzystującą ten algorytm jest metoda Lin-Kernighana. *[9]*



Rys. 5. Zastosowanie wymiany parami dla 2 przecinających się krawędzi [10]

2.1.3. Symulowane wyżarzanie

Symulowane wyżarzanie swoją nazwę bierze od wyżarzania w metalurgii, czyli kontrolowanego ogrzewania i chłodzenia materiału, aby zwiększyć rozmiar i jakość kryształów. Metoda ta jest techniką przybliżania globalnego optimum funkcji zwłaszcza w dużej przestrzeni wyszukiwania. Dla problemu komiwojażera symulowane wyżarzanie określamy w następujących krokach:

---

1. Wybierz dowolną parę wierzchołków (poza początkowym i końcowym jeśli takie są założenia problemu komiwojażera).
2. Zamień wybrane wierzchołki miejscami.
3. Jeżeli całkowity koszt uległ zmniejszeniu zaakceptuj zmianę, w przeciwnym przypadku wróć do ustawienia sprzed kroku 2.
4. Powtarzaj od kroku 1, aż do braku zmian w długości trasy.

Symulowane wyżarzanie z reguły nie daje optymalnego rozwiązania. Jednak jak w przypadku algorytmu zachłannego osiągamy jedynie minimum lokalne, tak w przypadku wyżarzania mamy możliwość wyjścia z tych minimów. [11]

2.1.4. Przeszukiwanie tabu

Algorytm wykorzystujący lokalne metody wyszukiwania lub najbliższe sąsiedztwo, po to, aby przenieść iteracje z jednego potencjalnego rozwiązania do ulepszonego. Dzieje się tak dopóki nie zostanie spełnione kryterium zatrzymania. Lokalne metody przeszukiwania mają tendencję do utknięcia w suboptymalnych regionach lub na płaskowyżach, gdzie wiele rozwiązań jest podobnej jakości. Chcąc uniknąć tych pułapek i zbadać obszary przestrzeni wyszukiwania, które pozostałyby niewykryte przez inne lokalne procedury, wyszukiwanie tabu dokładnie zbada sąsiedztwo każdego rozwiązania w miarę postępów wyszukiwania. Gdy rozwiązanie utknie w lokalnym minimum metoda przeszukiwania tabu pozwala na pogorszenie rozwiązania, aby wyjść z tej sytuacji. Wprowadzone są także specjalne zakazy, aby algorytm nie próbował wracać do poprzednich, wcześniejszych rozwiązań. Metoda ta wykorzystuje struktury pamięci opisujące odwiedzane już rozwiązania. Jeśli nastąpiło to w niedalekiej przeszłości lub potencjalne rozwiązanie naruszyło regułę, jest oznaczone jako tabu, aby algorytm nie uwzględniał tej możliwości wielokrotnie. Przeszukiwanie tabu często jest porównywane z innymi metodami rozwiązywania problemu komiwojażera, a nawet łączy się z innymi heurystykami tworząc nowe metody hybrydowe.

Struktury pamięci zawarte w tej metodzie możemy podzielić pod względem czasu na 3 kategorie. Krótkoterminowe to takie, w których mamy listę ostatnio otrzymanych rozwiązań. Jeśli na liście pojawi się potencjalne rozwiązanie nie można go wyświetlić dopóki nie osiągnie wymaganego progu. W średniookresowych strukturach pamięci zasady eskalacji rozwiązania mają wpływ na przeszukiwanie w kierunku obiecujących obszarów przestrzeni poszukiwań. Długoterminowe zaś kierują wyszukiwaniem do nowych regionów, gdy utknie ono na niewielkim obszarze. [12]

2.2. Algorytmy aproksymacyjne

Algorytm aproksymacyjny jest to algorytm, dla którego nie są znane szybkie metody znajdowania rozwiązania optymalnego. Różnicą między algorytmem aproksymacyjnym a heurystycznym jest informacja o jakości otrzymanego rozwiązania w stosunku do rozwiązania optymalnego.

2.2.1 Algorytm Christofidesa

Metoda znalezienia rozwiązania przybliżonego dla TSP, w przypadkach w których odstępy tworzą przestrzeń metryczną. W przypadku metody Christofidesa rozwiązanie jest 1,5 razy gorsze od rozwiązania optymalnego. Główną zaletą stosowania tej metody, w przeciwieństwie do heurystyki czy algorytmów genetycznych jest to, że już na samym początku gwarantuje ona nam jakość rozwiązania, która w pewnych przypadkach jest wystarczająca. Algorytm Christofidesa zawiera się w następujących krokach:

---

1. Utwórz minimalne drzewo rozpinającego dla badanego grafu.
2. Połącz w drzewie wierzchołki nieparzyste w taki sposób, aby waga grafu była minimalna.
3. Wierzchołki w wyznaczonym grafie mają równy stopień, więc istnieje cykl Eulera.
4. Wyznacz obwód Eulera i przekształć go w cykl Hamiltona.

Cykl Eulera to taka droga w grafie, która przechodzi przez każdą jego krawędź dokładnie raz. Natomiast cyklem Hamiltona nazywamy ścieżkę przechodzącą przez każdy wierzchołek grafu dokładnie raz, pod warunkiem, że ścieżka ta tworzy drogę zamkniętą. [13]

2.3. Metody inteligentne

Inteligentne metody obliczeniowe są stosowane w przypadku braku skuteczności tradycyjnych algorytmów. Powstały w wyniku obserwacji świata naturalnego. Sieci neuronowe inspirowane działaniem układu nerwowego, algorytm genetyczny na podstawie ludzkiej ewolucji i genetyki, a algorytm mrówkowy, jak sama nazwa wskazuje, wzięty z obserwacji tych zwierząt. Inteligencja obliczeniowa znacznie się rozwinęła w ostatnich latach i odgrywa dużą rolę w rozwiązaniu problemu komiwojażera.

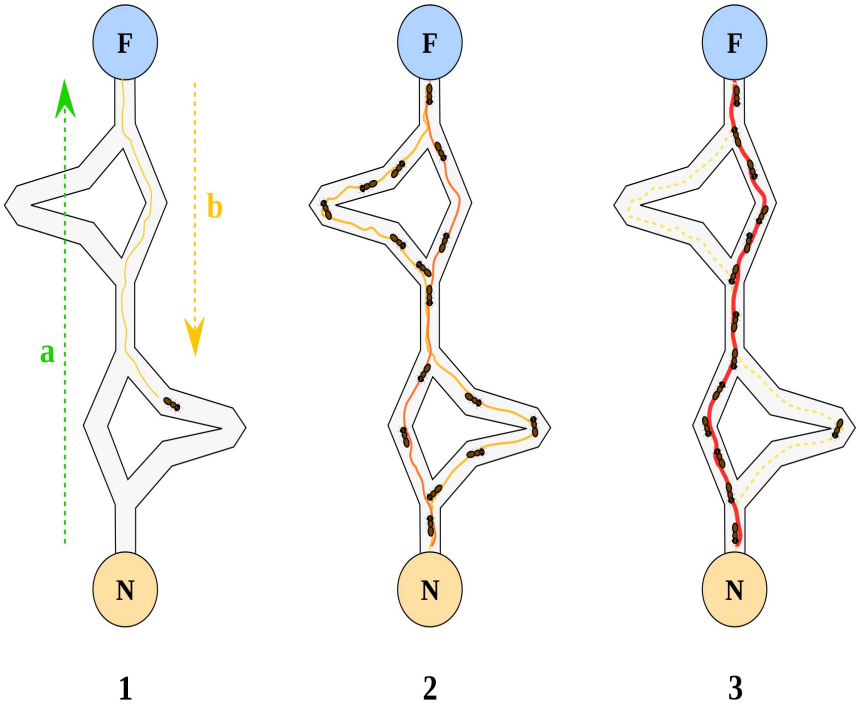
2.3.1. Algorytm mrówkowy

Algorytm zaczerpnięty z zachowania mrówek szukających najkrótszej drogi pomiędzy ich kolonią, a źródłem pożywienia. W świecie naturalnym mrówki wędrują losowo i po znalezieniu pożywienia wracają do swojej kolonii wydzielając feromony. W ten sposób powstają drogi feromonowe, a inne mrówki czując zapach podążają tą ścieżką wzmacniając tym samym gęstość feromonu. Z biegiem czasu ścieżka zaczyna odparowywać co zmniejsza jej atrakcyjność. Im więcej mrówek będzie podróżować i im krótsza to będzie ścieżka, tym więcej czasu feromon będzie potrzebował na odparowanie.

W ten sposób unika się także zbieżności z lokalnym rozwiązaniem, ponieważ jeśli feromon w ogóle by nie odparował, ścieżka byłaby nadmiernie atrakcyjna dla pozostałych mrówek, a więc badanie przestrzeni rozwiązania byłoby ograniczone. Zatem, gdy jedna z mrówek znajdzie tę najkrótszą trasę inne pójdą jej śladem, także wydzielając feromony. W ten sposób natężenie feromonu na optymalnej trasie będzie wzrastać, a na pozostałych drogach maleć i wszystkie mrówki będą przemieszczać się tą najkrótszą trasą. Algorytm ten przedstawiony jest na rysunku 6. Algorytm mrówkowy dla TSP zaproponowany 25 lat temu przez Marco Dorigo cechuje się poniższymi założeniami:

1. Mrówka musi odwiedzić każde miasto dokładnie raz.
2. Odległe miasta mają mniejsze szanse na wybranie, ze względu na ich widoczność.
3. Im bardziej intensywny zapach feromonowy pomiędzy dwoma miastami, tym większe prawdopodobieństwo, że ta krawędź zostanie wybrana.
4. Po zakończeniu podróży mrówka zostawia więcej feromonu na wszystkich przemierzonych krawędziach, jeśli trasa ta jest krótka.
5. Po każdej iteracji następuje odparowanie feromonu.

W problemie komiwojażera algorytm mrówkowy ma przewagę nad symulowanym wyżarzaniem, czy algorytmami genetycznymi, gdyż rozwiązanie może się zmieniać dynamicznie. Metoda ta może działać w sposób ciągły i dostosowywać się do zmian w czasie rzeczywistym. Dzięki temu ma znaczne zastosowanie w routingu sieciowym czy systemach transportu miejskiego. [14]



Rys. 6. Algorytm mrówkowy na przykładzie poszukiwania najkrótszej drogi z gniazda do pożywienia [15]

2.3.2. Algorytm genetyczny

Algorytm genetyczny należy do klasy algorytmów ewolucyjnych. Metoda ta jest powszechnie używa do generowania wysokiej jakości rozwiązań wykorzystując operatory mutacji, krzyżowania i selekcji. Ewolucja rozpoczyna się zazwyczaj od losowo generowanych jednostek i jest procesem iteracyjnym, a populacje w każdej iteracji nazywamy pokoleniem.

Algorytm genetyczny oparty jest na biologicznym środowisku naturalnym odwzorowując rozwój dowolnego istniejącego gatunku. Mamy tutaj do czynienia z krzyżowaniem, czyli rozmnażaniem się osobników. Występuję mutacja, która symuluje wpływ otoczenia na osobnika, na przykład wirus czy choroba zewnętrzna. Ostatnim etapem jest selekcja. W środowisku naturalnym może być to zjadanie osobnika przez gatunki znajdujące się wyżej na liście łańcucha pokarmowego. Ewidentnie zauważalne jest, iż biologia była inspiracją do stworzenia przez twórców algorytmu genetycznego.

Najważniejszym elementem w algorytmie jest pojedynczy osobnik. To z niego składa się cała populacja. Osobnik jest też modyfikowany i poddawany selekcji. Z tego powodu jego struktura, a przede wszystkim długość mają decydujące znaczenie na czas trwania algorytmu. Im bardziej złożona struktura pojedynczego osobnika, tym ten czas rośnie.

Pierwszym krokiem algorytmu genetycznego jest losowanie populacji. Każdy z osobników powinien być generowany całkowicie losowo, aby populacja była w jak największym stopniu różnorodna. By skrócić czas algorytmu populacja nie powinna być zbyt wielka. Z drugiej jednak strony zbyt mała liczba osobników może spowodować zatrzymanie algorytmu w pewnym, bliskim optymalnemu rozwiązaniu, minimum lokalnym.

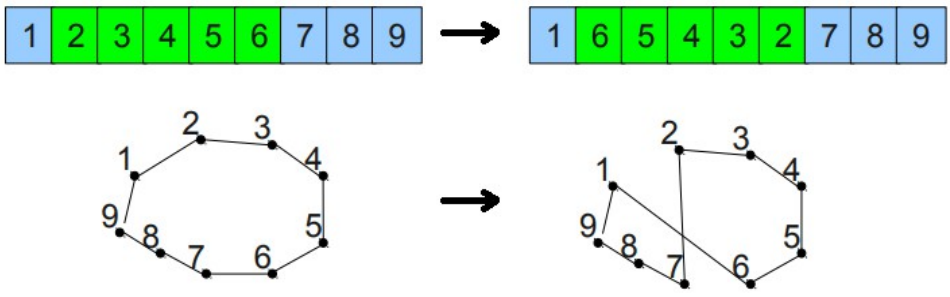
W każdym pokoleniu ocenia się przydatność każdego osobnika w populacji. W przypadku komiwojażera ocena ta jest proporcjonalna do długości trasy, jaką reprezentuje dana jednostka w populacji. Osobniki bardziej dopasowane są losowo wybierane z obecnej populacji, a genom każdego osobnika jest modyfikowany w celu utworzenia nowego pokolenia.

Istnieje kilka metod na wybranie bardziej dopasowanych osobników. Pierwszą z nich jest metoda ruletki. Polega ona na przypisaniu każdemu z osobników pewnej wartości przystosowania, zależnej od jego oceny przydatności w populacji. Następnie, gdy znamy już sumę wartości przystosowania całej populacji możemy obliczyć prawdopodobieństwo wylosowania osobnika do nowego pokolenia. Jest to wartość przystosowania rozważanego osobnika podzielona przez sumę przystosowania wszystkich jednostek. Kolejną metodą selekcji osobników jest ranking liniowy. Należy posortować osobniki względem ich oceny przydatności i nadać im wartość przystosowania, np. 1, 2, 3. Następnie obliczamy prawdopodobieństwo wylosowania osobników identycznie jak w metodzie ruletki. W metodzie rankingu liniowego zmniejszamy szanse wylosowania najlepszego rozwiązania, gdy jest ono dużo lepsze od pozostałych i zwiększamy, kiedy jest minimalnie lepsze. Ostatnią i najbardziej wydajną metodą selekcji jest turniej. Metoda polega na wylosowaniu z populacji tzw. grupy turniejowej składającej się z kilku losowych osobników. Z tej grupy wybieramy najlepiej przystosowanego osobnika i powtarzamy czynność, aż do otrzymania nowej populacji. Metoda turnieju, w przeciwieństwie do pozostałych działa tak samo, niezależnie czy minimalizujemy czy maksymalizujemy funkcję oceny.

Kolejnym etapem algorytmu genetycznego jest krzyżowanie osobników. Standardowe krzyżowanie polega na wymianie materiału genetycznego pomiędzy dwójką osobników będących rodzicami. Krzyżując rodziców pobieramy z każdego z nich część rozwiązania otrzymując potomka. Ten sposób nie daje zadowalających rezultatów, ponieważ dla problemu komiwojażera w otrzymanym rozwiązaniu, co najmniej jeden z wierzchołków

grafu dublowałby się. Lepiej zatem zaprojektować operator krzyżowania opierając się na permutacjach. Dla problemu komiwojażera stworzono kilka innych metod krzyżowania, które w każdym przypadku zwracają dopuszczalne rozwiązanie. Jedną z nich jest PMX (ang. Partially Matched Crossover), krzyżowanie z częściowym odwzorowaniem. Polega ono na wylosowaniu części rozwiązania u obojga rodziców i przekazaniu jej do potomków. Z pierwszego rodzica część genomu kopiowana do potomka drugiego, a z drugiego rodzica do potomka pierwszego. W kolejnej fazie uzupełniana jest pozostała struktura potomków tak, aby nie powstał konflikt. Rozważmy przypadek kopiowania z rodzica pierwszego określonego fragmentu osobnika do potomka drugiego. Dla rodzica nr 2 należy znaleźć elementy w tym samym fragmencie, które nie zostały skopiowane do potomka. Dla każdego takiego elementu trzeba określić, który element został skopiowany z rodzica nr 1 na jego miejsce. Następnie dla tych elementów należy znaleźć ich położenie w genomie rodzica nr 2 i w to miejsce wpisać wcześniej znalezione nie skopiowane elementy rodzica drugiego. Jeśli element z rodzica nr 1 jest już w potomku sprawdzamy, który element został jeszcze skopiowany na jego miejsce i powtarzamy czynność dla tego wierzchołka. Pozostałe elementy kopiujemy bezpośrednio z rodzica nr 2 do potomka. Dla drugiego potomka sytuacja jest identyczna, jedynie zamieniamy rodziców miejscami. Dużo prostszą metodą krzyżowania jest OX (ang. Order Crossover), krzyżowanie z porządkowaniem. Początek z kopiowaniem wcześniej wylosowanego fragmentu rozwiązania jest identyczny jak w poprzedniej metodzie. Jednak w kolejnym etapie, kopiujemy do potomka pierwszego elementy z rodzica drugiego w kolejności, w jakiej występują w osobniku. Oczywiście pomijamy wierzchołki, które zostały już skopiowane we fragmencie z rodzica nr 1.

Po operacji krzyżowania możemy dodatkowo zastosować operator mutacji. W przeciwieństwie do krzyżowania, w algorytmie genetycznym mutacji podlega zazwyczaj jeden z pary osobników. Najprostszą metodą mutacji jest zamiana miejscami dwóch elementów w genomie potomka. Bardziej złożony operator mutacji przedstawiony jest na rysunku 7. Mutacja w chromosomie wybiera 2 losowe wierzchołki i korzystając z inwersji odwraca kolejność pomiędzy nimi. Mutacja ta dokonuje zmiany tylko dwóch krawędzi w grafie, a może w znaczy sposób wpłynąć na jakość otrzymanego rozwiązania.



Rys. 7. Działanie mutacji w algorytmie genetycznym [16]

Po przeprowadzeniu operacji krzyżowania i mutacji otrzymujemy pełne, składające się z potomków pokolenie. To pokolenie jest następnie wykorzystywane w kolejnej iteracji algorytmu. Ponownie można przeprowadzać selekcję osobników i szukać nowych, bardziej optymalnych rozwiązań. Aby algorytm genetyczny nie działał bez końca, należy założyć pewien warunek wyjścia z pętli. Może być to np. określona z góry ilość iteracji, brak poprawy wyniku lub przeciwnie osiągnięcie pewnego zadowalającego rezultatu. [17]

2.4. Znalezienie najkrótszej drogi

W większości przypadków problemu komiwojażera droga z wierzchołka A do wierzchołka B przebiega bezpośrednio po linii prostej. Co jednak w przypadkach, gdzie mamy pewne narzucone ograniczenia i nie możemy tak po prostu wyznaczyć wektora pomiędzy dwoma punkami na układzie współrzędnych. W tego typu problemach należy znaleźć najkrótszą drogę korzystając z dostępnych algorytmów. Do najpopularniejszych z nich należą algorytm Dijkstra i algorytm A* (A gwiazdka). Najczęściej używane są one w protokołach routingu sieciowego lub jako podprogram do innych algorytmów.

2.4.1. Algorytm Dijkstry

Metoda ta służy do znalezienia najkrótszej drogi pomiędzy dwoma wierzchołkami w grafie. Istnieje też wariant algorytmu, który z wierzchołka źródłowego znajduje najkrótsze ścieżki do wszystkich pozostałych wierzchołków. W ten sposób powstaje drzewo najkrótszych ścieżek. Algorytm ten można w łatwy sposób zmodyfikować, dostosowując go do potrzeb użytkownika. Metoda ta rozpatruje każdy wierzchołek i krawędzie łączące go z innymi wierzchołkami. Sprawdza przy tym odległości pomiędzy nimi i biorąc to pod uwagę zapisuje stan badanych wierzchołków. Algorytm zaczyna zawsze od wierzchołków o niższym stanie (mniejszej odległości dojścia do niego), tak by nie pominąć żadnej krótszej trasy. [8]

Działanie metody przedstawiono poniżej:

---

1. Przypisz każdemu wierzchołkowi w grafie wstępną wartość odległości. Dla wierzchołka źródłowego będzie to 0, a dla wszystkich pozostałych nieskończoność.
2. Oznacz wierzchołek źródłowy jako obecnie rozpatrywany. Oznacz wszystkie pozostałe wierzchołki jako nieodwiedzone. Utwórz zestaw składający się z wszystkich nieodwiedzonych wierzchołków.

3. Dla bieżącego wierzchołka należy wziąć pod uwagę wszystkich sąsiadów i obliczyć wstępne odległości. Porównujemy obliczoną odległość z bieżącą wartością i wybieramy mniejszą.
4. Kiedy sprawdzimy wszystkich sąsiadów obecnego wierzchołka, zaznaczamy go jako odwiedzony i usuwamy z zestawu nieodwiedzonych wierzchołków.
5. Algorytm kończy działanie, jeżeli docelowy wierzchołek został oznaczony jako odwiedzony (dla połączenia między dwoma wierzchołkami) lub najmniejsza wstępna odległość pomiędzy wierzchołkami w zestawie nieodwiedzonych wierzchołków jest nieskończonością (dla przypadku z szukaniem drogi do wszystkich wierzchołków, jeżeli nie ma połączenia między wierzchołkiem źródłowym, a którymś z zestawu punktów nieodwiedzonych).
6. Wierzchołek, który ma najmniejszą bieżącą odległość ustaw jako obecnie rozpatrywany i powróć do kroku 3.

2.4.1. Algorytm A*

Kolejnym algorytmem do wyznaczenia najkrótszej drogi pomiędzy dwoma wierzchołkami jest algorytm A*. Posiada on szerokie zastosowanie ze względu na swoją wydajność i dokładność. Algorytm rozwiązuje problem, szukając wśród wszystkich możliwych ścieżek rozwiązania o najmniejszym koszcie. W pierwszej kolejności jednak poszukuje rozwiązania wśród ścieżek, które wydają się najszybciej prowadzić do rozwiązania. Metoda ta zaczynając od konkretnego wierzchołka konstruuje drzewo ścieżek, poszerzając ścieżki z każdym kolejnym krokiem, aż jedna z nich kończy się na wcześniej określonym wierzchołku docelowym. W każdej iteracji A* musi zdecydować, którą ze ścieżek rozszerzyć na jedną lub więcej. Czyni to w oparciu o oszacowanie kosztu.

Algorytm minimalizuje funkcję celu f(x) składającą się z funkcji kosztu g(x) oraz funkcji heurystycznej h(x). Funkcja g(x) stanowi koszt dojścia z wierzchołka źródłowego do bieżącego punktu – x. Natomiast funkcja h(x) określa szacunkowy koszt drogi z punktu x do wierzchołka docelowego. Funkcja h(x) powinna spełniać warunek dopuszczalności i monotoniczności. W pierwszym przypadku funkcja heurystyczna jest nadmiernie optymistyczna. Nie doszacowuje ona kosztu, gdy go minimalizujemy, oraz przeszacowuje, kiedy go maksymalizujemy. Działanie algorytmu A* przedstawiono w kilku poniższych krokach:

---

1. Ścieżka startuje w wierzchołku źródłowym. Długość drogi wynosi 0.

2. Oznaczamy odległości pomiędzy wierzchołkami - g, oraz odległości bezpośrednio między wierzchołkiem docelowym, a wszystkimi pozostałymi - h.
3. Wybieramy jako następny wierzchołek grafu ten, który w najmniejszym stopni powiększy długość naszej drogi, suma g + h.
4. Sprawdzamy czy suma, którejś z dróg nie jest mniejsza od naszej trasy. Jeśli tak przechodzimy do ostatniego wierzchołka tej ścieżki.
5. Powtarzamy krok 3, aż do momentu dotarciu do wierzchołka docelowego.

Algorytm A* można określić słowami „najpierw najlepszy”. Istnieje tutaj zorganizowany model pamięciowy, gwarantujący możliwość odwiedzenia każdego punktu w przestrzeni rozwiązań. Najlepiej metoda ta działa, gdy przestrzeń ta jest przestrzenią drzewiastą. Algorytm A* jest ulepszoną wersją algorytmu Dijksty i obecnie jest znany jako najlepszy algorytm wyszukiwania krótkich ścieżek. [18]

3. Opis aplikacji

3.1. Model sklepu

Aplikacja opisana w tej pracy magisterskiej skupia się na problemie klienta, który robi zakupy w supermarkecie. Ma ona na celu przeprowadzić klienta po sklepie tak, aby zakupił wszystkie wybrane produkty i trafił do wyjścia. Droga, którą porusza się klient, powinna być jak najkrótsza. Jednocześnie jest to droga czasowo optymalna, ponieważ na potrzeby aplikacji na wszystkich trasach w sklepie panuje jednakowy ruch. Zastosowanie programu w prawdziwym życiu może znacznie skrócić czas nie tylko zakupów robionych przez klientów sklepu, ale choćby czas pracowników sklepu podczas wystawiania i sprawdzania towaru.

Aplikacja jest przeznaczona głównie dla dużej powierzchni sklepów, gdzie droga pomiędzy różnymi produktami jest relatywnie długa. Oprócz supermarketów, aplikacja może być pomocna także w magazynach czy innych lokalizacjach, których układ podobny jest do dużych rozmiarów sklepu. Dla wielu przedsiębiorstw program ten z pewnością ułatwiłby pracę i zoptymalizował jej czas.

W odróżnieniu od typowego problemu komiwojażera połączenia między wierzchołkami nie mogą przebiegać zawsze w linii prostej ze względu na regały znajdujące się w sklepie. To ograniczenie komplikuje problem i stanowi główną różnice między sklepem a wyznaczeniem najkrótszej drogi w grafie ważonym. Problem ten jest zbliżony do problemu poruszania się po mieście komunikacją miejską w celu odwiedzenia kilku lokalizacji. W tym przypadku przystanki działają jak skrzyżowania alejek w supermarkecie.

Na potrzeby aplikacji stworzono schemat supermarketu przedstawiony na rysunku 8. W modelu tym możemy wyróżnić kilka charakterystycznych punktów takich jak wejście/wyjście do sklepu (kolor niebieski), 3 różne rodzaje regałów (zielony, żółty, czerwony) oraz przestrzeń, w której klient może się poruszać (kolor szary). Regały występujące w sklepie stanowią ograniczenie na poruszającego się klienta. Inna długość regałów ma na celu urozmaicenie modelu sklepu, a także utrudnienie zastosowanym algorytmom rozwiązanie tego problemu.

Ważnym czynnikiem jest możliwość przemieszczania się klienta. W aplikacji uproszczono to w ten sposób, że klient porusza się tylko wzdłuż całkowitych argumentów osi x lub y. Możemy tak działać, ponieważ regały w sklepie zawsze mają całkowitą długość i szerokość. Dzięki temu wszystkie alejki w sklepie mają szerokość równą 1 jednostce (możemy przyjąć, że klient porusza się środkiem alejki).

Rys. 8. Model sklepu wykorzystany w aplikacji

3.2. Zasada działania aplikacji

Aplikacja ma za zadanie porównanie kilku metod rozwiązania i wybranie tej najbliższej optymalnemu rozwiązaniu. Głównym kryterium jest tu długość przebytej drogi, ale czas działania programu też nie jest bez znaczenia. Podobnie jak dla metody „brute-force", choć dostajemy rozwiązanie optymalne, to czas potrzebny na jego znalezienie jest niewspółmierny co do korzyści z otrzymania lepszego rozwiązania. Z tych właśnie powodów będziemy zwracać uwagę na czas trwania każdego z algorytmów.

Program polega na wybraniu produktów do zakupu z dostępnej listy 66 sklepowych produktów. Dla każdej zaimplementowanej metody pokazana jest ścieżka składająca się z wybranych wcześniej produktów oraz skrzyżowań w sklepie. Punktem początkowym i końcowym jest zawsze wejście sklepowe znajdujące się na układzie współrzędnych w punkcie (1,0). Następnie dla każdej metody wyświetlana jest długość najkrótszej ścieżki oraz czas działania algorytmu. W sklepie znajduje się 25 skrzyżowań. Ich położenie jest wcześniej określone i aby nie pomylić ich z produktami ponumerowane są od 100 w górę. Na samym końcu działania programu na powyższym modelu sklepu rysowane są ścieżki poszczególnych algorytmów. Dzięki układowi współrzędnych w prosty sposób można odczytać pozycje wybranych produktów.

3.2.1. Algorytm zachłanny

Pierwszym i najprostszym algorytmem zastosowanym w aplikacji jest algorytm zachłanny opisany wcześniej w rozdziale drugim. Punktem startowym jest wejście do sklepu, punkt (1,0). Następnie sprawdzamy, który z produktów na liście jest najbliżej obecnej pozycji. Dla wybranego produktu znajdujemy najkrótszą ścieżkę wykorzystując skrzyżowania w sklepie. Z listy produktów usuwamy produkt, w którym obecnie się znajdujemy. Jeśli na liście są jeszcze produkty, ponownie sprawdzamy odległość do pozostałych produktów. Jeśli nie, szukamy najkrótszej drogi do wyjścia. Działanie tego algorytmu przedstawia poniższy pseudokod.

Algorytm zachłanny dla problemu komiwojażera w supermarkecie:

```
T[0] = wejście_sklepowe
i,j,y ← 0
N ← długość P
z ← N − 1

dla każdego i < N - 1
      dla T[i]
            dla każdego j < z
                  jeżeli odległość T[0] → P[j]) < min_dystans to
                        min_dystans = odległość T[0] → P[j]
                        T[i + 1] = P[j]
      dla każdego x < z
            jeżeli T[i + 1] == P[x] to
                  D = Dijkstra (T[i] , P[x])
                  dla każdego y < D
                        T[i + 1] =  D[y]
                        N++
                        i++
                  P[x] = P[z - 1]
                  z--
D = Dijkstra (T[N-1] , T[0])
dla każdego y < D
      T[i + 1] =  D[y]
      N++


Oznaczenia:
P – lista produktów do zakupu
T – najkrótsza droga dla algorytmu zachłannego
Dijkstra(a,b) – zbiór punktów najkrótszej drogi pomiędzy punktami a i b
N – ilość wierzchołków na ścieżce
```

```
i – ilość iteracji
z – ilość pozostałych produktów na liście
```

3.2.2. Algorytm Dijkstry

Do wyznaczenia najkrótszej drogi na zakup wybranych produktów w sklepie nie wystarczy tylko algorytm na rozwiązanie problemu komiwojażera. Trzeba także znaleźć najkrótsze połączenie między poszczególnymi produktami w sklepie. W tym przypadku skorzystano z algorytmu Dijkstry na wyznaczenie najkrótszej drogi między punktami a i b.

Metoda ta na wejściu dostaje punkt początkowy i końcowy oraz zbiór skrzyżowań w sklepie, który jest odpowiednikiem wierzchołków w grafie. Do listy wierzchołka źródłowego algorytm dodaje krawędź skierowaną, po czym zwraca listę sąsiedztwa danego wierzchołka. Dla każdego wierzchołka ustawiana jest największa wartość, jaką może on przechowywać. W przypadku wierzchołka źródłowego jest to 0. Tworzona jest kolejka priorytetowa przechowująca bieżące krawędzie w kolejności od najkrótszych do najdłuższych. Do danej kolejki wstawiany jest wierzchołek źródłowy, od którego zaczynamy poszukiwania. W kolejnym etapie jest przeprowadzana relaksacja grafu, aż do pustej kolejki. Przy przeglądaniu listy sąsiedztwa wszystkich wierzchołków sprawdza się, czy zmiana wierzchołka skróci dystans drogi z wierzchołka źródłowego. Jeżeli krawędź o tej wadze znajduje się już w kolejce priorytetowej usuwamy ją, ponieważ znaleźliśmy lepszą drogę. Ostatecznie wstawiamy nową krawędź z aktualną znalezioną odległością. Działanie algorytmu Dijkstry można prześledzić na podstawie poniższego pseudokodu.

Algorytm Dijksty dla problemu komiwojażera w supermarkecie:

```
dla każdego v
      d[v] = max_wartość
d[z] = 0
do p wstaw nowy D(z , 0)
dopóki (p ! = 0) wykonaj
      dla każdego k
            jeżeli (d[w] >  d[v] + k) to
                  d[w] =  d[v] + k
                  D x = nowy D(w, d[w])
            z p usuń x        //w przypadku braku x nic się nie dzieje
            do p wstaw x

Oznaczenia:
v – wierzchołek
z – wierzchołek źródłowy
```

```
w – wierzchołek docelowy
k - krawędź
graf – graf skierowany reprezentowany za pomocą wierzchołków skierowanych
d – najkrótsze znalezione odległości do danego wierzchołka
p – kolejka priorytetowa przechowująca bieżące krawędzie w kolejności
rosnącej
D – określa dystans do danej krawędzi, używane do przechowywania w kolejce
priorytetowej do wyboru najkrótszej krawędzi
```

Inną możliwą metodą rozwiązania problemu znalezienia najkrótszej ścieżki jest algorytm A*. W przypadku sklepu znając położenie wszystkich wierzchołków na układzie współrzędnych zastosowanie tego algorytmu byłoby optymalnym rozwiązaniem. Jednak ze względu na lepszą znajomość przez autora pracy algorytmu Dijksty, a także z powodu niewielkiej straty na drugorzędnym pod względem optymalizacji programu czasie, zdecydowano się właśnie na ten algorytm.

W aplikacji nieprzydatne są metody wymiany parami. Powodem tego są ograniczenia nałożone na nasz program. Krawędzie w sklepie ze względu na układ regałów rzadko się przecinają, a gdy nawet do tego dochodzi nie możemy tak po prostu zamienić miejscami dwóch przecinających się krawędzi. Spowodowałoby to złamanie innych ograniczeń w sklepie i w konsekwencji złe działanie aplikacji.

3.2.3. Symulowane wyżarzanie

Kolejną metodą wykorzystaną w programie może być natomiast symulowane wyżarzanie. Metoda ta pobiera dwa losowe wierzchołki z aktualnej ścieżki i zamienia je miejscami. Następnie sprawdza, czy droga taka jest bardziej optymalna od bieżącej. Jeśli tak, oznaczamy ją jako najlepsze dotychczasowe rozwiązanie. W przeciwnym przypadku nie pomija jednak badanej drogi, lecz przeprowadza na niej kolejną zamianę miejsc. Dzięki takiemu zabiegowi nie dążymy do osiągnięcia lokalnego minimum. Algorytm ten przeszukuje znacznie szerszą przestrzeń rozwiązań niż choćby algorytm zachłanny. Permutacje produktów są przeprowadzane do momentu, w którym nie widać poprawny w jakości rozwiązania.

Na początku jednak potrzebna nam jest pewna losowa trasa. Otrzymujemy ją wybierając losową kolejność produktów z listy. Wykorzystano do tego generator liczb losowych. Decyduje on o tym, który produkt wybrać jako następny z pozostałych dostępnych na liście. Oczywiście ścieżka zawsze zaczyna i kończy się w tym samym punkcie – wejściu sklepowym. Poniżej widnieje pseudokod pokazujący generowanie tej losowej trasy.

Losowa trasa dla problemu komiwojażera w supermarkecie:

```
T[0] = wejście_sklepowe
N ← długość P
z ← N – 1
i ← 1
y ← 0

dla każdego i < N
      x = generator licz losowych(z)
      dla każdego K
            jeżeli wierzchołek docelowy K == P[x]
                  jeżeli żadna współrzędna punktu startowego i docelowego
                  krawędzi K nie jest sobie równa
                        D = Dijkstra (T[i-1] , P[x])
                        dla każdego y < D
                              T[i] =  D[y]
                              N++
                              i++
                        P[x] = P[z - 1]
                        z--
D = Dijkstra (T[N-1] , T[0])
dla każdego y < D
      T[N] = D[y]
      N++

Oznaczenia:
P – lista produktów do zakupu
T – losowa trasa
K - krawędź
Dijkstra(a,b) – zbiór punktów najkrótszej drogi pomiędzy punktami a i b
N – ilość wierzchołków na ścieżce
i – ilość iteracji
z – ilość pozostałych produktów na liście
x – numer produktu losowany z listy pozostałych produktów
```

Losowa trasa może służyć jako podstawa do kilku innych bardziej złożonych algorytmów. W tym przypadku będzie to opisane poniższym pseudokodem symulowanego wyżarzania. Za każdym uruchomieniem algorytmu dostaniemy inną początkową losową trasę. Dzięki temu, w pewnych testach programu najlepsze rozwiązanie algorytmu otrzymamy szybciej, a w innych wolniej. Jednak sprawdzenie całej przestrzeni rozwiązań zawsze zajmuje tyle samo czasu. Działanie symulowanego wyżarzania można prześledzić na podstawie poniższego pseudokodu.

Algorytm symulowanego wyżarzania dla problemu komiwojażera w supermarkecie:

```
T = losowa trasa
licznik ← 1000
i ← 0
minT = T

dopóki licznik > 1 wykonaj
      najlepszy_wynik = oblicz dystans (minT)
      dla każdego wierzchołka T
            jeżeli wierzchołek a jest produktem
                  akT[i] = a
      nkT = akt
      losowanie dwóch pozycji p1 i p2 z nkT
      zamiana produktów w nkT z p1 i p2 miejscami
      nT[0] = T[0]

      x ← 1
      N ← liczba produktów do zakupu
      dla każdego i < N
            dla każdego K
                  jeżeli wierzchołek docelowy K == nkT[x]
                        jeżeli żadna współrzędna punktu startowego
                        i docelowego krawędzi K nie jest sobie równa
                              D = Dijkstra (nT[i-1] , nkT[x])
                              dla każdego y < D
                                    nT[i] =  D[y]
                                    N++
                                    i++
                              x++
      D = Dijkstra (nT[N-1] , nT[0])
      dla każdego y < D
            nT[N] = D[y]
            N++

      aktualnyDystans = oblicz dystans (T)
      nowyDystans = oblicz dystans (nT)

      jeżeli nowyDystans < najlepszy_wynik
            minT = nT
      T = nT
      licznik = licznik - 1
```

```
Oznaczenia:
T – aktualna droga do modyfikacji dla algorytmu symulowanego wyżarzania
nT – nowa droga rozpatrywana przez algorytm symulowanego wyżarzania
akT – aktualna krótka Trasa (T składająca się z samych produktów)
nkT – nowa krótka Trasa (nT składająca się z samych produktów) 4
minT – najkrótsza aktualnie znaleziona droga przez algorytm
K - krawędź
Dijkstra(a,b) – zbiór punktów najkrótszej drogi pomiędzy punktami a i b
N – ilość wierzchołków na ścieżce
i – ilość iteracji
```

W algorytmie symulowanego wyżarzania zaimplementowano specjalny licznik, który ogranicza liczbę stosowanych permutacji. Dzięki takiemu zabiegowi czas działania algorytmu jest ograniczony, niezależnie od ilości produktów do zakupu. Ponadto licznik ten można powiązać z jakością rozwiązania i zakończyć działanie algorytmu po otrzymaniu zadowalającej trasy.

Algorytm po otrzymaniu losowej trasy uznaje ją za najlepsze tymczasowe rozwiązanie. Następnie tworzy nowe rozwiązanie, w którym dokonuje permutacji dwóch wcześniej wylosowanych produktów. Jeżeli rozwiązanie to jest lepsze od poprzedniego, to ono staję się aktualnie najlepszym wynikiem. W kolejnej iteracji dokonujemy permutacji na najnowszym rozwiązaniu, niezależnie czy jest ono lepsze czy gorsze od najlepszego. Powtarzamy powyższe działania do czasu wyzerowania się naszego, wcześniej ustalonego licznika.

3.2.4. Algorytm genetyczny

Algorytm genetyczny jest najbardziej złożonym algorytmem ze wszystkich użytych w aplikacji. Składa się on z kilku etapów. Na początku tworzona jest pewna populacja złożona z określonej liczby osobników. Każdy z tych osobników stanowi odrębne rozwiązanie problemu komiwojażera. Wykorzystywana jest tutaj przedstawiona wcześniej losowa trasa i generowane są kolejne osobniki. Następnie za pomocą selekcji wybierane są jednostki poddawane krzyżowaniu.

Selekcja przeprowadzana jest metodą turnieju. Z całego pokolenia wybiera się losową grupę jednostek, tzw. grupę turniejową. Z niej do krzyżowania selekcjonowany jest najlepiej przystosowany osobnik. Ocena przystosowania zależy bezpośrednio od jakości rozwiązania danego osobnika. W tym przypadku jest to długość trasy po sklepie. W ten sposób rozwijamy tylko te rozwiązania, które dają najlepsze rezultaty. Po wybraniu określonej liczby osobników przeprowadzane jest krzyżowanie OX – krzyżowanie z porządkowaniem.

Metoda ta opisana jest we wcześniejszym rozdziale pracy. W ten sposób otrzymujemy potomków, którzy stworzą nowe pokolenie osobników. Nim to się stanie, możemy przeprowadzić mutację jednego z potomków otrzymanych po krzyżowaniu. Mutacja wykorzystana w aplikacja to po prostu permutacja dwóch produktów w rozwiązaniu reprezentowanym przez modyfikowanego osobnika. Po zastosowaniu operatorów krzyżowania i mutacji otrzymujemy nowe pokolenie osobników, na którym powtarzamy cały proces w poszukiwaniu lepszego rozwiązania.

Z każdego pokolenia wyszukujemy najlepiej przystosowanego osobnika. Jest to rozwiązanie problemu komiwojażera, którego długość jest najkrótsza. Po każdym utworzeniu nowego pokolenia sprawdzamy, czy nie znaleziono lepszego rozwiązania. Jeżeli po określonej liczbie iteracji, w tym przypadku jest to 100, nie ma poprawy rezultatu przerywamy cykl. Następnie tworzona jest nowa populacja złożona z losowych osobników i cały proces jest powtarzany. Po utworzeniu 10 pokoleń otrzymujemy 10 rozwiązań naszego problemu, po jednym dla każdego pokolenia. Najbardziej optymalne rozwiązanie z nich jest wynikiem algorytmu genetycznego. Poniżej przedstawiono pseudokod prezentujący działanie algorytmu genetycznego.

Algorytm genetyczny dla problemu komiwojażera w supermarkecie:

```
liczba_osobników ← 10
liczba_pokoleń ← 10
licznik_postępu ← 100
progres ← 0
N ← długość P

dla każdego i < liczba_pokoleń
   utwórz pokolenie[liczba_osobników][N]
   utwórz nowe_pokolenie[liczba_osobników][N]

   dla każdego i < liczba_osobników
      pokolenie[i] = losowa trasa
   wykonaj
      licznik_potomków = 0
      wykonaj
         // selekcja turniej
         utwórz osobnik [N]
         pozycja = losowanie z przedziału(0-1)*(długość pokolenie)
         pozycja2 = losowanie z przedziału(0-1)*(długość pokolenie)
         dla każdego pozycja < pozycja2
            minimum = obliczDystans(liczba_osobników[pozycja1])
            pozycja++
         rodzic = minimum
```

```
         // krzyżowanie z porządkowaniem
         rodzicPozycja = losowanie z przedziału(0-1)*(długość rodzic)
         rodzicPozycja2 = losowanie z przedziału(0-1)*(długość rodzic)
         utworz potomek [N+1]

         skopiowaneProdukty = 0
         dla każdego rodzicPozycja < rodzicPozycja2
            potomek[rodzicPozycja] = rodzic[rodzicPozycja]
            skopiowaneProdukty++
         rodzicProdukty = rodzicPozycja2 + 1
         dla każdego x<N+1-skopiowaneProdukty
            jeżeli (rodzicPozycja2+x) % (N+1) równe 0 lub N
               potomek((rodzicPozycja2+x)%(N+1)) = rodzic[0]
            w przeciwnym razie
               wykonaj
                  jeżeli (rodzicProdukty)%(N+1) różne od 0 i N
                     dla każdego a < potomek
                        jeżeli a nie jest puste
                           jeżeli numer a jest równy numerowi
rodzic(rodzicProdukty)%(N+1)
                              rodzicProdukty++
                     potomek(rodzicPozycja2+x)%(N+1) =
rodzic(rodzicProdukty)%(N+1)
                     rodzicProdukty++
                  w przeciwnym razie jeżeli (rodzicProdukty)%(N+1)==0 lub N
                     rodzicProdukty++
               dopóki (potomek((rodzicPozycja2+x)%(N+1)) jest puste)
         //mutacja identyczna jak w przypadku symulowanego wyżarzania
         licznik_potomków += 2
      dopóki (licznik_potomków < liczba_osobników)
      dla każdego i <  liczba_osobników
         pokolenie[i] = pusty
         pokolenie[i] = nowe_pokolenie[i]
      progres++
   dopóki (progres < licznik_postępu)

Oznaczenia:
P – lista produktów do zakupu
N – ilość wierzchołków na ścieżce
rodzic – osobnik z pokolenia wybrany do krzyżowania
potomek – osobnik powstały po krzyżowaniu
progres – liczba iteracji bez poprawy rozwiązania
```

4. Przeprowadzone testy

W tym rozdziale przetestowane zostaną wszystkie zaimplementowane wcześniej algorytmy. Za pomocą zebranych wyników będzie można ocenić jakość zastosowanych metod i ich efektywność względem parametru długości drogi, a także czasu trwania algorytmów.

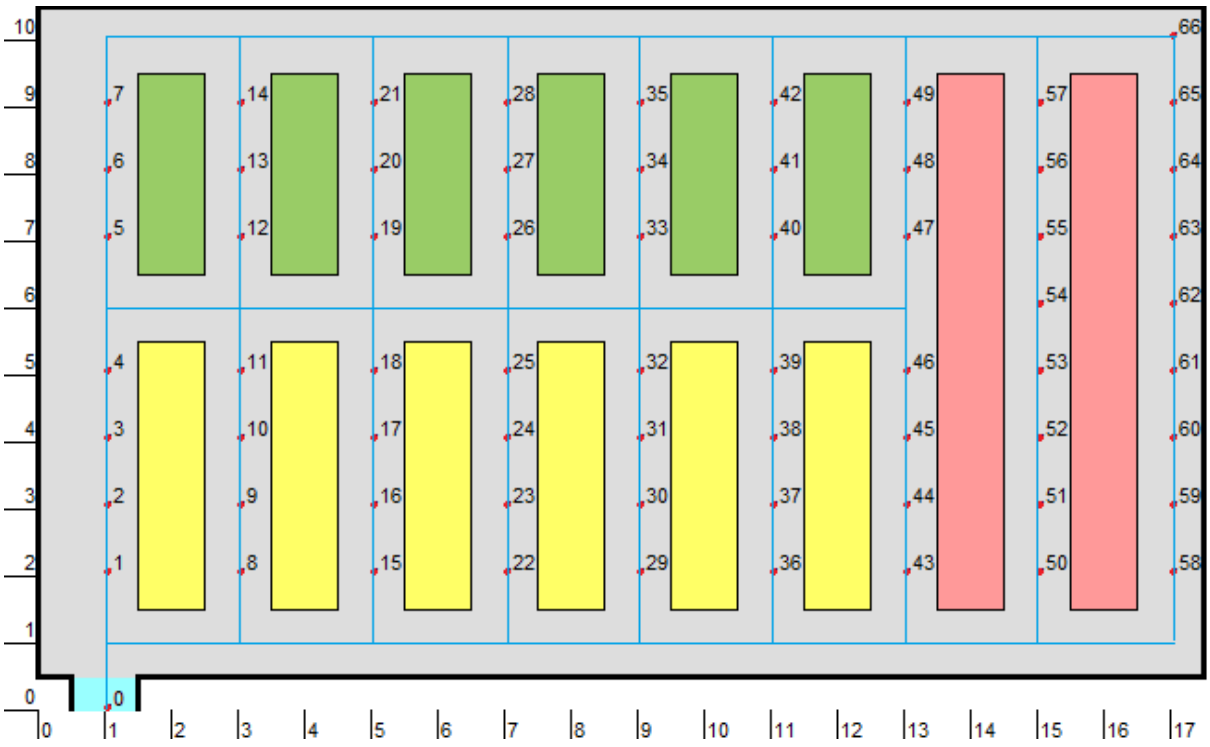
4.1. Dane testowe

Przyjrzyjmy się danym wykorzystanym do przeprowadzania testów na 3 wykorzystanych algorytmach. W sklepie znajduje się 66 produktów do zakupu ponumerowanych od 1 do 66. Każdy z nich ma swoje miejsce w sklepie określone za pomocą dwóch zmiennych – współrzędnej x oraz y. Pierwsza współrzędna tak jak w układzie współrzędnych oznacza długość, natomiast druga szerokość. Listę produktów przedstawiono w tabeli 1.

produkt	x	y	produkt	x	y	produkt	x	y	produkt	x	y	produkt	x	y
1	1	2	15	5	2	29	9	2	43	13	2	58	17	2
2	1	3	16	5	3	30	9	3	44	13	3	59	17	3
3	1	4	17	5	4	31	9	4	45	13	4	60	17	4
4	1	5	18	5	5	32	9	5	46	13	5	61	17	5
5	1	7	19	5	7	33	9	7	47	13	7	62	17	6
6	1	8	20	5	8	34	9	8	48	13	8	63	17	7
7	1	9	21	5	9	35	9	9	49	13	9	64	17	8
8	3	2	22	7	2	36	11	2	50	15	2	65	17	9
9	3	3	23	7	3	37	11	3	51	15	3	66	17	10
10	3	4	24	7	4	38	11	4	52	15	4			
11	3	5	25	7	5	39	11	5	53	15	5			
12	3	7	26	7	7	40	11	7	54	15	6			
13	3	8	27	7	8	41	11	8	55	15	7			
14	3	9	28	7	9	42	11	9	56	15	8			
									57	15	9			

Tab. 1. Baza produktów wykorzystana w aplikacji

W tabeli produktów można zauważyć pewną zależność. Produkty posiadają jedynie nieparzyste współrzędne x oddalone od siebie zawsze o 2 jednostki. Jest to szerokość alejek w sklepie. Każda z nich posiada po 7 produktów, prócz ostatnich dwóch alejek. Produkty w alejkach są oddalone na szerokość o 1 jednostkę. Wyjątkiem jest współrzędna y równa 6, gdzie przebiega duża alejka poprzeczna. Usytuowanie owych produktów na mapie sklepu obrazuje rysunek 9.



Rys. 9. Położenie produktów względem modelu sklepu

Produkty zaznaczone są czerwony kropkami i oznaczone od 1 do 66. Punktem 0 na rysunku 9 jest wejście do sklepu. Z niego użytkownik robiący zakupu startuje i do niego wraca na końcu. Wejście jest pierwszym i ostatnim wierzchołkiem rozwiązania problemu komiwojażera. Dla uproszczenia problemu przyjęto, że produkty znajdują się nie na regałach, lecz na drodze poruszania się konsumenta po sklepie. Droga ta wyznaczona jest przez niebieską linię na rysunku 9.

ilość	produkty
5	40, 41, 59, 15,34
10	30, 59, 50, 41, 3, 31, 21, 17, 11, 49
15	56, 35, 58, 15, 60, 3, 63, 20, 57, 40, 39, 32, 17, 61, 36
20	28, 38, 43, 37, 12, 36, 26, 59, 3, 46, 24, 32, 13, 18, 55, 64, 25, 14, 6, 42
25	8, 5, 33, 25, 50, 16, 49, 61, 24, 42, 2, 52, 32, 3, 1, 47, 36, 48, 14, 27, 46, 12, 55, 3 1, 7
30	33, 47, 19, 21, 51, 61, 35, 13, 3, 37, 65, 54, 55, 38, 66, 58, 42, 12, 49, 59, 24, 36, 25, 22, 60, 32, 18, 10, 20, 4
35	48, 39, 65, 28, 13, 42, 35, 23, 64, 32, 30, 9, 44, 38, 27, 1,5, 50, 17, 34, 33, 12, 63,3, 20, 25, 19, 60, 49, 10, 47, 46, 55, 16, 52
40	63, 5, 15, 20, 42, 2, 48, 40, 19, 50, 8, 1, 23, 14, 52, 45, 24, 37, 25, 51, 16, 9, 46, 4 4,39, 64, 58, 4, 65, 13, 56, 36, 27, 21, 11, 7, 61, 35, 59, 62

Tab. 2. Listy produktów użyte do testów

Jako dane wejściowe do testów przyjęto kilka zestawów produktów, każdy z różną ich ilością. Listy produktów do testów zostały wybrane w sposób losowy. Tabela 2 przedstawia używane w testach dane wejściowe.

4.2. Testy praktyczne

W pracy zaimplementowane zostały 3 różne metody rozwiązywania problemu komiwojażera. Stopień ich złożoności także jest inny dla każdego algorytmu. Algorytm zachłanny znajduje tylko jedno rozwiązanie. Symulowane wyżarzanie przeprowadza szereg permutacji, aby zwiększyć przestrzeń poszukiwań. Najbardziej złożoną metodą jest jednak algorytm genetyczny wykorzystujący różne operatory do modyfikacji rozwiązania. Generuje on kilka populacji rozwiązań, z których każda modyfikuje to rozwiązanie setki razy. Czy jednak najbardziej złożony algorytm oznacza, że jest to także najlepsza metoda? Na to i kilka innych pytań pomogą nam odpowiedzieć przeprowadzone testy. Przedstawione wcześniej algorytmy zostaną sprawdzone między innymi pod względem optymalnego rozwiązania, powtarzalności otrzymanych wyników, wpływu wielkości danych wejściowych na rozwiązanie czy czasu działania algorytmów.

4.2.1. Test algorytmu genetycznego

Nim przejdziemy do porównywania wszystkich 3 algorytmów należy popracować nieco nad algorytmem genetycznym. Musimy dostosować parametry algorytmu tak, aby osiągał jak najlepsze wyniki w jak najkrótszym czasie. Do ustawienia mamy takie parametry jak:
- liczba osobników w pokoleniu
- ilość populacji
- rozmiar progresu (ilość iteracji po których algorytm zostaje zatrzymany ze względu na brak poprawy rezultatu)

Przeprowadźmy kilka testów dla różnego rozmiaru danych wejściowych. Zmieniając wyżej wymienione parametry możemy sprawdzić, który układ daje najlepsze rezultaty. Po kilku testach szybko uznano, że liczba osobników i liczba populacji nie mogą przekroczyć 10. Powoduje to znaczne wydłużenie działania algorytmu lub brak pamięci, zwłaszcza dla większej grupy danych wejściowych. Poniżej przedstawiono tabelę z wynikami przeprowadzonych testów. Produkty do testów zostały pobrane z wcześniej wygenerowanej listy, patrz tabela 2.

Liczba osobników	Liczba populacji	Progres	Produkty 10	Czas [s]	Produkty 20	Czas [s]	Produkty 30	Czas [s]
6	5	20	73,84	3	122,11	28	179,43	69
10	5	20	71,88	11	113,46	69	141,62	622
6	10	20	71,88	11	111,40	37	203,14	466
10	10	20	73,84	20	111,12	108	brak pamięci	
6	5	50	71,88	15	111,34	59	165,23	191
10	5	50	71,88	17	101,18	121	brak pamięci	
6	10	50	71,88	20	113,07	107	brak pamięci	
10	10	50	71,88	48	107,34	187	brak pamięci	

Tab. 3. Wyniki testów dla algorytmu genetycznego

Z tabeli wyników widać, iż dla mniejszej liczby produktów parametry algorytmu genetycznego nie mają dużego wpływu na rozwiązanie. Im liczba produktów większa tym dobór parametrów ma większy wpływ na wynik. Natomiast odbija się to czasem działania algorytmu i potrzebną pamięcią. Jednym słowem nie ma złotego środka na wybór parametrów. Najlepszym wyborem byłoby dostosowanie ich w zależności od liczy produktów do zakupu. Zdecydowano się jednak na parametry 6, 5, 50.

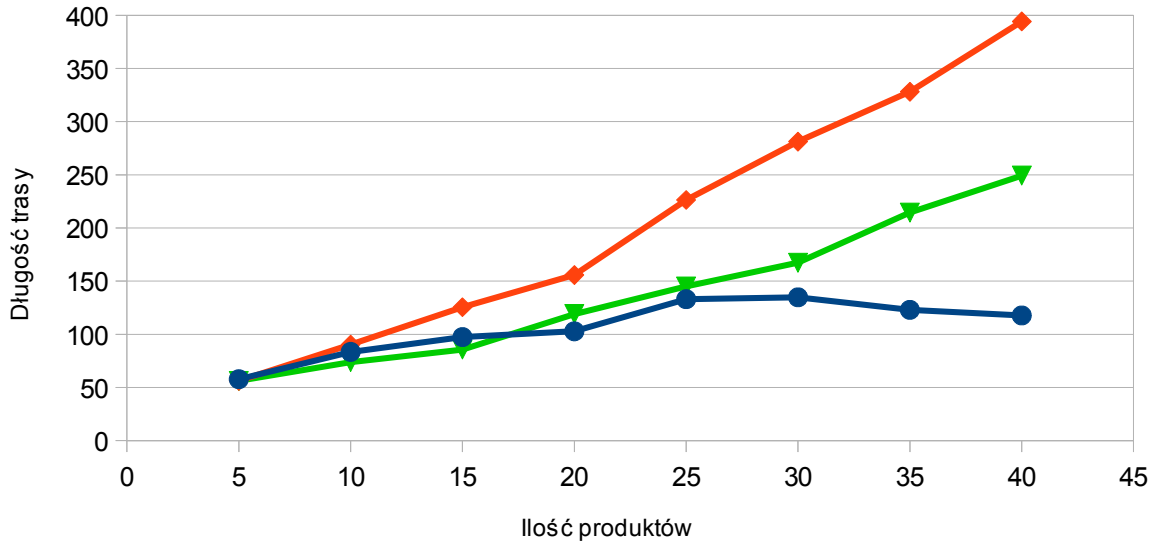
4.2.1. Optymalizacja drogi i czasu

Kolejnym testem będzie rozwiązanie kluczowego zagadnienia tej pracy magisterskiej. Znalezienie najkrótszej drogi po supermarkecie w poszukiwania określonej liczby produktów. Test przeprowadzony zostanie dla wszystkich 3 metod. Liczba produktów będzie się zmieniać. W ten sposób będziemy w stanie sprawdzić zależność ilości danych wejściowych na jakość końcowego rozwiązania. Listy produktów przedstawiono w tabeli 2.

Produkty	Zachłanny Wynik	Zachłanny Czas [s]	Sym. wyżarzanie Wynik	Sym. wyżarzanie Czas [s]	Genetyczny (6,5,20) Wynik	Genetyczny (6,5,20) Czas [s]
5	57,80	< 1	56,16	2	56,16	1
10	83,47	< 1	90,28	4	73,80	9
15	97,40	< 1	125,53	8	85,64	16
20	102,84	< 1	155,75	13	119,10	38
25	133,08	< 1	226,36	20	145,06	70
30	134,84	< 1	281,23	27	167,30	117
35	123,02	< 1	328,08	39	214,53	413
40	117,75	< 1	394,25	54	249,26	1022

Tab. 4. Porównanie 3 zastosowanych algorytmów

Na pierwszy rzut oka widać, że wszystkie algorytmy zwracają prawidłowe wyniki w skończonym czasie. Algorytm zachłanny niezależnie od długości produktów oblicza trasę w bardzo krótkim czasie. W przeprowadzonym teście symulowane wyżarzanie zabiera nieco więcej czasu, lecz nawet dla 40 produktów jest to czas zadowalający. Natomiast testowany jako ostatni, algorytm genetyczny, generuje rozwiązanie problemu w zależności od ilości produktów w stosunku niemal wykładniczym. O ile do 30 produktów trwa to w przyzwoitym czasie, tak dla większej ilości czas rzędu kilku do kilkunastu minut nie przekonuje do wyboru algorytmu genetycznego.



Rys. 10. Porównanie algorytmów dla różnych list produktów (niebieski – algorytm zachłanny, czerwony – symulowane wyżarzanie, zielony - algorytm genetyczny)

Przejdźmy jednak do porównania najważniejszej statystyki, czyli długości obliczonej drogi. Dla mniejszej ilości produktów wyniki 3 metod są bardzo do siebie zbliżone. Wraz ze wzrostem ilości danych coraz łatwiej jest zauważyć rozbieżności w wynikach, co obrazuje rysunek 10. Do 25 produktów algorytmy zachłanny i genetyczny niemal się pokrywają. Różnice widać dopiero w późniejszych testach. Natomiast jeśli chodzi o symulowane wyżarzanie niemal od początku widać przewagę pozostałych metod nad tym algorytmem. Biorąc pod uwagę najlepsze rozwiązanie i potrzebny do jego osiągnięcia czas, bezdyskusyjnie test ten wygrywa algorytm zachłanny.

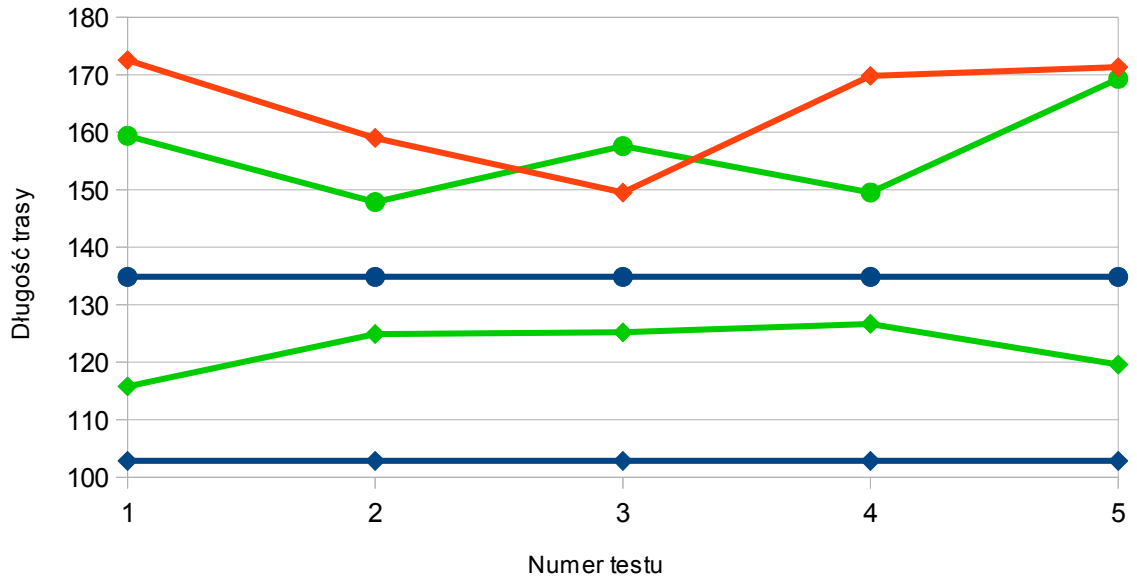
4.2.3. Test powtarzalności

W tym teście sprawdzona zostanie powtarzalność każdego z algorytmów. Nie wiemy, czy otrzymane wyniki są miarodajne. Aby to sprawdzić uruchomimy każdy z algorytmów kilka razy i ocenimy ewentualną rozbieżność wyników. Jako dane wejściowe przyjęto ilość 20 i 30 produktów.

Długość trasy		Zachłanny	Sym. wyżarzanie	Genetyczny (6,5,20)
20		102,84	172,54	115,80
		102,84	159,01	124,90
		102,84	149,51	125,19
		102,84	169,81	126,67
		102,84	171,32	119,58
30		134,84	274,79	159,38
		134,84	283,32	147,86
		134,84	263,12	157,59
		134,84	279,29	149,52
		134,84	264,82	169,29

Tab. 5. Wyniku algorytmów dla testu powtarzalności

Przeglądając wyniki widać, iż algorytm zachłanny jako jedyny zwraca w każdym powtórzeniu identyczny rezultat. Chcąc lepiej zobrazować różnicę wyniku sporządzono wykres (rysunek 11). Testy symulowanego wyżarzania dla 30 produktów zostały pominięte ze względu na lepszą przejrzystość wykresu. Dokonano również obliczenia błędu bezwzględnego i względnego dla 2 niestabilnych algorytmów. Dla symulowanego wyżarzania wyniosły one odpowiednio dla 20 produktów 8,10 i 4,93% oraz dla 30 - 10,25 i 3,75%. Natomiast w przypadku algorytmu genetycznego błąd ten był nieco większy, konkretnie dla 20 produktów 6,63 i 5,41% oraz dla 30 – 12,56 i 8,02%.



Rys. 11. Porównanie algorytmów dla testu powtarzalności wyniku
(romby – test dla 20 produktów, kółka – dla 30, reszta oznaczeń jak dla rys. nr 10)

Do przeprowadzenia obliczeń skorzystano z poniższych wzorów:

$$\Delta x = x - x_0$$

$$\delta = \frac{\Delta x}{x} * 100\,\%$$

gdzie: Δx – błąd bezwzględny,

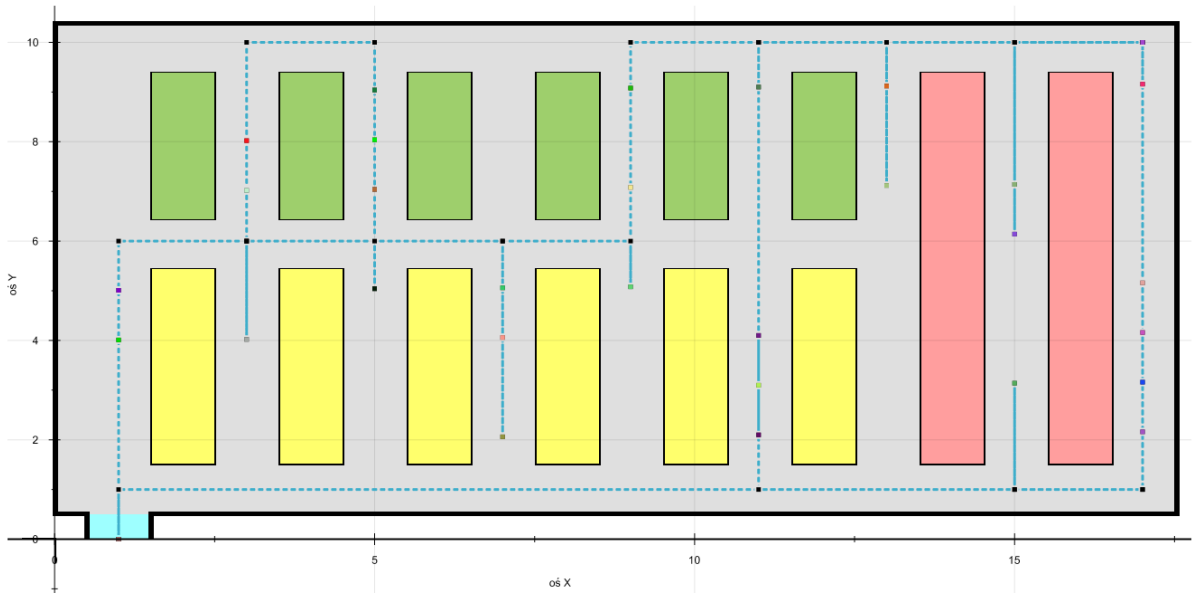
x – dokładna wartość (średnia arytmetyczny otrzymanych wyników),

x_0 – zmierzona wartość (maksymalna odchyłka od dokładnej wartości),

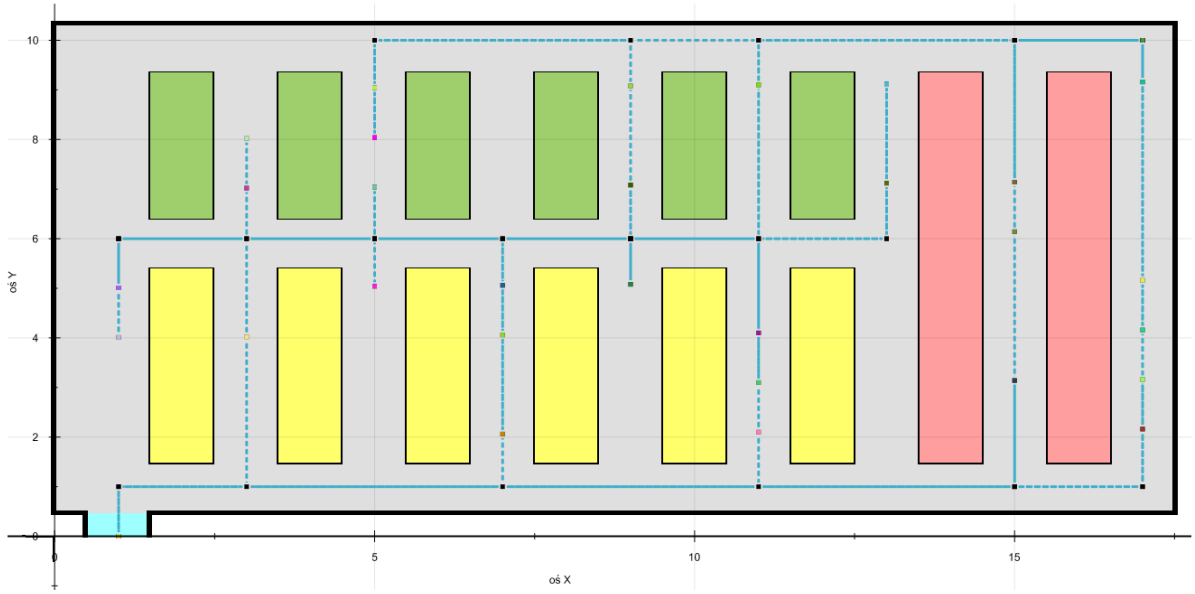
δ – błąd względny.

4.2.3. Rozwiązanie graficzne

W ostatnim podrozdziale przedstawione zostaną graficznie wyniku omawianych wcześniej algorytmów. Jako dane wejściowe przyjęto listę 30 produktów do zakupu.

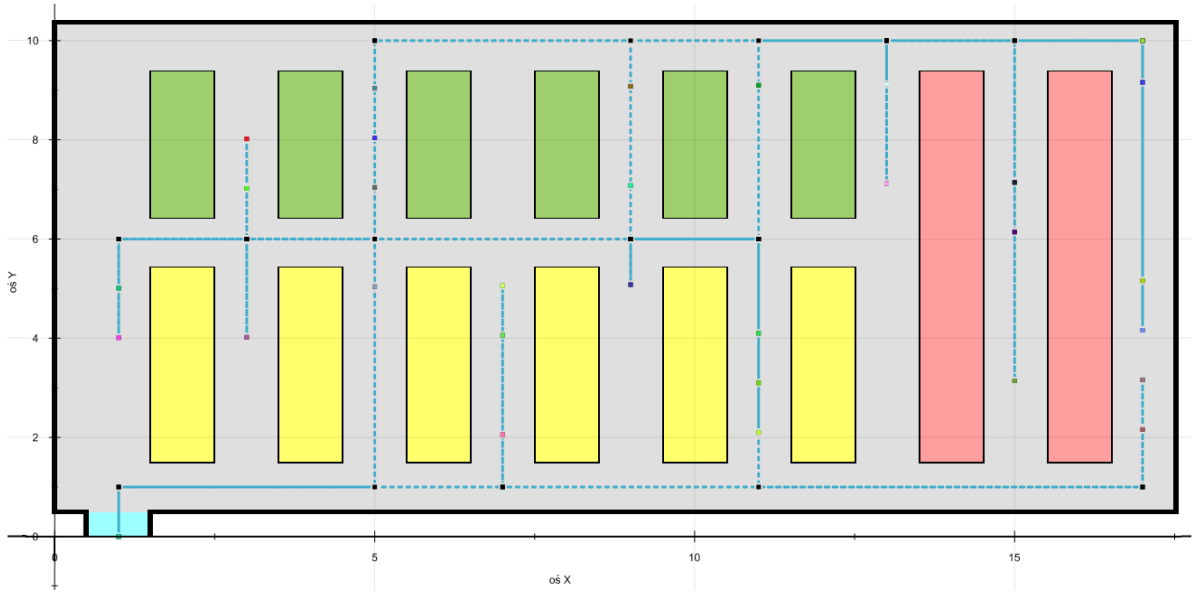


Rys. 12. Graficzne rozwiązanie algorytmu zachłannego



Rys. 13. Graficzne rozwiązanie symulowanego wyżarzania

Na rysunkach 12, 13 i 14 pokazane są kolejno algorytm zachłanny z drogą równą 134,84, symulowane wyżarzanie z wynikiem 261,32 oraz algorytm genetyczny z rozwiązaniem wynoszącym 170,98. Kolor niebieski to droga poruszania się klienta po sklepie. Lokalizacja produktów oznaczona jest różnokolorowymi kwadratami, natomiast czarne kwadraty reprezentują wykorzystane skrzyżowania w sklepie.



Rys. 14. Graficzne rozwiązanie algorytmu genetycznego

Podsumowanie

Celem pracy dyplomowej było stworzenie aplikacji do wyznaczenia najkrótszej drogi w supermarkecie, aby kupić określoną liczbę produktów. Cel ten udało się zrealizować korzystając z języka programowania JAVA. Do realizacji wykorzystano trzy różne metody rozwiązania problemu komiwojażera: algorytm zachłanny, symulowane wyżarzanie oraz algorytm genetyczny. Znaczne ułatwienie w pisaniu pracy dyplomowej miały zajęcia z metod optymalizacji na kierunku Automatyka i Robotyka. W tworzeniu aplikacji pomogły także wcześniej realizowane projekty studenckie na temat optymalizacji.

Głównym zadaniem aplikacji było rozwiązaniem problemu komiwojażera. Na starcie użytkownik określał, jakie towary z puli 66 dostępnych produktów chce zakupić. Ze względu na złożoność obliczeniową problemu nie ma możliwości rozwiązania go w sposób jednoznaczny. Po wyborze towarów, każdy z trzech algorytmów wyznaczał w określonym czasie trasę najbliższą optymalnej. Trasa rozpoczynała się i kończyła zawsze w tym samym punkcie – wejściu do sklepu. Połączenia pomiędzy poszczególnymi produktami obliczane zostały metodą najkrótszej ścieżki z punktu A do B.

W pierwszej części pracy dyplomowej przybliżono problem komiwojażera, jego strukturę, zasadę działania oraz mnogość zastosowań. Zwrócono uwagę na złożoność obliczeniową zagadnienia dla większej liczby danych. Następnie przedstawiono kilka metod, które przybliżają lub też poszukują rozwiązania optymalnego problemu. Zaczynając od metod heurystycznych, poprzez algorytmy aproksymacyjne, aż po inteligentne. Ze względu na pewne ograniczenia nie można było zastosować niektórych metod.

W kolejnej części wprowadzono czytelnika w zasadę działania tworzonej aplikacji. Opisano dokładnie każdy algorytm zawarty w projekcie. Uwzględniono również algorytm Dijkstry i wyznaczenie losowej trasy, które były potrzebne do wyznaczenia rezultatów w trzech głównych algorytmach. Zaprezentowano graficznie model sklepu, na podstawie którego testowano metody. Ostatecznie porównano wymienione wcześniej trzy metody głównie pod względem jakości rozwiązania. Zwrócono jednak uwagę na inne aspekty, między innymi na czas potrzebny do wyznaczenia wyników.

Do porównania przejęto takie parametry algorytmów, aby nie dochodziło do sytuacji braku pamięci czy zbyt długiego czasu obliczeń. Porównując rozwiązania metod można zauważyć, iż rezultaty algorytmów zachłannego i genetycznego są bardzo zbliżone. Dopiero dla ilości danych powyżej 25 przewagę ma ten pierwszy algorytm. Symulowane wyżarzanie natomiast niezależnie od ilości produktów zwraca znacznie gorsze wyniki od pozostałych dwóch metod. O ile dla niewielkiej liczy towarów wszystkie rozwiązania są bliskie optymalnemu, tak wraz ze wzrostem ilości produktów coraz bardziej one od niego odbiegają.

Biorąc pod uwagę czas potrzebny do wyznaczenia rozwiązania algorytm zachłanny deklasuje konkurencje. Dzięki temu, że potrzebuje tylko jednej iteracji w każdym teście zwraca wynik poniżej 1 sekundy. W przeciwieństwie do niego symulowane wyżarzanie potrzebuje zazwyczaj kilkunastu sekund, natomiast algorytm genetyczny w przypadkach większej ilości produktów nawet kilku minut.

Symulowane wyżarzanie i algorytm genetyczny są ponadto obarczone rozbieżnościami pomiarowymi. Ich wyniki mogą się różnić w zależności od powtórzenia testu nawet o kilka procent. Niemniej jednak algorytmu zachłannego jako jedynego z tej trójki nie można ulepszyć. W pozostałych metodach zmieniając parametry takie jak ilość iteracji czy wielkość populacji można by usprawnić algorytm. Jednak na przeszkodzie stoi brak pamięci komputera i długi czas obliczeniowy.

Jedną z możliwości jest uruchomienie programu na potężnym superkomputerze, takim jak np. znajdujący się na terenie AGH „Prometheus”. Zapobiegnie to braku pamięci komputera i skróci czas obliczeniowy algorytmów. Dla wyższych parametrów oba algorytmy dadzą na pewno lepsze rezultaty. Choć algorytm genetyczny nie będzie szybszy od zachłannego, ma on duże szanse, aby zwracać lepsze rozwiązania. Inną możliwością byłoby dodanie kolejnego, być może sprawniejszego algorytmu.

Ta praca dyplomowa jest dla mnie inspiracją do dalszych badań nad jej praktycznym wykorzystaniem. Aplikacja ma na celu ułatwienie życia codziennego zwykłym użytkownikom. Mam nadzieje, że będzie ona początkiem prac do przyszłego wdrożenia tego pomysłu w życie nie tylko dla sklepów, ale też na przykład dużych halach targowych czy magazynów.

Spis ilustracji

Literatura

[1] Maciej Marek Sysło, Narsingh Deo, Janusz S. Kowalik „Algorytmy optymalizacji dyskretnej”, Wydawnictwo Naukowe PWN, Warszawa, 1995;

[2] Mateusz Łyczek, „Metoda podziału i ograniczeń”, Wrocław, 2011;

[3] http://www.math.uwaterloo.ca/tsp/index.html;

[4] https://pl.wikipedia.org/wiki/Przestrzeń_metryczna;

[5] Richard E. Ladner, „On the structure of polynomial time reducibility”, Journal of the ACM, 1975;

[6] Antoni Niederliński, „Programowanie w logice z ograniczeniami”, Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice, 2014;

[7] https://en.wikipedia.org/wiki/Kruskal_algorithm;

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, „Wprowadzenie do algorytmów”, Wydawnictwo Naukowo-Techniczne, Warszawa, 2007;

[9] G. A. Croes, „A method for solving traveling salesman problems”, Shell Development Company, Houston, 1958;

[10] https://en.wikipedia.org/wiki/2-opt;

[11] A. Das, B. K. Chakrabarti, „Quantum Annealing and Related Optimization Methods”, Springer, Heidelberg, 2005;

[12] Fred Glover „Tabu Search”. ORSA Journal on Computing, 1989;

[13] https://en.wikipedia.org/wiki/Christofides_algorithm;

[14] M.Dorigo, 1992. „Optimization, Learning and Natural Algorithms”, Włochy, 1992;

[15] M.Duran Toksari, „A hybrid algorithm of Ant Colony Optimization (ACO) and Iterated Local Search (ILS) for estimating electricity domestic consumption”, 2015;

[16] http://aragorn.pb.bialystok.pl/~wkwedlo/EA5.pdf;

[17] Zbigniew Michalewicz, „Algorytmy genetyczne + struktury danych = programy ewolucyjne”, Wydawnictwa Naukowo-Techniczne, 1996;

[18] Ivan Bratko, „Prolog programming for artificial intelligence” Harlow, 2001;