AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA AUTOMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa magisterska

Algorytmy przybliżone dla zagadnienia przydziału kwadratowego
Approximation algorithms for qadratic assignment problem

Autor: *Stefan Kultys*
Kierunek studiów: *Automatyka i Robotyka*
Opiekun pracy: *dr inż. Wojciech Chmiel*

Kraków, 2014

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdeczne podziękowania

Spis treści

1. Wstęp

Nadejście rewolucji przemysłowej spowodowało powstanie wielkiej liczby firm, przedsiębiorstw, które były dużo większe niż znane wcześniej zakłady rzemieślnicze. Ich rozmiar powodował również rozrost złożoności problemów związanych z organizacją tychże firm. W związku z kompleksowością pojawiły się problemy jak najlepszego przydziału dostępnych zasobów oraz najwłaściwszej organizacji pracy. Zaistniała więc potrzeba stworzenia różnych metod, dzięki którym można by powyższe problemy w jakiś sposób rozwiązać. Ta potrzeba doprowadziła do powstania badań operacyjnych.

Chociaż początki badań operacyjnych faktycznie związane są z rewolucją przemysłową, to jednak pojęcie badań operacyjnych, które znamy obecnie, związane jest z działaniami podejmowanymi przez agencje wojskowe już na początku drugiej wojny światowej. Można by stwierdzić, że pewną ironią losu jest fakt, iż wiele wykorzystywanych dzisiaj odkryć i wynalazków, które bardzo ułatwiają nam codziennie życie, zostało powołanych do życia w związku z działaniami kojarzącymi się najczęściej z cierpieniem i przemocą. Brytyjskie i amerykańskie organizacje wojskowe zatrudniły ogromną liczbę naukowców, by ci wdrożyli naukowe podejście do spraw związanych z efektywnym zbrojeniem się, zarządzaniem zasobami oraz taktycznymi i strategicznymi problemami związanymi z prowadzeniem działań wojennych.

Mówi się, że podjęte wysiłki miały duży wpływ na takie znane wydarzenia jak Bitwa o Anglię, czy też Bitwa o Atlantyk.

Sukces, jaki odniosły badania operacyjne w wojskowości, zachęcił ludzi związanych z przemysłem do zaadaptowania ich również w samym przemyśle. Ożywienie w gospodarce, spowodowane zakończeniem wojny, doprowadziło do wzrostu złożoności działalności firm, a więc badania operacyjne idealnie nadawały się jako narzędzie wspierające organizację i zarządzanie tymi przedsiębiorstwami.

Niewątpliwie następujący szybki rozwój badań operacyjnych miał swą przyczynę w tym, że wielu naukowców, którzy parali się nimi podczas wojny, szukając pracy w swojej branży, chętnie zajęło się dalszymi studiami nad badaniami operacyjnymi w dziedzinach związanych nie tylko z wojskowością. Oczywiście nie oznaczało to, że wojsko całkowicie zrezygnowało z badań operacyjnych. Również postęp związany z powstaniem komputerów dał odpowiednie narzędzia do analizy coraz bardziej złożonych problemów. Wiele problemów związanych z podejmowaniem decyzji, wyborem najlepszego rozwiązania można było rozwiązać podpierając się matematycznym modelem. Mając więc problem w sformalizowanej postaci można zaproponować algorytm, który rozwiąże dane zagadnienie. Sam algorytm jako ciąg kolejnych instrukcji, które należy wykonać, by osiągnąć dany cel, bardzo dobrze nadaje się do

zaimplementowania i wykonania na komputerze. Coraz szybsze komputery o coraz pojemniejszych pamięciach, a także wykorzystanie technik programowania równoległego i współbieżnego pozwalają na rozwiązywanie coraz bardziej złożonych problemów w rozsądnym czasie. Z czasem więc zaczęły się pojawiać kolejne algorytmy, ale też nowe problemy. Również dokonywane odkrycia naukowe pozwoliły na wykorzystanie występujących w naturze procesów do tworzenia nowatorskich metod rozwiązywania skomplikowanych zagadnień.

Niestety, istnieje wiele problemów, w przypadku których można jedynie powiedzieć, że mają optymalne rozwiązanie, nie da się jednak znaleźć go przy wykorzystaniu obecnie dostępnej technologii. Poprzez oszacowanie złożoności obliczeniowej algorytmów można tylko stwierdzić, że potrzebny czas do znalezienia rozwiązania problemu przy ich wykorzystaniu jest niejednokrotnie dłuższy niż przeciętny czas życia człowieka. Przykładem takiego zagadnienia jest tzw. problem przydziału kwadratowego, polegającego na przydziale pewnej liczby placówek do takiej samej liczby miejsc. Wynika z tego, że dla n placówek możliwe jest w sumie *n!* wszystkich permutacji. Wraz ze wzrostem liczby placówek, które należy przydzielić, ilość możliwych rozwiązań rośnie bardzo szybko. Już dla stosunkowo małej ilości placówek możliwa jest ogromna liczba rozwiązań. Istnieje więc wiele algorytmów przybliżonych, inaczej nazywanych aproksymacyjnych, które znajdują jedynie przybliżone rozwiązanie postawionego problemu. Nie oznacza to jednak, że zwrócone przez algorytm rozwiązanie nie może być faktycznie optymalne, ale przeważnie nie da się tego sprawdzić.

Jak już zostało to nadmienione wyżej, istnieje wiele algorytmów wykorzystujących analogie do zachowań występujących w przyrodzie. Przykładem są algorytmy genetyczne, których działanie wzorowane jest na ewolucji biologicznej - spośród znalezionych w danym pokoleniu rozwiązań, wybierane są najlepsze z nich (według pewnych ustalonych dla danego problemu kryteriów), traktowane są jako rodzice dla następnego pokolenia, które dziedziczy po rodzicach ich cechy. Wykorzystywane są również różnego rodzaju operatory mutacji, katastrofy itp.

1.1. Cel pracy

Celem niniejszej pracy jest dokonanie przeglądu wybranych algorytmów przybliżonych, ich wad i zalet oraz przedstawienie ich wykorzystania w kontekście problemu przydziału kwadratowego. Następnie, przy użyciu specjalnie napisanej na potrzeby pracy aplikacji, która rozwiązuje problem przydziału kwadratowego, należy zaprezentować rezultaty przeprowadzonych eksperymentów oraz opisać zastosowane scenariusze testowe i dokonać analizy otrzymanych wyników.

1.2. Zawartość pracy

Rozdział nr 2 zawiera opis problemu przydziału kwadratowego, obszar jego zastosowań i jego model matematyczny.

W rozdziale trzecim zostały przedstawione wybrane algorytmy aproksymacyjne, geneza ich powstania oraz wybrane sposoby ich wykorzystania w celu rozwiązania problemu przydziału kwadratowego.

Rozdział czwarty poświęcony jest idei kwantowych algorytmów ewolucyjnych, a także opisano w nim wybrany algorytm kwantowy NPQGA, który został zaimplementowany w aplikacji będącej jednym z celów niniejszej pracy. Wprowadzone zmiany i modyfikacje we wspomnianym algorytmie są tematem kolejnego, piątego rozdziału pracy.

Następny, szósty rozdział zawiera opis utworzonej aplikacji, informacje o działaniu algorytmu i obsłudze programu.

Rozdział siódmy omawia metodykę eksperymentów przeprowadzanych przy wykorzystaniu napisanej aplikacji. Zawarte w nim są opisy wybranych instancji testowych, a także przedstawione są scenariusze testowe.

Rozdział ósmy poświęcony jest faktycznie przeprowadzonym eksperymentom, zawiera informacje o tym, jakie ustawienia parametrów algorytmu były testowane i porównywane dla wybranych instancji testowych.

W kolejnym rozdziale zostały przedstawione rezultaty opisanych w poprzednim rozdziale testów, zawarte zostały wykresy obrazujące działanie algorytmów dla różnych nastaw oraz różne statystyki dające obraz o tym, jaki wpływ na rezultaty mają zmiany w ustawieniach konkretnych parametrów.

Rozdział dziesiąty skupia się na analizie rezultatów uzyskanych z przeprowadzonych testów. Opisane są wnioski dotyczące poszczególnych eksperymentów.

Ostatni, jedenasty rozdział poświęcony jest ogólnym wnioskom dotyczącym tematyki całej pracy oraz jej podsumowaniu.

2. Zagadnienie przydziału kwadratowego

2.1. Opis problemu

Zagadnienie przydział kwadratowego (Qadratic Assignment Problem - QAP) jest jednym najtrudniejszych problemów optymalizacji kombinatorycznej. Należy on do klasy problemów NP - trudnych. Dla rozmiarów o wartości większej niż 30 wymagane jest stosowanie algorytmów przybliżonych w celu jego rozwiązania. Zagadnienie przydziału kwadratowego zostało przedstawione przez Koopmansa i Beckmanna w roku 1957 jako sposób rozwiązania zagadnień ekonomicznych. Problem ten jest matematycznym modelem sytuacji, w której chcemy przydzielić pewną ilość placówek do takiej samej ilości lokalizacji (miejsc) znając przy tym odległości pomiędzy danymi lokalizacjami oraz wartość przepływu miedzy placówkami. Przydziału tego należy dokonać minimalizując koszt tej operacji, który jest proporcjonalny do przepływu pomiędzy placówkami pomnożonego przez odległość między miejscami, do których te placówki zostały przydzielone. Istnieją również wersje tego problemu, w których podany jest też koszt samego przydziału placówki do lokalizacji. Ponieważ ten problem jest trudny do rozwiązania oraz modeluje on wiele faktycznych zagadnień, wielu autorów poświęciło mu dużo uwagi, przez co znaleźć można wiele różnych publikacji traktujących o problemie QAP. Niewątpliwie postępujący rozwój w dziedzinie informatyki i elektroniki pozwolił na analizę coraz bardziej złożonych problemów i tworzenie nowych metod, które dotychczas nie byłyby możliwe do wykorzystania. Rezultatem tego jest możliwość rozwiązywania problemu QAP dla coraz większych rozmiarów i stosowania go do modelowania coraz nowszych zagadnień.

2.2. Obszary zastosowań

Przy pomocy problemu przydziału kwadratowego można modelować wiele różnych zagadnień, które występują w otaczającym nas świecie. Do dziedzin, w których zagadnienie QAP znajduje zastosowanie, należą m. in:

– ekonomia,

– informatyka,

– elektronika,

– logistyka,

– mechanika,

– architektura.

Do wybranych problemów z wymienionych wyżej dziedzin należą m. in:

– projektowanie zagospodarowania przestrzennego w nowopowstających miastach,

– projektowanie układów elektroniki [10],

– właściwa lokalizacja fabryk [10],

– organizacja biur, oddziałów szpitalnych [10],

– wyważanie turbin w silnikach odrzutowych [10].

2.3. Model matematyczny

Model matematyczny zagadnienia przydziału kwadratowego może być przedstawiony w następujący sposób [10]:

Dany jest zbiór:

$$N = \{1, ..., n\} \tag{2.1}$$

oraz następujące macierze o wymiarach $n \times n$*:*

$$A = (a_{ij}),\ B = (b_{ij}),\ C = (C_{ij}) \tag{2.2}$$

gdzie macierz A jest macierzą odległości pomiędzy lokalizacjami. Z tego powodu często macierz ta oznaczana jest też literą D, od angielskiego słowa distance, oznaczającego odległość. Macierz B jest macierzą określającą pewne powiązania pomiędzy placówkami, np. przepływ informacji, ilość połączeń, ilość towaru, jaką należy przetransportować z jednej lokalizacji do drugiej, itp. Macierz ta jest też oznaczana literą F (ang. flow - przepływ). Macierz C określa koszt przydziału placówki do lokalizacji. Dana jest również funkcja celu określona w następującej sposób:

$$\Phi(\pi) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} \cdot b_{\pi(i),\pi(j)} + \sum_{i=1}^{n} c_{\pi(i),i} \tag{2.3}$$

gdzie π *jest permutacją:* $\pi = (\pi(1), \pi(2), ..., \pi(n))$*, a* $\pi(i)$ *oznacza numer placówki przydzielonej do i-tej lokalizacji. Funkcja celu określa więc ogólny koszt przydziału i eksploatacji przydzielonego systemu. Szukana jest zatem permutacja minimalizująca funkcję celu, czyli taka, dla której wspomniany koszt jest najmniejszy.*

2.4. Złożoność obliczeniowa

Rozwiązanie problemu QAP jest permutacją. Należy przydzielić n placówek do n miejsc. Wynika stąd, że wszystkich możliwości przydziału jest $n!$. Jak zostało wspomniane wcześniej zagadnienie przydziału kwadratowego jest problemem NP-trudnym, czyli zadaniem o złożoności co najmniej wykładniczej. Zadanie o złożoności silni jest zadaniem o złożoności jeszcze większej niż wykładnicza. Wynika z tego fakt, iż już dla problemu o stosunkowo małych rozmiarach, znalezienie rozwiązania poprzez wykorzystanie algorytmów znajdujących dokładne rozwiązanie staje się praktycznie niemożliwe. Zjawiska modelowane zagadnieniem QAP mają rozmiary nierzadko liczone w setkach i większe. Znalezienie dokładnego rozwiązania, przy wykorzystaniu znanych metod i dostępnego obecnie sprzętu, mogłoby wtedy zająć czas nawet dłuższy niż znany wiek Wszechświata. Chcąc więc znaleźć rozwiązanie postawionego problemu należy stosować algorytmy, które poradzą sobie w czasie zdecydowanie krótszym. Receptą są algorytmy przybliżone, inaczej zwane aproksymacyjnymi. Zagadnieniu algorytmów przybliżonych poświęcony jest następny rozdział niniejszej pracy.

3. Algorytmy przybliżone

Złożoność otaczającego nas świata powoduje, że bardzo często występujące problemy bardzo trudne do rozwiązania. Dotyczy to praktycznie każdej sfery ludzkiego życia. W wielu sytuacjach natura problemu nie pozwala na zastosowanie metod matematycznych, jednakże nawet w przypadku takich trudności, w których matematyka przychodzi z pomocą, można stwierdzić jedynie, że problem ma rozwiązanie i to nawet najlepsze z możliwych, optymalne, lecz znalezienie go jest praktycznie niewykonalne. Złożoność obliczeniowa algorytmów pozwalających na ich rozwiązanie jest zbyt duża, by w ogóle warto było je stosować. Pojawia się więc potrzeba zastosowania czegoś, co pozwoli na znalezienie rozwiązania dobrego, przybliżającego chociaż rozwiązanie optymalne. I faktycznie jest grupa algorytmów, które pozwalają na uzyskanie takiego efektu. Są to algorytmy przybliżone, inaczej zwane aproksymacyjnymi.

W przeciwieństwie do problemów optymalizacji, których rozwiązanie można znaleźć w czasie wielomianowym, problemy NP-trudne nie dają „punktu wyjścia" do znalezienia rozwiązania optymalnego. Jednakże, niejednokrotnie istnieje „punkt wyjścia", który pozwala na dojście do rozwiązania znajdującego się w pobliżu rozwiązania najlepszego. W tym sensie algorytmy przybliżone podobne są do algorytmów dokładnych: również polegają na uchwyceniu istoty problemu i następnie na znalezieniu algorytmu, który pozwoli na jej wykorzystanie [**?**].

Ogromna ilość problemów, dla których nie jesteśmy w stanie znaleźć rozwiązania optymalnego, przyczyniła się do powstania wielu algorytmów aproksymacyjnych. Przy tworzeniu algorytmów dąży się do tego, by działały one jak najszybciej i algorytmy przybliżone nie są w tym przypadku wyjątkiem. Niestety, w przypadku rozwiązywania przy ich użyciu wielu problemów czas ich działania jest dosyć długi. Jednakże, należy podkreślić, że pozwalają one na znalezienie dobrego rozwiązania w sytuacji, gdy użycie algorytmów dokładnych nie pozwoliłoby uzyskać rozwiązania w ogóle.

Ciekawą rzeczą związaną z algorytmami przybliżonymi jest fakt, że wiele z nich powstało na podstawie obserwacji zjawisk występujących w przyrodzie, takich jak zachowanie się większych grup zwierząt, mechanizmów jakie wykorzystują w celu zwiększenia swoich szans na przeżycie, adaptacja do nowych warunków, podatność na zmiany, ewolucja.

Poniżej zostaną przedstawione niektóre algorytmy aproksymacyjne, podstawowe informacje na ich temat, opisany będzie schemat ich działania, a także to, w jaki sposób przy ich pomocy można rozwiązać problem przydziału kwadratowego.

3.1. Particle Swarm Optimization

3.1.1. Geneza i opis algorytmu

Algorytm Particle Swarm Optimization (PSO), czyli algorytm optymalizacji rojem cząstek, po raz pierwszy został przedstawiony w pracy Jamesa Kennedy'ego i Russella Eberharta w 1995 roku, jako metoda optymalizacji nieliniowych funkcji ciągłych. Metoda powstała w oparciu o przeprowadzane symulacje uproszczonych modeli zachowań społecznych. Inspiracją dla autorów były również przeprowadzane przez naukowców komputerowe symulacje zachowań stad ptaków czy ławic ryb.

Zachowania stad ptaków zawsze interesowały naukowców. Chcieli oni dociec, w jaki sposób ptaki potrafią, latając w licznych stadach, lecieć w sposób synchroniczny, często zmieniając przy tym kierunek lotu czy też błyskawicznie się przegrupowując. Z czasem powstawały różnego rodzaju modele tychże zachowań, programy pozwalające na symulowanie ich. Również ciekawą rzeczą był fakt, że ptaki potrafią znaleźć sobie pożywienie, ominąć zagrożenie, mimo że nie posiadają początkowo wiedzy na ten temat. Pojawiły się tezy, że potrafią one wykorzystać zdobytą wiedzę przez inne osobniki, czy też poprzednie pokolenia. Dążenie do znalezienia pokarmu, próby unikania sytuacji niebezpiecznych czy drapieżników są czynnikami decydującymi o poprawie „sytuacji życiowej" ptaków. Jest to swego rodzaju optymalizacja dokonywana samoistnie przez naturę. Analiza tych zachowań stała się punktem wyjścia do tworzenia algorytmów pozwalających na rozwiązywanie wielu trudnych problemów.

Algorytm PSO w pewien sposób przypomina wspomniane wcześniej symulacje, lecz zawiera też parę istotnych różnic. W klasycznej wersji, algorytm zawiera rój cząstek poruszających się w wielowymiarowej przestrzeni, który inicjowany jest w sposób losowy. Cząstki te reprezentują rozwiązania problemu i scharakteryzowane poprzez swoją prędkość i położenie. Ruch cząstek w kolejnych iteracjach ma na celu przeszukiwanie przestrzeni rozwiązań. Każda z cząstek zapamiętuje znalezioną przez siebie dotychczas najlepszą pozycję. W oparciu o te pozycje, w każdej iteracji cząstki mają aktualizowaną swoją prędkość i położenie.

Algorytm PSO posiada wiele zalet. Przede wszystkim jest bardzo prosty i wydajny, oraz pozwala na optymalizację wielu różnych funkcji. Aktualizacja prędkości i położenia cząstek wymaga jedynie podstawowych operacji matematycznych. Algorytm nie wymaga również zapamiętywania dużej ilości danych, dlatego jest wydajny z punktu widzenia szybkości działania i nie wymaga wielu zasobów pamięci. Ważną cechą jest również to, że jest on bardzo odporny na wpadnięcie do minimum lokalnego.

3.1.2. Model matematyczny algorytmu

Model matematyczny algorytmu PSO może być przedstawiony w następujący sposób:
Mamy dany rój, który składa się z n cząstek. Każda z nich porusza się w d-wymiarowej przestrzeni i każda z nich opisana jest przez dwa wektory:

– *wektor położenia:*

$$x_i = [x_{i1}, x_{i2}, ..., x_{id}] \tag{3.1}$$

– *wektor prędkości:*

$$v_i = [v_{i1}, v_{i2}, ..., v_{id}] \tag{3.2}$$

Ponadto, każda z cząstek zapamiętuje znalezioną przez siebie najlepszą dotychczas pozycję w wektorze:

$$x_i^b = [x_{i1}^b, x_{i2}^b, ..., x_{id}^b] \tag{3.3}$$

Zapamiętywana jest również w wektorze x^* najlepsza pozycja znaleziona dotychczas przez wszystkie cząstki w roju.

Wartości prędkości i położenia w każdej iteracji algorytmu aktualizowane są odpowiednio według poniższych wzorów[odniesienie]:

$$v_{ij}(t) = w \cdot v_{ij}(t-1) + c_1 \cdot r_1 \cdot (x_{ij}^b(t-1) - x_{ij}(t-1)) + c_2 \cdot r_2 \cdot (x_j^*(t-1) - x_{ij}(t-1)) \tag{3.4}$$

$$x_{ij}(t) = x_{ij}(t-1) + v_{ij}(t) \tag{3.5}$$

gdzie liczby r_1 i r_2 są wybierane losowo z przedziału $[0, 1]$, natomiast współczynniki c_1 i c_2 odpowiadają za to, w jakim stopniu do aktualizacji prędkości brane są pod uwagę najlepsze znalezione dotychczas położenia każdej z cząstek z osobna i najlepsze położenie w ogóle. Parametr w określa bezwładność cząstek i z czasem maleje liniowo do 0.

3.1.3. Pseudokod dla algorytmu PSO

Poniżej znajduje się pseudokod, który opisuje jak krok po kroku działa algorytm optymalizacji rojem cząstek:

Wczytaj rozmiar roju n, wymiar d, ilość iteracji t i inne parametry;
while *nie wystąpił warunek stopu* **do**
  $t \leftarrow t + 1$;
  for $i \leftarrow 1$ **to** n **do**
    Policz dopasowanie cząstki x_i;
    if x_i *jest lepsza niż* x_i^b **then**
      $x_i^b \leftarrow x_i$
    end
    if x_i^b *jest lepsza niż* x^* **then**
      $x^* \leftarrow x_i^b$
    end
  end
  for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** d **do**
      Zaktualizuj prędkość v_{ij};
      Zaktualizuj położenie x_{ij};
    end
  end
end

Algorithm 1: Algorytm PSO

3.1.4. Zastosowanie algorytmu PSO dla problemu QAP

Aby było możliwe zastosowanie algorytmu PSO do rozwiązania problemu przydziału kwadratowego, należy odpowiednio ująć problem QAP, by dało się go wpasować w model algorytmu. Przede wszystkim rozwiązaniami zagadnienia przydziału kwadratowego są permutacje, czyli jest to problem dyskretny. Pozycje cząstek w algorytmie PSO mogą zmieniać się w sposób ciągły, położenie nie musi być określone współrzędnymi całkowitymi. Również w permutacji elementy nie mogą się powtarzać. Natomiast nic nie stoi na przeszkodzie, by zwrócona przez algorytm pozycja cząstki była opisana w każdym kierunku przez współrzędne o tej samej wartości. Proste mapowanie: wartość położenia w $i-tym$ kierunku określa przydzielenie do $i-tej$ lokalizacji obiektu o tejże wartości może powodować, że dany obiekt będzie przydzielony wielokrotnie. Jedno z możliwych zastosowań algorytmu PSO dla problemu QAP zostało zaproponowane w publikacji [14]. Dla danych zbiorów obiektów i lokalizacji, odpowiednio:

$$F = \{F_1, F_2, ..., F_n\} \tag{3.6}$$

$$L = \{L_1, L_2, ..., L_n\} \tag{3.7}$$

tworzy się macierz:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \tag{3.8}$$

gdzie a_{ij} oznacza stopień przynależności *j-tego* obiektu do *i-tej* lokalizacji. Ponieważ rozwiązanie jest permutacją, do jednej lokalizacji należy przypisać tylko jeden obiekt, więc muszą być spełnione ograniczenia:

$$\sum_{i=1}^{n} a_{ij} = 1 \tag{3.9}$$

$$\sum_{j=1}^{n} a_{ij} = 1 \tag{3.10}$$

oraz

$$a_{ij} \in \{0, 1\},\ i = 1, 2, ..., n,\ j = 1, 2, ..., n \tag{3.11}$$

Zastosowanie algorytmu optymalizacji rojem cząstek wymaga zatem przedefiniowania pozycji i prędkości w następujący, bazujący na macierzy 3.8, sposób:

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix} \tag{3.12}$$

$$V = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n1} & v_{n2} & \cdots & v_{nn} \end{bmatrix} \tag{3.13}$$

przy czym, by spełnione były ograniczenia 3.9 i 3.10 stosuje się normalizację macierzy położenia:

$$X_{norm} = \begin{bmatrix} \frac{x_{11}}{\sum_{i=1}^{n} x_{i1}} & \frac{x_{12}}{\sum_{i=1}^{n} x_{i2}} & \cdots & \frac{x_{1n}}{\sum_{i=1}^{n} x_{in}} \\ \frac{x_{21}}{\sum_{i=1}^{n} x_{i1}} & \frac{x_{22}}{\sum_{i=1}^{n} x_{i2}} & \cdots & \frac{x_{2n}}{\sum_{i=1}^{n} x_{in}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{x_{n1}}{\sum_{i=1}^{n} x_{i1}} & \frac{x_{n2}}{\sum_{i=1}^{n} x_{i2}} & \cdots & \frac{x_{nn}}{\sum_{i=1}^{n} x_{in}} \end{bmatrix} \tag{3.14}$$

By uzyskać z macierzy położenia *X* rozwiązanie problemu QAP, w każdej kolumnie macierzy wybierany jest element o największej wartości i przypisywana jest jemu wartość *1*, a pozostałym *0*. Po sprawdzeniu wszystkich kolumn i wierszy otrzymuje się w ten sposób macierz określającą, w jaki sposób przypisać obiekty do lokalizacji. Macierz ta spełnia również ograniczenia 3.9 oraz 3.10.

3.2. Algorytm Tabu Search

3.2.1. Geneza i opis algorytmu

Algorytm Tabu Search został zaproponowany przez Freda Glovera w roku 1986. Jest to metaheurystyka pozwalająca innym metodom optymalizacji unikać sytuacji, w których te wpadają w minima lokalne. Dzięki metodzie tabu search udało się znaleźć optymalne lub prawie optymalne rozwiązania dla bardzo wielu problemów optymalizacji takich jak szeregowanie zadań, problem przydziału kwadratowego, rozpoznawanie charakteru, kolorowanie grafów [15].

Słowo tabu kojarzone jest przede wszystkim z czymś zakazanym, najczęściej na tle kulturowym. W przypadku algorytmu należy je rozumieć bardziej jako ograniczenie. W ogólności algorytm tabu search polega na zabranianiu wykonywania tak zwanych ruchów, czyli operacji modyfikujących rozwiązania. Ruch jest funkcją, która transformuje dane rozwiązanie w inne, w przypadku permutacji może to być zamiana miejscami dwóch jej elementów. W danym momencie możliwy jest pewien podzbiór rozwiązań, w które inne może być przetransformowane. Z dostępnych ruchów wybierany jest ten, który powoduje polepszenie rozwiązania i ostatnio wykonany ruch dodawany jest do tablicy ruchów zabronionych na pewną określoną liczbę iteracji algorytmu. Mechanizm ten pozwala na wyjście z minimum lokalnego i pozwala uniknąć ruchów cyklicznych. Jednakże w pewnych określanych sytuacjach możliwe jest wykonanie ruchu zabronionego. Zdefiniowana jest specjalna funkcja, zwana funkcją aspiracji, która pozwala obliczyć, czy zabroniony ruch będzie jednak opłacalny [12]. Najczęstszym przypadkiem dopuszczenia do użycia zabronionego ruchu jest sytuacja, gdy jego wykonanie pozwoli na uzyskanie lepszego rozwiązania niż najlepsze dotychczas.

Algorytm zatrzymuje się, gdy spełniony jest jeden z warunków zatrzymania. Takimi warunkami mogą być wykonanie z góry założonej iteracji algorytmu czy też wykonanie ustalonej liczby ruchów, które nie prowadzą do dalszej poprawy rozwiązania.

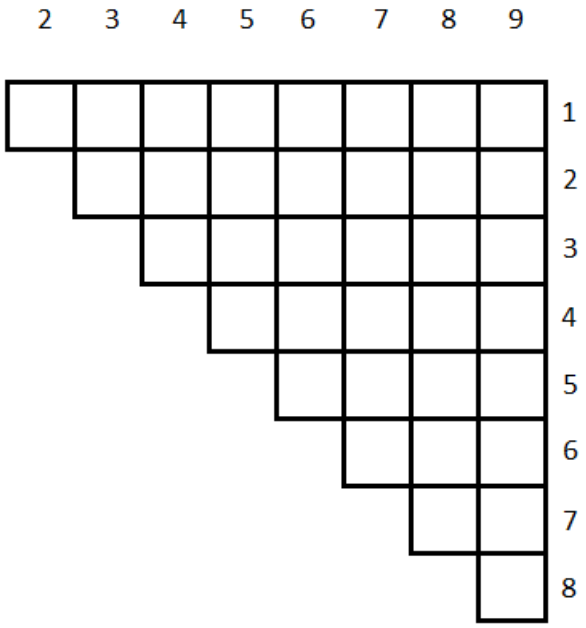
3.2.2. Pseudokod algorytmu Tabu Search

Inicjalizuj pierwsze rozwiązanie x;
Inicjalizuj rozwiązanie najlepsze x^b: $x^b \leftarrow x$;
while *nie wystąpił warunek stopu* **do**
  Przygotuj listę możliwych ruchów dla obecnego rozwiązania;
  Wybierz najlepszy możliwy ruch z uwzględnieniem tablicy tabu i kryterium aspiracji;
  Przypisz otrzymane w ruchu rozwiązanie do rozwiązania aktualnego x;
  if *rozwiązanie x jest lepsze od* x^b **then**
    $x^b \leftarrow x$
  end
  Zaktualizuj tablicę tabu i kryterium aspiracji;
end

Algorithm 2: Algorytm Tabu Search

3.2.3. Zastosowanie algorytmu Tabu Search dla problemu QAP

Rozwiązaniami problemu przydziału kwadratowego są permutacje określające przydział placówek do lokalizacji. Należy więc, mając dane aktualne rozwiązanie problemu QAP, określić w jaki sposób będzie wykonywany ruch w kolejnych iteracjach działania algorytmu. Zmiana aktualnego rozwiązania musi odbyć się w sposób, który nie spowoduje, że do danej lokalizacji zostanie przypisany więcej niż jeden obiekt, jak również któryś z obiektów nie zostanie desygnowany do żadnego z miejsc. W przeciwieństwie do, przykładowo, algorytmu PSO, strategia Tabu Search pozwala na przeszukiwanie przestrzeni rozwiązań problemu QAP w sposób dosyć prosty. Istnieje wiele metod dokonywania ruchów w przypadku, gdy rozwiązanie jest permutacją. Najczęściej spotykany w literaturze jest sposób polegający na zamianie miejscami dwóch elementów permutacji. Wynika stąd, że dla permutacji o długości n istnieje $\binom{n}{2}$ kombinacji takiego wyboru. Wykonane ruchy zapisywane są w tablicy tabu i trzymane w niej przez określoną liczbę iteracji algorytmu. Poniżej znajduje się przykładowa tablica [5]:



Rysunek 3.1: Tablica tabu

W powyższej tablicy w komórce o indeksie (i, j) wpisuje się liczbę iteracji algorytmu, podczas których obiektów o wartościach (nie indeksach) *i* i *j* nie można zamienić miejscami. Po każdej iteracji liczba ta jest zmniejsza o 1. Wpis dodawany jest, gdy nastąpiła zamiana miejscami obiektów o wartościach i i j.

Innymi metodami pozwalającymi na wykonanie ruchu w algorytmie TS są przykładowo wstawienie jednego z elementów permutacji w inne miejsce i przesunięcie pozostałych elementów, czy też inwersja wybranej grupy elementów permutacji o określonej szerokości.

3.3. Algorytm mrówkowy

3.3.1. Geneza i opis algorytmu

Algorytm mrówkowy (Ant Algorithm) został stworzony przez Marco Dorigo, jako metoda rozwiązywania trudnych problemów optymalizacji jakimi są przykładowo problem komiwojażera (TSP - Travelling Salesman Problem), czy problem przydziału kwadratowego QAP. Inspiracją do powstania algorytmu była obserwacja faktycznych, istniejących w naturze, rojów mrówek. Uwagę naukowców przykuło to, że mrówki, które same są dosyć prostymi stworzeniami, działając w grupie potrafią osiągnąć wysoki poziom organizacji, żyją w zhierarchizowanym społeczeństwie. Również ciekawą cechą w zachowaniu mrówek jest to, że nastawione są bardziej na przeżycie całej społeczności niż pojedynczego osobnika.

Posiadają one także niespotykane umiejętności znajdywania najkrótszej drogi pomiędzy mrowiskiem a miejscem, w którym znajduje się pożywienie [9].

Ważnym czynnikiem, pozwalającym na znajdywanie najkrótszej ścieżki do źródła pokarmu oraz zapamiętywania tejże drogi, są substancje chemiczne wydzielana przez mrówki, zwane feromonami. Insekty te mają zdolność wyczuwania feromonów i dzięki temu najprawdopodobniej potrafią wybrać drogę, dla której stężenie feromonów jest największe. Pozwala to również innym osobnikom na wykorzystanie informacji o lokacji pożywienia zdobytej przez inne mrówki. Im częściej dana ścieżka jest uczęszczana przez mrówki, tym większe stężenie feromonów. Mrówki będą zatem korzystać z dróg, na których feromony są bardziej wyczuwalne. Również idąc drogą o mniejszym stężeniu feromonów, mrówki po wyczuciu ich większego stężenia na innej trasie, skierują się na nią tworząc, poprzez zostawianie tegoż związku chemicznego, nowe połączenia. Można w związku z powyższym metaforycznie stwierdzić, że mrówki posiadają zdolność wychodzenia z minimów lokalnych i wybierają minimum globalne.

Z pierwotnych założeń o algorytmie mrówkowym, wyewoluowała cała rodzina algorytmów mrówkowych, rozszerzając w ten sposób ilość problemów, które można dzięki nim rozwiązać.

3.3.2. System mrówkowy - Ant System (AS)

Jednym z możliwych algorytmów mrówkowych jest system mrówkowy. Jest to pierwszy algorytm bazujący na optymalizacji kolonii mrówek (ACO). Na jego podstawie zostało później opracowanych wiele innych algorytmów. Odpowiednie podejście do problemu przydziału kwadratowego pozwala na wykorzystanie systemu mrówkowego do jego rozwiązania. Podczas wypracowywania rozwiązania problemu QAP mrówka określa z pewnym prawdopodobieństwem, który obiekt przypisać do danej lokalizacji. Prawdopodobieństwo to można wyznaczyć z poniższego wzoru [11]:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum\limits_{l \in N_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta},\ j \in N_i^k \tag{3.15}$$

gdzie α i β są parametrami określającymi odpowiednio wagę śladu feromonowego τ_{ij} i wartości heurystycznej η_{ij}, a N_i^k jest tzw. sąsiedztwem *i-tego* węzła, czyli zbiorem pozostałych wolnych pozycji, do których nie zostały jeszcze przydzielone żadne obiekty.

Ślad feromonowy jest aktualizowany w następujący sposób [11]:

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{3.16}$$

gdzie ρ jest współczynnikiem wyparowywania feromonów zostawianych przez mrówki, a Δ_{ij}^k określone jest wzorem [11]:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{J^k},\ gdy\ obiekt\ i\ jest\ przypisany\ do\ lokalizacji\ j \\ 0,\ w\ przeciwnym\ przypadku \end{cases} \tag{3.17}$$

gdzie J^k jest funkcją celu, a parametr *Q* określa ile feromonów zostawia mrówka.

Obliczenie informacji heurystycznej wymaga wykorzystania dwóch wektorów: *a*, którego *i-ty* element jest sumą odległości lokalizacji *i* do pozostałych lokalizacji, a także wektora *b*, którego *i-ty* element jest analogiczną sumą przepływów. Na podstawie tych wektorów wyznaczana jest macierz E [11]:

$$E = b \cdot a^T. \tag{3.18}$$

Dzięki tej macierzy zwiększane jest prawdopodobieństwo przypisania do lokalizacji znajdujących się blisko siebie obiektów o dużym przepływie.

Obiekty oraz lokalizacja, które zostały już przydzielone, zostają zablokowane dopóki rozwiązanie problemu QAP nie zostanie ukończone [11].

3.3.3. Algorytm MMAS

Algorytm MMAS (*max - min ant system*) jest modyfikacją systemu mrówkowego z wprowadzeniem minimalnego i maksymalnego poziomu feromonów. W tej wersji algorytmu tylko jedna z mrówek - najlepsza globalnie lub w danej iteracji, zostawia za sobą ślad feromonów. Przy Inicjacji algorytmu każdy ze śladów feromonowych jest ustawiany na maksymalną wartość τ_{max}. Następnie, podobnie jak w omówionym wcześniej systemie mrówkowym, mrówki przydzielają do niewybranych jeszcze lokalizacji nieprzydzielone obiekty z pewnym prawdopodobieństwem, które określone jest dla *k-tej* mrówki w następujący sposób [11]:

$$p_{ij}^{k}(t) = \frac{\tau_{ij}(t)}{\sum\limits_{l \in N_i^k} \tau_{il}(t)},\ j \in N_i^k \tag{3.19}$$

a uaktualnienie śladu feromonów, który jest przez tę mrówkę zostawiany, uzyskiwane jest z równania [11]:

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}^{best} \tag{3.20}$$

gdzie

$$\Delta\tau_{ij}^{best} = \begin{cases} \frac{1}{J^{best}},\ gdy\ obiekt\ i\ jest\ przypisany\ do\ lokalizacji\ j \\ 0,\ w\ przeciwnym\ przypadku \end{cases} \tag{3.21}$$

Oznaczenia w powyższych wzorach są analogiczne, do tych dla systemu mrówkowego.

3.4. Algorytmy ewolucyjne

Algorytmy ewolucyjne są, najogólniej rzecz biorąc, algorytmami optymalizacji, które bazują na stopniowym polepszaniu pewnej populacji rozwiązań danego problemu. Z powodzeniem są stosowane od wielu lat do rozwiązywania wielu zarówno praktycznych jak i teoretycznych problemów. Istnieje wiele różnorodnych implementacji algorytmów ewolucyjnych. Należą do nich między innymi [19]:

– algorytmy genetyczne, stworzone przez Johna Henry'ego Hollanda,

– strategie ewolucyjne, opracowane przez Ingo Rechenberga oraz Hansa Paul Schwefela,

– programowanie ewolucyjne, stworzone przez Lawrence'a Fogela.

Algorytmy ewolucyjne posiadają wiele cech, które odróżniają je od innych metod optymalizacji. Przede wszystkim zmianom poddawana jest zakodowana w łańcuchu znaków postać problemu, a nie jego parametry bezpośrednio i wykorzystywane jest doświadczenie poprzednich pokoleń. Łańcuch ten ma ustaloną długość i korzysta ze znaków ze skończonego alfabetu. Jak już zostało to wspomniane wcześniej, przetwarzana jest też pewna populacja rozwiązań, nie tylko jedno. By ocenić dane rozwiązanie potrzebna jest jedynie funkcja celu, bądź też coś, co pozwoli porównać dwa rozwiązania i wyłonić lepsze z nich. Kolejnym elementem, który cechuje algorytmy ewolucyjne jest fakt, że stosowane są w nich niedeterministyczne, probabilistyczne reguły wyboru [2].

Algorytmy genetyczne są chyba najczęściej stosowanymi i najbardziej znanymi implementacjami algorytmów ewolucyjnych. Często, choć niepoprawnie, terminy *algorytmy ewolucyjne* oraz *algorytmy genetyczne* stosowane są zamiennie. Z powodu ich popularności, dalsza część rozdziału będzie poświęcona tejże podgrupie algorytmów ewolucyjnych.

3.4.1. Geneza i opis algorytmów genetycznych

Algorytmy genetyczne zostały opracowane, jak już zostało to wspomniane, przez Johna Hollanda przy pomocy jego kolegów i studentów związanych z Uniwersytetem Michigan. Celami, które im przyświecały podczas tworzenia algorytmów były chęć opisania i wyjaśnienia istoty zjawisk zachodzących w świecie przyrody, dzięki którym możliwa jet adaptacja do różnych warunków, a także utworzenie oprogramowania mogącego symulować mechanizmy obecne w systemach biologicznych. Ważną cechą rzeczywistych systemów biologicznych jest odporność tych systemów. Potrafią one szybko zaadaptować się do zmieniających się otaczających ich warunków, posiadają duże zdolności regeneracyjne. Niewątpliwie są to cechy pożądane także przy projektowaniu różnego rodzaju systemów: inżynierskich, ekonomicznych [3]. Skoro algorytmy genetyczne pojawiły się w oparciu o symulacje zjawisk zachodzących

w naturze, może pojawić się pytanie, czy one same posiadają podobne zdolności. Odpowiedź na nie podał Holland w roku 1975 udowadniając, że algorytmy genetyczne są odporną metodą poszukiwania rozwiązań nawet w skomplikowanych przestrzeniach [3]. Inną ważną zaletą algorytmów genetycznych jest to, że stosowanie ich nie wymaga spełniania założeń dotyczących przestrzeni poszukiwań, jakimi są przykładowo ciągłość czy różniczkowalność [3]. Dzięki stosowaniu probabilistycznych technik wyboru, a także dzięki przetwarzaniu całej populacji rozwiązań, w przeciwieństwie do wielu innych analitycznych metod optymalizacji, znacznie zmniejszona jest szansa na zatrzymanie się w ekstremum lokalnym. W tym sensie metody analityczne przejawiają swój brak odporności. Jednakże, z faktu, iż algorytm genetyczny jest metodą przybliżoną, w przypadku optymalizowania wielu problemów nie jest możliwe stwierdzenie, czy znalezione rozwiązanie, zwrócone przez algorytm jest optymalne i jak daleko znajduje się od optimum. Nie można również zagwarantować, że start algorytmu przy pewnych początkowych ustawieniach zawsze zwróci ten sam rezultat.

W klasycznej, najprostszej wersji, algorytm genetyczny składa się z kilku podstawowych operacji. Mając wygenerowaną losowo populację początkową uruchamiamy algorytm na określoną liczbę operacji. W każdej iteracji dokonujemy oceny każdego z rozwiązań w populacji i w oparciu o ich wartość dopasowania dokonujemy selekcji osobników, na których będą wykonane operacje krzyżowania oraz mutacji. W zależności od postaci rozwiązania, a także od metody jego kodowania, operatory genetyczne mogą działać w różnoraki sposób. W ogólnym przypadku, krzyżowanie polega na wymianie informacji pomiędzy osobnikami wybranymi na drodze selekcji, a mutacja na losowej zmianie wybranego rozwiązania, mogącej zmienić dane rozwiązanie na lepsze bądź gorsze. Teoretycznie z każdą iteracją, ogólne dopasowanie całego pokolenia powinno się polepszać. Ogólną ideą algorytmu jest przetrwanie najsilniejszych i eliminacja słabszych.

3.4.2. Operatory selekcji

Już sam sposób doboru osobników, które posłużą za podstawę dla kolejnego pokolenia, ma kluczowe znaczenie. Istnieje wiele sposobów wyboru rozwiązań. Do takich metod można zaliczyć między innymi:

- metodę ruletki,
- metodę rankingową,
- metodę turniejową,
- metodę progową.

Metoda ruletki polega na „kręceniu” wirtualną ruletką, na której każde z rozwiązań z aktualnej populacji ma wyznaczony swój wycinek koła. Szerokość wycinka zależy od dopasowania danego rozwiązania. Dla każdego rozwiązania jest liczona wartość funkcji dopasowania i wyznaczana jest oprócz tego suma wszystkich wartości dopasowania rozwiązań w pokoleniu. Stosunek wartości dopasowania danego osobnika do sumy wszystkich dopasowań wyznacza szerokość wycinka ruletki dla danego rozwiązania.

Metoda ta faworyzuje osobniki lepsze, nie pozbawiając jednak szansy wyboru tych gorszych. Jednak może to powodować, że osobniki wybrane do krzyżowania będą się składać głównie z tego jednego w przypadku, gdy to rozwiązanie jest wyraźnie lepsze od pozostałych.

Metoda bazująca na rankingu rozwiązań nieco zmniejsza rolę dopasowania danego rozwiązania w kontekście jego szans do bycia wybranym na rodzica. W tej metodzie rozwiązania są szeregowane według swojego dopasowania, a o prawdopodobieństwie wyboru decyduje pozycja w rankingu. W oparciu o ranking definiuje się funkcję, która określa ile kopii danego rozwiązania jest brane pod uwagę. W tym sposobie dysproporcje między prawdopodobieństwami wyboru rozwiązań są mniejsze w stosunku do metody ruletkowej. Pozycja w rankingu określa jedynie, że rozwiązanie, które jest w nim na wyższym miejscu, jest lepsze, nie biorąc pod uwagę rozmiaru różnicy pomiędzy innymi.

Metoda turniejowa polega na losowym wyborze dwóch rozwiązań z populacji i lepsze z nich jest brane pod uwagę w dalszych operacjach. Rozwiązania lepsze mają taką samą szansę bycia wybranym do porównania, co te słabsze, lecz i tak w przypadku pojedynku, to one wygrają. Istnieją też wersje tej metody, w których rywalizacja odbywa się pomiędzy większą liczbą osobników niż dwa.

Metoda progowa dozwala na losowy wybór rozwiązań spośród tych, których wartość dopasowania przekracza określony próg. W tej metodzie zawsze pewna pula rozwiązań będzie od razu wykluczona z możliwości do dalszej reprodukcji. Istnieje więc ryzyko, iż pewne rozwiązania, które mogą posiadać obiecujące fragmenty, lecz ogólnie są gorsze od pozostałych rozwiązań w populacji, zostaną odrzucone, a cenna informacja, którą posiadają, na drodze późniejszego krzyżowania, nie będzie mogła być wykorzystana.

Należy również wspomnieć, że funkcja dopasowania powinna zwracać liczby nieujemne i dla rozwiązania lepszego jego wartość dopasowania powinna być większa, niż dla gorszego. Nie zawsze da się taką informacje uzyskać bezpośrednio z funkcji celu. Znana jest własność dualności zadań minimalizacji kosztu i maksymalizacji zysku, lecz w przypadku funkcji przystosowania, zwracana wartość zawsze musi być nieujemna. Z tego powodu należy dokonać przekształcenia funkcji celu w funkcję dopasowania. Najczęściej dokonywane to jest poprzez odejmowanie wartości funkcji celu od pewnej liczby. Jeśli taka różnica jest ujemna, zwracamy 0. Wartość tej liczby może być ustalona w wieloraki sposób. Przykładowo, może to być z góry ustalona liczba, może też być modyfikowana wraz z kolejnymi iteracjami algorytmu, np. może przyjąć wartość największej znalezionej do tej pory wartości funkcji celu.

3.4.3. Operatory krzyżowania

Celem operatorów krzyżowania, zwanych inaczej mieszania, jest spowodowanie wymiany informacji pomiędzy wybranymi rozwiązaniami i utworzenie na tej podstawie kolejnych. Mając wyselekcjonowaną populację rozwiązań, łączymy w pary osobniki i dokonujemy ich krzyżowania. W jaki sposób zostanie to wykonane, zależy od postaci rozwiązania. Istnieje wiele różnych metod krzyżowania. Inne są wykorzystywane dla rozwiązań kodowanych w sposób binarny, inne dla kodów wykorzystujących liczby rzeczywiste itp. Do najbardziej znanych metod krzyżowania należą:

– krzyżowanie jednopunktowe,

– krzyżowanie wielopunktowe,

– krzyżowanie z częściowym odwzorowaniem PMX,

– krzyżowanie cykliczne CX,

– krzyżowanie z zachowaniem porządku OX.

Trzy ostatnie z wymienionych operatorów dotyczą rozwiązań będących permutacjami i ich użycie pozwala na zachowanie tej postaci po dokonaniu krzyżowania. Operatory jednopunktowy i wielopunktowy powodują wymianę pomiędzy osobnikami fragmentów kodu pomiędzy wylosowanymi punktami. Operacja krzyżowania wykonywana jest z pewnym określonym jako parametr algorytmu prawdopodobieństwem. Dla każdego z rozwiązań, wybranych na drodze selekcji, losowane jest, czy będzie ono brane pod uwagę w dalszych działaniach. W zależności od przyjętych założeń, może zaistnieć potrzeba doboru dodatkowych rozwiązań w przypadku, gdy jest ich niewystarczająca ilość, by móc dokonać operacji krzyżowania.

3.4.4. Operatory mutacji

Operacja mutacji jest losową zmianą dokonywaną na rozwiązaniach wzorowaną na mutacjach występujących faktycznie w przyrodzie. Jest ona błądzeniem przypadkowych w przestrzeni ciągów kodowych [3]. Operator mutacji pozwala na przywrócenie utraconych ważnych informacji w kodzie rozwiązania, bądź na uzyskanie dobrych (lub złych) cech w rozwiązaniu, których często nie dałoby się otrzymać na drodze krzyżowania. Z racji iż prawdopodobieństwo wystąpienia mutacji jest wielokrotnie mniejsze niż wystąpienie krzyżowania, operacja ta odgrywa drugorzędną rolę w działaniu algorytmu, choć niejednokrotnie pozwala na uzyskanie zaskakujących rezultatów.

W przypadku binarnej postaci kodu rozwiązań operacja mutacji najczęściej polega na zamianie wartości bitów z 0 na 1 i na odwrót. W innych przypadkach mutacja polega na zmianie wybranego elementu na inny dopuszczalny. W sytuacji, gdy rozwiązaniem jest permutacja, należy zadbać, by operator mutacji pozostawiał rozwiązanie w poprawnej postaci. Mutacja w takim przypadku może polegać przykładowo na zamianie miejscami dwóch elementów, czy też na inwersji pewnego fragmentu kodu.

3.4.5. Schemat działania algorytmu genetycznego

Poniżej znajduje się pseudokod dla klasycznej wersji algorytmu genetycznego:
Inicjalizuj populację początkową;
while *nie wystąpił warunek stopu* **do**
    Dokonaj selekcji rozwiązań będącej podstawą dla nowego pokolenia;
    Dokonaj operacji krzyżowania na wybranych osobnikach;
    Dokonaj operacji mutacji na osobnikach otrzymanych na drodze krzyżowania;
    Zaktualizuj populację w oparciu o otrzymane rozwiązania w wyniku działania operatorów genetycznych
end

Algorithm 3: Algorytm genetyczny

3.4.6. Zastosowanie algorytmu genetycznego dla problemu QAP

Algorytm genetyczny można w łatwy sposób zaimplementować dla problemu QAP. Postać rozwiązania problemu przydziału kwadratowego to permutacja. Należy więc na każdym etapie działania algorytmu dokonywać zmian w taki sposób, by zwracane w kolejnych iteracjach populacje rozwiązań były populacjami permutacji. Wyżej zostały wymieniowe metody krzyżowania, takie jak operatory OX, PMX, CX, i mutacji, które mogą być stosowane dla rozwiązań permutacyjnych. Optymalizacja problemu QAP polega na minimalizacji kosztu przydziału obiektów do lokalizacji. Z tego powodu funkcja celu (2.3) nie nadaje się wprost do oceny przystosowania osobników. Funkcję dopasowania można więc uzyskać z funkcji celu poprzez odejmowanie wartości funkcji celu od pewnej liczby, którą może być największa znaleziona dotychczas wartość funkcji celu, czy też najgorsza wartość dopasowania w ostatniej iteracji itp. Poprzez stopniowe polepszanie rozwiązań wraz z kolejnymi iteracjami algorytmu, otrzymuje się ostatecznie rozwiązanie problemu przydziału kwadratowego.

Podobnie jak w przypadku algorytmu tabu search, algorytm genetyczny pozwala na bardzo proste i intuicyjne użycie w celu rozwiązania problemu QAP. Sposób przeszukiwania przestrzeni rozwiązań nie prowadzi do uzyskania rozwiązań niezgodnych z założeniami problemu przydziału kwadratowego i postać zwracanych rozwiązań wprost podaje informację o sposobie przydziału placówek do lokalizacji.

4. Zastosowanie algorytmu quantum EA dla zagadnienia QAP

Sposób w jaki działają algorytmy ewolucyjne, ich otwarty schemat działania, skłania do tworzenia wielu modyfikacji. Przykładowo, w kontekście algorytmów genetycznych, zmianom mogą podlegać operatory krzyżowania, mutacji, sposób kodowania rozwiązania. Można dodawać też nowe operatory o działaniu nieobjętym przez tradycyjne operatory. Z tego powodu, na przestrzeni lat, pojawia się wiele publikacji na temat algorytmów ewolucyjnych i nowych sposób podejścia do tego tematu. W jednej z takich publikacji [20], autorzy Jinwei Gu, Xingsheng Gu i Manzhan Gu zaproponowali algorytm o nazwie *„a novel parallel quantum genetic algorithm" - NPQGA* i przedstawili jego wykorzystanie dla problemu szeregowania zadań. Algorytm ten należy do grupy tak zwanych kwantowych algorytmów ewolucyjnych i nadaje się także dla innych zastosowań do jakich należy na przykład problem QAP.

4.1. Opis algorytmu

Główną cechą kwantowych algorytmów ewolucyjnych jest zastosowanie w nich bitów kwantowych - kubitów. Wykorzystywane są one do reprezentacji rozwiązań algorytmów. Kubit w danym momencie może reprezentować teoretycznie nieskończenie wiele stanów będących superpozycją stanu 0 i 1. Obserwacja bitu kwantowego pozwala dopiero na jednoznaczną ocenę jego stanu. Stan kubitu może być reprezentowany przez równanie:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{4.1}$$

gdzie $|\alpha|^2$ jest prawdopodobieństwem, że kubit znajduje się w stanie 0, oraz $|\beta|^2$ jest prawdopodobieństwem, że kubit jest w stanie 1. α i β są liczbami zespolonymi. Obie liczby są znormalizowane, co znaczy, że:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{4.2}$$

Kubit jest więc najmniejszą jednostką informacji w tych algorytmach i jest reprezentowany poprzez parę liczb $[\frac{\alpha}{\beta}]$. Podobnie jak w innych algorytmach ewolucyjnych, algorytm NPQGA bazuje na zmienia-

jących się w czasie, dynamicznych populacjach rozwiązań i korzysta z funkcji oceniającej te rozwiązania wykorzystując własności bitów kwantowych. Oprócz stosowania tradycyjnie rozumianych operatorów selekcji, krzyżowania oraz mutacji, autorzy zaproponowali również operator katastrofy, a także operator związany z bramkami kwantowymi, służący do zmiany stanów kubitów.

4.1.1. Kodowanie rozwiązań

W publikacji, został zawarty przykład obrazujący działanie algorytmu dla problemu szeregowania zadań. Został również przedstawiony sposób kodowania rozwiązań, który nadaje się także dla problemu przydziału kwadratowego. Ogólnie, rozwiązanie problemu jest ciągiem kubitów i można je przedstawić w następujący sposób:

$$\left[\begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & ... & \alpha_l \\ \beta_1 & \beta_2 & ... & \beta_l \end{array}\right] \tag{4.3}$$

gdzie

$$l = ([\log_2^n] + 1) \cdot n \tag{4.4}$$

a *n* oznacza rozmiar problemu, czyli iluelementowa jest permutacja reprezentująca rozwiązanie problemu. Nawiasy kwadratowe oznaczają cechę liczby. Niestety, z samego ciągu kubitów nie wynika od razu jaką permutację ten ciąg koduje. By uzyskać rozwiązanie permutacyjne, które jest używane dla problemu QAP, należy wykonać następujące kroki:

1. dla każdego kubitu wylosuj liczbę η z przedziału *[0,1]*,
2. jeśli $\eta < |\alpha_i|^2$, to określ stan *i-tego* kubitu na 0, w przeciwnym przypadku na 1,
3. dla utworzonego ciągu bitów, każde $[\log_2^n] + 1$ bitów zamień na postać dziesiętną,
4. mając ciąg liczb naturalnych posortuj go rosnąco z zapamiętaniem pozycji liczb w ciągu,
5. jeśli dwie kolejne liczby są różne, to mniejsza z nich reprezentuje przydzielony do placówki o numerze indeksu obiekt o mniejszym numerze, a jeśli są równe, to liczba z mniejszym indeksem reprezentuje obiekt o niższym numerze. Elementowi o najmniejszej wartości przyporządkuj obiekt o pierwszym numerze.
6. ustaw rosnąco według indeksów powyższy ciąg liczb naturalnych zastępując te liczby odpowiadającymi im numerami przydzielonych obiektów według zasad z punktu piątego.

W ten sposób uzyskana zostaje permutacja, w której pozycja określa numer lokalizacji, a wartość liczby na tej pozycji, określa przydzielony do niej obiekt.

4.1.2. Operatory genetyczne

Jako, że algorytm NPQGA jest modyfikacją algorytmu genetycznego, w swym działaniu korzysta z typowych operatorów genetycznych. Autorzy algorytmu w zaprezentowanym przykładzie zaproponowali selekcję ruletkową, operator krzyżowania CX, mutację polegającą na zamianie w losowym kubicie parametrów α i β oraz bramkę kwantową do zmiany stanów kubitów o nazwie *rotation gate*.

Pewnym nietypowym rozwiązaniem związanym z krzyżowaniem jest zmiana prawdopodobieństwa zajścia krzyżowania z czasem. Im więcej iteracji algorytmu minęło, tym mniejsze jest prawdopodobieństwo krzyżowania. Należy ustawić jako parametry algorytmu prawdopodobieństwa maksymalne i minimalne zajścia krzyżowania.

$$P_c^+ = \begin{cases} \frac{P_{cmax}}{1+\frac{t}{t_{max}}},\ P_c^+ > P_cmin \\ P_{cmax},\ P_c^+ < P_cmin \end{cases} \tag{4.5}$$

Operator CX działa w następujący sposób:

1. Wybierany jest dowolny element z pierwszego z rodziców, najczęściej jest to pierwszy element permutacji.
2. Sprawdzana jest wartość elementu w drugim rodzicu na pozycji tej samej, co wybrany element w pierwszym rodzicu.
3. Znajdywany jest element w pierwszym rodzicu o wartości sprawdzonej w punkcie 2 i dla tego elementu powtarzamy krok 2.
4. Wykonywane są powyższe kroki, aż do dotarcia w pierwszym rodzicu do punktu startowego.
5. Uzyskane w ten sposób zestawy punktów w obu rodzicach przenoszone są do rozwiązań potomków z zachowaniem indeksów elementów permutacji w taki sposób, że elementy z rodzica pierwszego umieszczane są w potomku nr 2 i na odwrót.
6. Powtarzane jest szukanie punktów poczynając od pierwszego niewybranego punktu w rodzicu pierwszym i znalezione grupy punktów są kopiowane do potomków, lecz tym razem elementy z pierwszego rodzica zostają umieszczone w potomku pierwszym. W następnym wyszukiwaniu ponownie elementy z rodzica pierwszego kopiowane są do potomka drugiego itd.
7. Wyszukiwanie cykli elementów powtarza się aż wszystkie elementy zostaną wybrane.

Mutacja zachodzi wtedy dla danego osobnika, gdy wylosowana dla niego liczba z przedziału *[0,1]* jest mniejsza niż prawdopodobieństwo mutacji p_m. Wtedy losuje się, który kubit z rozwiązania poddany będzie modyfikacji, która wygląda w sposób następujący:

$$\begin{bmatrix} \alpha_i' \\ \beta_i' \end{bmatrix} = \begin{bmatrix} \beta_i \\ \alpha_i \end{bmatrix} \tag{4.6}$$

Po dokonaniu powyższej zamiany, należy ponownie sprawdzić stan kubitu, co może się wiązać ze zmianą liczby dziesiętnej, w której skład wchodzi zmodyfikowany kubit, co dalej może pociągać za sobą zmianę całej permutacji.

Operator bramki kwantowej jest tym elementem algorytmów kwantowych, który ma największy wpływ na zmianę stanów bitów kwantowych. Istnieje wiele różnych odmian bramek kwantowych, takie jak bramka NOT, CNOT, bramka Hadamarda. W algorytmie NPQGA została zaproponowana bramka rotacyjna - *rotation gate*. Uaktualnienie parametrów α i β następuje w następujący sposób:

$$\begin{bmatrix} \alpha_i' \\ \beta_i' \end{bmatrix} = \begin{bmatrix} \cos(\Theta_i) & -\sin(\Theta_i) \\ \sin(\Theta_i) & \cos(\Theta_i) \end{bmatrix} \tag{4.7}$$

Kąt Θ_i określony jest poprzez swoją wartość i kierunek obrotu:

$$\Theta_i = \Delta\Theta_i \cdot s(\alpha_i, \beta_i), \tag{4.8}$$

gdzie $\Delta\Theta_i$ określa wartość kąta o jaki należy dokonać rotacji, a $s(\alpha_i, \beta_i)$ określa kierunek obrotu. Zarówno wartość kąta jak i jego kierunek odczytuje się z tablicy *Look Up* i zależą one od najlepszego rozwiązania znalezionego w danej populacji i wartości parametrów α i β. Sprawdzany jest stan każdego kubitu w rozwiązaniu najlepszym i porównywany ze stanem odpowiadającego mu kubitu w poddawanym działaniu bramki kwantowej rozwiązaniu. Poniżej znajduje się tablica *look up* z wartościami zaproponowanymi przez autorów algorytmu:

r_i	b_i	$f(r) < f(b)$	$\Delta\Theta_i \cdot \pi$	$s(\alpha_i, \beta_i)$			
				$\alpha_i \cdot \beta_i > 0$	$\alpha_i \cdot \beta_i < 0$	$\alpha_i = 0$	$\beta_i = 0$
0	0	False	0.2π	0	0	0	0
0	0	True	0	0	0	0	0
0	1	False	0.5π	0	0	0	0
0	1	True	0	-1	+1	+1 lub -1	0
1	0	False	0.5π	-1	+1	+1 lub -1	0
1	0	True	0	+1	-1	0	+1 lub -1
1	1	False	0.2π	+1	-1	0	+1 lub -1
1	1	True	0	+1	-1	0	+1 lub -1

Tablica 4.1: LUT dla bramki kwantowej

W przypadku gdy stany porównywanych kubitów są różne i wybrane rozwiązanie jest gorsze niż najlepsze dotychczas, proponowana jest zmiana o większy kąt, a gdy mają taką samą wartość, to zaleca się kąt o mniejszej wartości. Kierunek obrotu zależy od iloczynu prawdopodobieństw, że kubit znajduje się w stanie *0* i *1*. W przypadku problemu szeregowania zadań z minimalizacją czasu wykonań wszystkich

z nich, rozwiązanie lepsze ma mniejszą wartość funkcji celu, dlatego kąt zmieniany jest, gdy w trzeciej kolumnie tabeli znajduje się wartość *False*. Celem działania operatora *rotation gate* jest modyfikacja rozwiązań w danym pokoleniu, dzięki której będą one bardziej podobne do rozwiązania najlepszego. Rozwiązanie poddane działaniu tego operatora z większym prawdopodobieństwem będzie podobne do osobnika najlepszego.

Autorzy algorytmu zaproponowali również tak zwany operator katastrofy. Jest on wykorzystywany w sytuacji, w której nie uzyskiwana jest poprawa rozwiązania podczas określonej liczby iteracji algorytmu i skutkuje ponownym zainicjowaniem populacji. Zakłada się, że zdarzenie to spowodowane jest znalezieniem ekstremum lokalnego.

4.1.3. Równoległość algorytmu

Ciekawym elementem algorytmu jest sposób w jaki dokonywany jest przegląd rozwiązań. Zaproponowany został model, w którym istnieje wiele równoległych populacji pogrupowanych w tak zwane uniwersa. W jednym uniwersum znajduje się wiele populacji. Wymiana informacji obydwa się na dwóch poziomach:

1. pomiędzy uniwersami,
2. pomiędzy populacjami w danym uniwersum.

Wymiana informacji w obrębie jednego uniwersum wzorowana jest na osmozie. Informacje o dobrych rozwiązaniach wędrują w jedną stronę,w kierunku populacji, dla której suma dopasowania rozwiązań jest gorsza. Natomiast wymiana informacji pomiędzy uniwersami bazuje na czasowej zmianie wartości, do której dążą rozwiązania w innych uniwersach. Po tej wymianie, rozwiązania z jednego uniwersum jako cel swego rozwoju obierają optymalny kierunek innego i na odwrót. Obie powyższe strategie zachodzą z ustalaną częstotliwością. Autorzy podają *10%-20%* wszystkich iteracji jako typową wartość tego parametru.

4.2. Pseudokod algorytmu NPQGA

Poniżej znajduje się pseudokod algorytmu:

Wczytaj parametry algorytmu;
Zainicjuj populacje, wyznacz ich permutacyjna postać, policz ich dopasowanie, zapisz najlepszy rezultat;
while *nie wystąpił warunek stopu* **do**
  $t \leftarrow t + 1$;
  for *dla każdej populacji* **do**
    Wybierz najlepsze rozwiązanie z populacji;
    Dokonaj krzyżowania i mutacji;
    if *zaistniały warunki dla operatora katastrofy* **then**
      Użyj operatora katastrofy do wygenerowania następnego pokolenia;
    end
    else
      Użyj bramki kwantowej do wygenerowania następnego pokolenia;
    end
  end
  for *dla każdej populacji w każdy uniwersum* **do**
    Dokonaj wymiany informacji między populacjami;
  end
  for *dla każdego uniwersum* **do**
    Dokonaj wymiany informacji między uniwersami;
  end
end

Algorithm 4: Algorytm NPQGA

Powyższy schemat działania odnosi się do wersji algorytmu wykorzystanej przez jego autorów do rozwiązania problemu szeregowania zadań. Jednym z celów niniejszej pracy było stworzenie aplikacji wykorzystującej wybrany algorytm przybliżony do rozwiązywania problemu przydziału kwadratowego. Powyższy algorytm został zaimplementowany, lecz z pewnymi modyfikacjami. Zostały one przedstawione w następnym rozdziale.

5. Modyfikacja algorytmu NPQGA

Przestawiony w poprzednim rozdziale algorytm NPQGA nadaje się do rozwiązywania problemu przydziału kwadratowego, lecz w kontekście stworzenia aplikacji, będącej jednym z celów niniejszej pracy, zostały wprowadzone liczne modyfikacje, a także algorytm został uzupełniony o dodatkowe, nie zaproponowane przez jego autorów, cechy. Wszystkie opisane zmiany, a także dodane funkcjonalności zostały zaproponowane i uzgodnione z opiekunem pracy i zostaną przedstawione w tym rozdziale. Jednakże, główna cecha algorytmu, jaką jest wykorzystanie bitów kwantowych do reprezentacji rozwiązań problemu, pozostała bez zmian. Kodowanie osobników wygląda tak samo, jak zostało to opisane w rozdziale poprzednim.

5.1. Lista zmian i modyfikacji

5.1.1. Nierównoległa wersja algorytmu

Główną zmianą wprowadzoną do implementowanego algorytmu jest zrezygnowanie z równoległej wersji algorytmu. Istnieje zatem tylko jedna populacja, która ewoluuje razem z kolejnymi iteracjami algorytmu. Implikuje to również brak potrzeby wymiany informacji pomiędzy uniwersami, a także pomiędzy populacjami w każdym z uniwersów.

5.1.2. Operator katastrofy

W związku z rezygnacją z wielu populacji rozwijanych równolegle, zaniechano również korzystania z operatora katastrofy. Jego stosowanie mogłoby powodować utratę wypracowanego z czasem dobrego rozwiązania, jeśli to nie zmieniałoby się od ustalonej liczby pokoleń. W przypadku wielu równolegle ewoluujących populacji, użycie tego operatora mogłoby pozwolić na wyjście z lokalnego minimum w danej populacji, lecz w przypadku jednej, powodowałoby to utratę jedynego znalezionego rozwiązania i rozpoczęcie szukania optimum od początku. W sytuacji, gdy operator zostałby użyty pod koniec wykonywania algorytmu, szukane od nowa rozwiązanie, w związku z małą ilością pozostałych iteracji, mogłoby odbiegać bardzo od faktycznego optimum.

5.1.3. Operatory selekcji

W zmodyfikowanej wersji algorytmu zostały wykorzystane dwie wersje operatora selekcji. Pierwsza z nich polega na selekcji ruletkowej. Funkcja dopasowania rozwiązań została uzyskana z funkcji celu w następujący sposób:

$$f(x_i) = F_{cmax} - F(x_i) \tag{5.1}$$

gdzie $f(x_i)$ jest funkcją dopasowania *i-tego* rozwiązania w pokoleniu, F_{cmax} jest wartością funkcji celu dla najgorszego rozwiązania w danym pokoleniu, czyli o największym koszcie przydziału, a $F(x_i)$ jest funkcją celu *i-tego* rozwiązania w pokoleniu. W ten sposób funkcja dopasowania jest zawsze nieujemna, a większa jej wartość oznacza, że rozwiązanie ma lepsze dopasowanie. Następnie w oparciu o wartości dopasowania rozwiązań, w standardowy sposób, budowane jest koło ruletki.

Drugim z operatorów jest operator selekcji bazujący na rankingu rozwiązań. Rozwiązania w aktualnym pokoleniu są szeregowane rosnąco według wartości funkcji celu, czyli rozwiązania o mniejszej wartości funkcji celu mają niższy indeks na liście, czyli mają w rankingu wyższe miejsce. Następnie w oparciu o ranking budowana jest funkcja, której wartość określa prawdopodobieństwo wyboru danego rozwiązania podczas selekcji. Istnieje wiele wariantów tych funkcji. W zaimplementowanym operatorze selekcji rankingowej wykorzystano funkcję liniową o poniższym wzorze na prawdopodobieństwo wyboru rozwiązania na *i-tej* pozycji w rankingu:

$$p_s(x_i) = \frac{1}{n}(\eta_{max} - (\eta_{max} - \eta_{min})\frac{i-1}{n-1}) \tag{5.2}$$

gdzie *n* jest rozmiarem populacji, a parametry η_{min} i η_{max} są określone następująco:

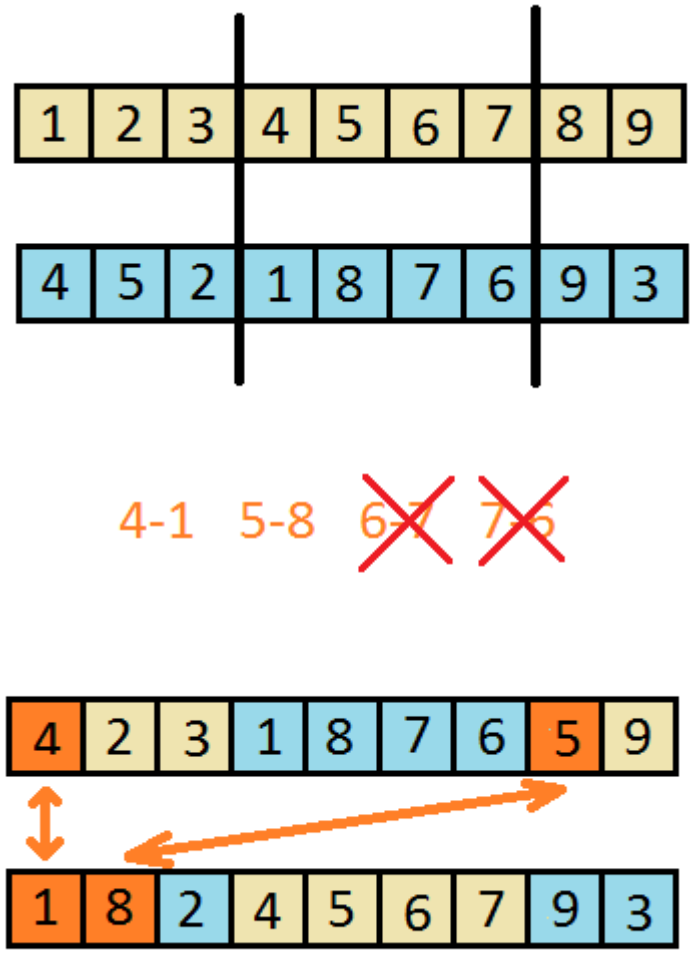
$$\eta_{min} = 2 - \eta_{max},\ 1 \leq \eta_{max} \leq 2. \tag{5.3}$$

Ustawienie parametru η_{max} na 1 powoduje, że prawdopodobieństwo wyboru danego rozwiązania jest takie samo jak dla pozostałych w populacji, natomiast ustawienie na wartość 2 powoduje, że różnice w prawdopodobieństwach są maksymalne z korzyścią na rzecz rozwiązań lepiej dopasowanych

5.1.4. Operatory krzyżowania

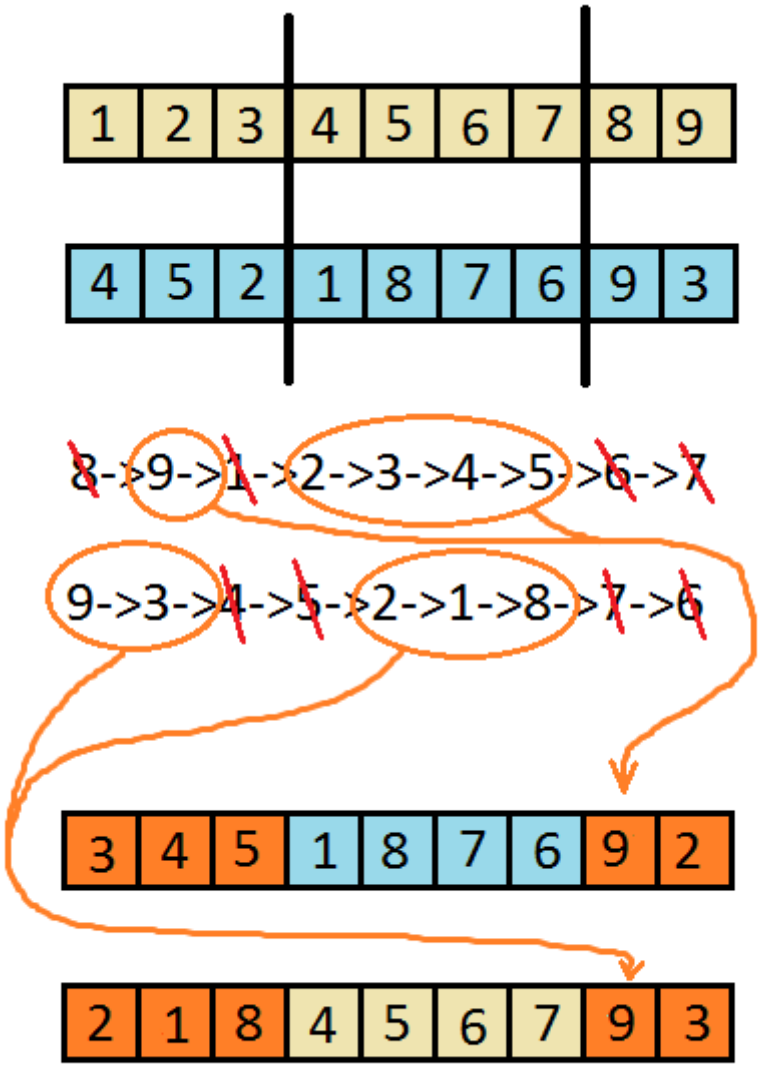
Oprócz proponowanego przez autorów operatora krzyżowania cyklicznego, zostały również wykorzystane operatory PMX oraz OX. Operator PMX, czyli operator krzyżowania z częściowym odwzorowaniem, polega na zamianie w wybranym fragmencie genów pomiędzy rodzicami i utworzeniu na tej podstawie listy odwzorowań. W przypadku odwzorowań *a - b* i *b - c*, oba odwzorowania redukuje się do postaci *a - c*, natomiast w przypadku występowania cyklu, tworzące ten cykl odwzorowania pomija

się. Elementy spoza wybranego fragmentu są wymieniane na zasadzie element za element, jeśli taka wymiana znajduje się na liście z odwzorowaniami, a pozostałe elementy w osobnikach przepisywane są bez zmian. Poniżej znajduje się schemat z przykładowym działaniem tegoż operatora:



Rysunek 5.1: Krzyżowanie PMX

Podczas działania operatora OX, czyli operatora z zachowaniem porządku, wybierane są dwie pozycje genów z rozwiązań rodziców i spomiędzy tych pozycji kopiowane są geny z rodzica pierwszego do potomka pierwszego i z rodzica drugiego do potomka drugiego. Następnie, począwszy od pierwszej pozycji za kopiowanym fragmentem, przenoszone są geny z rodzica pierwszego do rozwiązania potomnego nr 2, z wyłączeniem elementów już się w nim znajdujących i na odwrót, czyli geny rodzica drugiego przenoszone są do potomka pierwszego. Przeniesione elementy są również umieszczane od pierwszej pozycji za skopiowanym fragmentem. Poniżej znajduje się rysunek obrazujący działanie tego operatora:



Rysunek 5.2: Krzyżowanie OX

Zgodnie z założeniem autorów algorytmu, prawdopodobieństwo zajścia krzyżowania powinno zmniejszać się wraz z rosnącą liczbą iteracji algorytmu. W zmodyfikowanej wersji algorytmu możliwe jest ustawienie maksymalnego i minimalnego prawdopodobieństwa na tę samą wartość, co skutkuje niezmiennym prawdopodobieństwem krzyżowania podczas działania algorytmu.

Jeśli spośród wybranych na drodze działania operatora selekcji do krzyżowania zostanie przeznaczone mniej rozwiązań niż wynosi rozmiar populacji, z wybranych rozwiązań losowo kopiowane są brakujące osobniki i wstawiane są do listy przeznaczonych do krzyżowania rozwiązań w losowe miejsca. Następnie każde dwa kolejne rozwiązania na liście rodziców poddawane są wybranemu sposobowi krzyżowania i w ten sposób otrzymywane jest następne pokolenie rozwiązań, zastępujące poprzednie.

5.1.5. Operator bramki kwantowej

Oprócz przedstawionych wartości w tabeli 4.1.2 zaproponowana została druga wersja tablicy *Look Up*. Kąt zmieniany jest tylko w przypadku, gdy stany porównywanych kubitów są różne. W przypadku, gdy dopasowanie zapamiętanego rozwiązania najlepszego jest mniejsze niż poddawanego działaniu bramki kwantowej, zmiana kąta powoduje zwiększenie prawdopodobieństwa, że dany kubit pozostanie w swoim aktualnym stanie i wartość, o którą zmieniany jest kąt jest stosunkowo mała. W sytuacji, gdy dopasowanie rozwiązania najlepszego jest rzeczywiście lepsze, zmiana kąta następuje w kierunku zwiększenia prawdopodobieństwa, że kubit znajdzie się w stanie takim samym jak porównywany z nim odpowiadający mu kubit z rozwiązania najlepszego. W tym przypadku kąt jest zmieniany o wartość dużo większą niż w poprzedniej sytuacji. Zarówno jednak wartości, o które oba kąty są zmieniane, są dużo mniejsze niż wartości, które zostały zaproponowane w tabeli 4.1.2.

Poniżej znajduje się alternatywna wersja tablicy *Look Up*:

r_i	b_i	$f(r) < f(b)$	$\Delta\Theta_i \cdot \pi$	$s(\alpha_i, \beta_i)$			
				$\alpha_i \cdot \beta_i > 0$	$\alpha_i \cdot \beta_i < 0$	$\alpha_i = 0$	$\beta_i = 0$
0	1	False	0.08π	+1	-1	0	+1 lub -1
0	1	True	0.001π	-1	+1	+1 lub -1	0
1	0	False	0.08π	-1	+1	+1 lub -1	0
1	0	True	0.001π	+1	-1	0	+1 lub -1

Tablica 5.1: Alternatywna LUT dla bramki kwantowej

W nieuwzględnionych przypadkach nie następuje aktualizacja kąta. Dotyczy to sytuacji, gdy zarówno kubit z rozwiązania najlepszego i aktualnie poddawanego działaniu bramki kwantowej, znajdują się w tym samym stanie, niezależnie od sytuacji, które z rozwiązań jest lepsze.

5.1.6. Pozostałe zmiany

W stosunku do algorytmu w postaci proponowanej w publikacji [20], wprowadzone zostały jeszcze dwie zmiany. Pierwszą z nich jest ustawianie wartości prawdopodobieństwa krzyżowania na wartość P_{cmin} w przypadku, gdy wyliczana w każdej iteracji jego wartość spadnie poniżej wartości P_{cmin}. Druga modyfikacja związana jest z dekodowaniem stanów kubitów. Wedle autorów algorytmu, stan kubitu powinien być ustawiony na wartość 1, gdy wylosowany parametr η jest mniejszy niż wartość $|\alpha|^2$, co nie jest prawdą, gdyż $|\alpha|^2$ określa prawdopodobieństwo, że kubit znajduje się w stanie 0.

6. Aplikacja rozwiązująca problem przydziału kwadratowego z wykorzystaniem kwantowego algorytmu ewolucyjnego

Jednym z celów realizowanej pracy dyplomowej było napisanie aplikacji, która przy wykorzystaniu jednego z algorytmów aproksymacyjnych będzie rozwiązywać problem przydziału kwadratowego. Wybór padł na kwantowy algorytm ewolucyjny NPQGA opisany w rozdziale czwartym. Uwzględnione w nim zmiany oraz modyfikacje zostały przedstawione w piątym rozdziale. Program został zrealizowany jako aplikacja konsolowa i napisany w języku C#. O tym, że aplikacja będzie konsolowa zadecydował fakt, iż jej celem nadrzędnym jest znajdywanie rozwiązania problemu QAP i ma służyć jako narzędzie pozwalające zweryfikować działanie algorytmu. Aspekty wizualne są jedynie dodatkiem, który nie wpływa na jakość rozwiązania problemu.

6.1. Interfejsy klas

Punktem wyjścia do rozpoczęcia prac był zestaw interfejsów klas przygotowanych przez opiekuna niniejszej pracy. Interfejs klasy w sensie języka C# jest narzędziem wykorzystywanym w technice dziedziczenia i określa metody i właściwości, jakie klasa dziedzicząca po nim musi implementować. Jednakże ciała metod nie są określone i zależą wyłącznie od implementacji w danej klasie. W przeciwieństwie do dziedziczeniu po klasach, istnieje możliwość dziedziczenia po wielu interfejsach. Dzięki wykorzystaniu interfejsów, napisane na potrzeby pracy klasy będzie można wykorzystać w innych aplikacjach :już istniejących, ale i w tych, które dopiero powstaną. Poniżej znajduje się lista interfejsów, które należało zaimplementować:

1. IEvolutionAlgorithm,
2. IOptimisationAlgorithm,
3. IPopulation,
4. ISolution,
5. IEvolutionaryOperator,
6. IMutationOperator,

7. ICrossoverOperator.

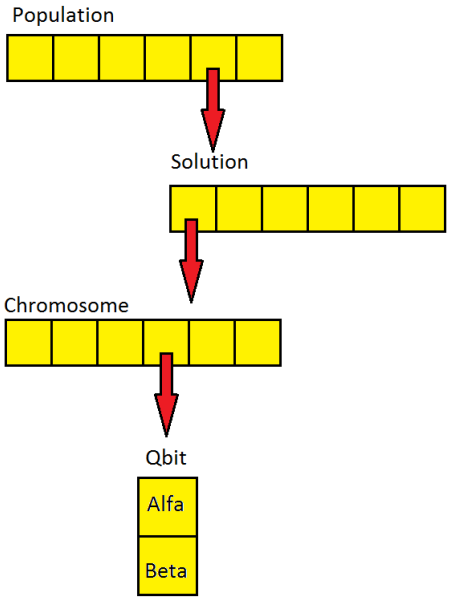
Interfejs *IEvolutionAlgorithm* dziedziczy po interfejsie *IOptimisationAlgorithm*. Na bazie tych interfejsów powstała klasa *QgAlgorithm* zawierająca metody pozwalające na ustawienie i uruchomienie algorytmu oraz pozwalająca na zwrócenie rezultatów działania algorytmu.

Interfejsy *ICrossoverOperator* i *IMutationOperator* dziedziczące po interfejsie *IEvolutionaryOperator* posłużyły jako podstawa, dla stworzenia klas realizujących zadania operatorów krzyżowania oraz mutacji. Na podstawie interfejsu *ICrossoverOperator* powstała również klasa reprezentująca operator bramki kwantowej.

Interfejsy *IPopulation* i *ISolution* posłużyły do napisania klas reprezentujących odpowiednio pojedyncze rozwiązanie algorytmu i populację algorytmu.

6.2. Struktura danych

By algorytm mógł znaleźć rozwiązanie problemu, musi przetwarzać całą populację rozwiązań. Dlatego też została napisana klasa *Population*, dziedzicząca po interfejsie *IPopulation*. Populacja natomiast zawiera w sobie listę osobników, rozwiązań, które reprezentowane są przez obiekty klasy *Solution* dziedziczącej z kolei po interfejsie *ISolution*. Postacią rozwiązania w problemie przydziału kwadratowego jest permutacja, określająca, który obiekt został przypisany do kolejnych lokalizacji. Z tego powodu obiekty klasy *Solution* zawierają w sobie listę obiektów reprezentujących elementy permutacji. Klasa tych obiektów została nazwana *Chromosome*. Należy tutaj wyjaśnić pewną nieścisłość w terminach używanych w algorytmach genetycznych. *Chromosomami* zwykło się nazywać kompletne rozwiązania, a ich fragmenty kodujące rozwiązanie *genami*. Z racji, iż w aplikacji rozwiązania problemu reprezentowane są przez obiekty klasy *Solution*, a element permutacji nie jest najmniejsza porcją informacji, postanowiono nazwać te porcje chromosomami, a ich najmniejszą część *genem*. A więc *chromosomy* składają się z kubitów reprezentowanych przez klasę *Qbit*. Poprzez zdekodowanie stanu kubitów określa się wartość chromosomu, a następnie poprzez omówioną w rozdziale czwartym konwersję otrzymuje się permutacyjną postać rozwiązania problemu QAP.



Rysunek 6.1: Struktura danych

W kontekście struktury danych należy jeszcze wspomnieć o macierzach przepływu i odległości problemu QAP. Ich wartości są wczytywane z plików *.dat* zawierających rozmiar problemu oraz odpowiednio macierze odległości i przepływu i są następnie trzymane w obiekcie klasy *QapData* zrealizowanej jako singleton, czyli jako specjalna klasa o globalnym dostępnie pozwalająca na utworzenie tylko jednego jej obiektu. Posiada ona także mechanizmy, które bez wiedzy użytkownika same dbają o to, by faktycznie istniał jeden jej obiekt i jej instancja została utworzona podczas pierwszej próby jej użycia.

6.3. Interfejs użytkownika

Po uruchomieniu aplikacji użytkownik ma możliwość ustawienia kilku parametrów algorytmu. Należą do nich:

– wybór instancji problemu QAP,

– rozmiar populacji, czyli ilość rozwiązań w populacji,

– ilość iteracji algorytmu,

– wybór sposobu selekcji rozwiązań,

– wartość parametru η_{max} w przypadku selekcji rankingowej,

– minimalne i maksymalne prawdopodobieństwo krzyżowania,

– operator krzyżowania,

– prawdopodobieństwo mutacji,

– wybór, czy stosować bramkę kwantową,

– wersja bramki kwantowej,

– wybór, czy otrzymane w wyniku działania algorytmu rezultaty i ustawione wartości parametrów zapisać do pliku,

– nazwa pliku, do którego zostaną zapisane powyższe informacje.

W przypadku operatora selekcji do wyboru są dwie metody: ruletkowa oraz rankingowa. Spośród metod krzyżowania dostępne są trzy opcje: CX, OX i PMX. Wersje bramki kwantowej zostały opisane w rozdziałach 4 i 5.

Aplikacja została napisana w sposób niepozwalający na przyjęcie nieprawidłowych danych.

Po zakończeniu zwracane są rezultaty działania algorytmu i możliwe jest ponowne uruchomienie algorytmu. Użytkownik ma możliwość wyboru spośród trzech opcji:

1. ponowne uruchomienie dla tych samych parametrów i tej samej populacji początkowej,

2. ponowne uruchomienie dla tej samej populacji, lecz dla nowych wartości parametrów, z oczywistym pominięciem ustawiania rozmiaru populacji a także wyboru instancji testowej,

3. wygenerowanie nowej populacji i ustawienie nowych wartości parametrów.

Opcja pierwsza pozwala sprawdzić różnice w otrzymanych rezultatach działania algorytmu przy tych samych parametrach. W tej sytuacji można sprawdzić, jaki wpływ na efekty końcowe ma czynnik losowy algorytmu.

Druga opcja pozwala na porównanie wpływu różnych wartości konkretnych parametrów na uzyskiwane rezultaty.

Opcja trzecia po prostu pozwala na ponowne uruchomienie algorytmu.

6.4. Funkcja testująca parametry

Oprócz opisanego wyżej interfejsu użytkownika została napisana również funkcja pozwalająca na przetestowanie zmieniającej się wartości wybranego parametru przy ustalonych pozostałych parametrach. Użytkownik podaje, który parametr chce przetestować, określa dla ilu jego wartości chce przeprowadzić testy oraz ile razy powtórzyć dany test, a także podaje, dla których instancji testowych przeprowadzić eksperymenty. Następnie należy określić wartości pozostałych parametrów, które będą stałe dla każdego z testów. Wymagane jest również podanie szablonu nazwy plików, w których zapisane są rezultaty eksperymentów. Cała nazwa pliku tworzona jest w następujący sposób sposób:
[szablon]_[parametr testowany]_[wartość parametru]_[instancja testowa]_[nr testu].

6.5. Rezultaty działania aplikacji

Po wykonaniu wszystkich iteracji algorytmu, program zwraca najlepsze znalezione rozwiązanie wraz z wartością funkcji celu oraz informację o tym, w której iteracji zostało zwrócone rozwiązanie. Zwracane jest również najlepsze rozwiązanie wraz z jego wartością funkcji celu uzyskane w ostatniej iteracji algorytmu.

Jeśli została wybrana opcja zapisu do pliku, tworzone są dwa pliki: w pierwszym z nich zapisane są ustawione parametry algorytmu wraz z datą uruchomienia algorytmu. W pliku zostaje też zawarta informacja o wartości najlepszego rozwiązania w ogóle wraz z numerem iteracji, w której zostało znalezione, a także jego postać. Podobnie, zawarta jest również informacja o najlepszym rozwiązaniu znalezionym w ostatniej iteracji. Drugi zawiera 3 kolumny danych zawierających odpowiednio numer iteracji, wartość najlepszego rozwiązania w danej iteracji i średnią wartość wszystkich rozwiązań w populacji. Zapis w tym formacie pozwala na późniejszy łatwy import danych do innych aplikacji pozwalających np. na utworzenie wykresów.

6.6. Szczegóły związane z implementacją poszczególnych elementów algorytmu

Kilka szczegółów związanych z działaniem algorytmu nie zostało poruszonych w publikacji [20]. Jednakże , by algorytm mógł działać, należało arbitralnie przyjąć pewne założenia. Jednym z takich założeń jest moment aktualizacji stanu kubitów. Zostało przyjęte, że należy ocenić, w którym ze stanów 1 lub 0 znajduje się bit kwantowy, gdy wiadome jest, że rozwiązanie, w którym dany kubit się znajduje, poddane zostało zmianom wynikającym z działania operatora bramki kwantowej, lub dla tego rozwiązania zostały spełnione warunki na zajście mutacji. Natomiast po dokonaniu operacji krzyżowania, nie jest oceniany stan kubitów rozwiązań potomnych. Taka ocena, szczególnie w początkowych iteracjach algorytmu powodowałaby, że rozwiązania otrzymane na drodze krzyżowania mogłyby tracić informację uzyskaną z rozwiązań rodziców. Ideą krzyżowania jest wymiana informacji.

W przypadku, gdy użytkownik programu postanowi, że algorytm ma znajdywać rozwiązanie bez wykorzystania bramki kwantowej, kwantowa idea algorytmu przejawia się wtedy jedynie podczas tworzenia populacji początkowej oraz operacji mutacji, która dalej polega na zamianie wartości parametrów α i β.

Rozwiązanie, które jest podstawą do działania bramki kwantowej i nazywane jest najlepszym, jest faktycznie najlepszym rozwiązaniem, ale uzyskanym w poprzedniej iteracji. Operator bramki kwantowej używany jest dopiero na populacji poddanej już działaniu operatorów krzyżowania i mutacji. Z tego powodu istnieje możliwość, że dopasowanie zapamiętanego rozwiązania będzie gorsze niż dopasowanie niektórych rozwiązań poddawanych działaniu bramki kwantowej. Dlatego rozważane są w tablicach *Look Up* przypadki, dla których warunek $f(x) > f(best)$ może być prawdą.

Foldery z rezultatami działania algorytmu - *Results*i instancjami testowymi - *QAPLib* znajdują się w tym samym folderze co plik wykonywalny aplikacji. Podczas tworzenia obiektu klasy *QgAlgorithm* sprawdzane jest czy te foldery istnieją i jeśli nie, to są tworzone.

Próba przeprowadzenia testów, gdy folder z instancjami testowymi jest pusty, skutkuje zakończeniem działania funkcji interfejsu użytkownika i testującej parametry wraz z pojawieniem się na ekranie krótkiej informacji informującej, że folder jest pusty.

7. Metodyka eksperymentów

Możliwość ustawienia wielu różnych wartości parametrów zaimplementowanego w napisanej aplikacji algorytmu pozwala na przeprowadzenie serii eksperymentów badających ich wpływ na otrzymywane rezultaty. Również wprowadzenie wielu modyfikacji do samego algorytmu powoduje, że należy sprawdzić, czy te zmiany powodują istotne różnice w kwestii rozwiązywania problemu. Być może wprowadzone zmiany powodują, że algorytm znajduje rozwiązania niezgodne z oczekiwaniami, jego zbieżność uległa pogorszeniu. Z wymienionych przyczyn została przeprowadzona znaczna ilość testów z wykorzystaniem ogólnodostępnych instancji testowych weryfikujących poprawność ogólnego działania algorytmu oraz badających wpływ poszczególnych parametrów algorytmu na jego działanie.

7.1. Instancje testowe

Popularność problemu QAP powoduje, że istnieje cała gama instancji testowych problemu z podanym optymalnym rozwiązaniem, lub najlepszym znalezionym rozwiązaniem dotychczas. Wykorzystane w eksperymentach instancje testowe pochodzą ze strony [adres] i są w postaci plików **.dat* zawierających rozmiar problemu i odpowiednio macierze odległości i przepływu. W przypadku, gdy jest inaczej, instancja opatrzona jest stosownym komentarzem. Podawane w instancji rozwiązania są permutacjami, których elementy reprezentują przydzielone obiekty, a ich pozycja reprezentuje lokalizacje, do której zostały przydzielone.

Do testów zostały wybrane trzy konkretne instancje o różnym rozmiarze, wszystkie ze znanym optymalnym rozwiązaniem. Wybór instancji ze znanym rozwiązaniem podyktowany był tym, że znajomość optimum pozwala na lepszą analizę otrzymanych rezultatów oraz wybranych ustawień algorytmu.

Pierwszą wybraną instancją jest instancja *Had12*. Jej autorami są S.W. Hadley, F. Rendl i H. Wolkowicz. Nazwa instancji pochodzi od nazwiska pierwszego z autorów i rozmiaru (12). Macierz odległości reprezentuje odległości pomiędzy połączonymi obiektami na Mahattanie, natomiast macierz przepływu jest wygenerowana na podstawie równomiernego rozkładu na przedziale *[1,12]*. Znane jest optymalne rozwiązanie i jego wartość wynosi 1652, a jego postać wygląda w następujący sposób:
(3, 10, 11, 2, 12, 5, 6, 7, 8, 1, 4, 9).

Kolejną wybraną instancją jest problem o nazwie *Lipa20a* autorstwa Y. Li i P.M. Pardalos. Podobnie jak powyżej, nazwa instancji pochodzi od jej rozmiaru i nazwisk autorów. Instancja ta nie ma konkretnego przełożenia na rzeczywistość, została wygenerowana w generatorze problemów. Optymalna war-

tość dla tego problemu to 3683, a postać rozwiązania jest następująca:
(19, 17, 7, 1, 5, 9, 10, 12, 4, 16, 20, 6, 3, 14, 11, 15, 13, 8, 2, 18).

Ostatnią, trzecią z wybranych instancji jest instancja *Kra32*. Jej autorami są J. Krarup i P.M. Pruzan, a zawarte w niej dane są związane z rzeczywistymi danymi wykorzystanymi podczas projektowania kliniki w Regensburgu w Niemczech. Optymalna wartość wynosi 88700, a postać rozwiązania optymalnego jest następująca:
(31, 23, 18, 21, 22, 19, 10, 11, 15, 9, 30, 29, 14, 12, 17, 26, 27, 28, 1, 7, 6, 25, 5, 3, 8, 24, 32, 13, 2, 20, 4, 16).

7.2. Scenariusze testowe

Z racji tego, iż stosunkowo duża ilość parametrów algorytmu może być zmieniana, nieraz w szerokim zakresie, nie jest możliwe przetestowanie wszystkich możliwych wariantów i porównania uzyskanych dla tych wariantów wyników. Dlatego skupiono się przede wszystkim na badaniu różnic w otrzymywanych rezultatach, gdy zmieniany był tylko jeden parametr. Poprzez wielokrotne uruchamianie algorytmu dla tej samej wygenerowanej populacji początkowej i dla tych samych parametrów uzyskiwane były uśrednione rezultaty. Algorytm z powodu swojej niedeterministycznej natury, zawierającej element losowości, nie pozwala na uzyskanie tych samych rezultatów przy tych samych ustawieniach przy wielokrotnym uruchamianiu. Dlatego więc uśrednianie wyników z wielu prób jest uzasadnione. W skrajnych przypadkach otrzymany wynik może informować o bardzo dobrym działaniu algorytmu, w innym o fatalnym. Algorytmy przybliżone stosowane są przede wszystkim tam, gdzie próby rozwiązania problemu metodami dokładnymi z góry skazane są na niepowodzenie. W przeciwieństwie do eksperymentów, nie jest znane rozwiązanie problemu (gdyby było, jaki byłby cel stosowania tych algorytmów). Poszukiwanie rozwiązania jest wtedy procesem składającym się z wielu prób. Stosowana metodyka testów oddaje więc charakter faktycznych poszukiwań rozwiązania postawionego problemu. Analiza rezultatów eksperymentów pozwala na dobór odpowiednich ustawień parametrów algorytmu, przyczyniających się do efektywnego rozwiązywania problemów podobnych, dla których rozwiązanie nie jest znane. Każdy z opisanych niżej testów polegał na zmianie pewnego jednego testowanego parametru i powtarzany był dla każdej z instancji. Otrzymywane wyniki były porównywane ze sobą, z uwzględnieniem tego, dla której testowej instancji problemu były przeprowadzone. Uśrednione wyniki otrzymywane były na podstawie dziesięciu powtórzeń testu. Z racji tego iż, testowany algorytm w swoich założeniach wykorzystuje bramkę kwantową, poza pierwszą grupą testów, bramka ta zawsze była wykorzystywana, w wersji, która lepiej wypadła w pierwszym eksperymencie.

7.2.1. Bramka kwantowa

Jeszcze w trakcie pisania aplikacji, pojawiło się wiele wątpliwości związanych z operatorem bramki kwantowej. Idea jej stosowania wydaje się być wprawdzie słuszna - odpowiednia modyfikacja osobników w populacji zwiększająca ich prawdopodobieństwo upodobnienia się ich do rozwiązania lepszego

od nich i „utwierdzenia" ich w swojej postaci, gdy są lepsze niż porównywane z nimi rozwiązanie gorsze. Jednakże, wykorzystanie jej w praktyce, w postaci proponowanej w publikacji [20], często skutkuje otrzymywaniem rezultatów złych. Pojawiła się więc wątpliwość, czy na pewno operator ten jest dobry. Jego stosowanie wydaje się być jednak wymagane - poza mutacją, która teoretycznie zachodzi dość rzadko, operator ten ma możliwość zmiany stanu bitów kwantowych, które są główną cechą algorytmów ewolucyjnych. Bez operatora bramki kwantowej algorytm staje się praktycznie typowym algorytmem genetycznym, w którym jego jedyny związek z kwantowym podejściem do problemu widoczny jest w operatorze mutacji. Z tego powodu została zaproponowana druga wersja bramki rotacyjnej, opisana w rozdziale 5. Seria eksperymentów związanych z wpływem bramki kwantowej na otrzymywane rezultaty miała więc na celu porównania trzech wariantów poszukiwania rozwiązania:

1. bez wykorzystania bramki kwantowej,

2. z wykorzystaniem bramki kwantowej w wersji proponowanej przez autorów algorytmu NPQGA,

3. z wykorzystaniem bramki kwantowej w wersji proponowanej przez autora niniejszej pracy.

7.2.2. Operator selekcji

Każda z dwóch zaimplementowanych metod selekcji ma swoje dobre i złe strony. Należało jednak sprawdzić, która z metod bardziej przydatna jest w przypadku postawionego problemu. Dodatkowo metoda rankingowa pozwala na ustawienie parametru η, dodatkowo więc należało sprawdzić wpływ tego parametru na uzyskiwane wyniki. Przeprowadzona została więc seria testów porównujących działania obu operatorów oraz seria eksperymentów porównujących działanie operatora selekcji rankingowej z różnymi wartościami parametru η: wartością minimalną, maksymalną i pośrednią.

7.2.3. Operator krzyżowania

Każdy z trzech zaimplementowanych operatorów krzyżowania (CX, OX, PMX) ma inne działanie. W związku z tym krzyżowanie przeprowadzone przy wykorzystaniu jednego z operatorów powinno dać rezultaty inne niż dla innego operatora. Przebieg działania algorytmu genetycznego, uruchomionego dla tych samym parametrów przeważnie wygląda inaczej, nawet jeśli zwracane jest to samo rozwiązanie. W związku z tym, ciężko jest wprost porównać, który z operatorów krzyżowania działa lepiej lub gorzej. Jednakże w przypadku wielokrotnego powtórzenia tego samego testu z wykorzystaniem jednego operatora krzyżowania i uśrednienia wyników, może pozwolić na dostrzeżenie pewnych różnic pomiędzy rezultatami analogicznych testów z wykorzystaniem innego operatora. Z tego powodu testy dotyczące porównania działania operatorów krzyżowania polegały na wielokrotnym uruchamianiu algorytmu dla tych samych parametrów, populacji początkowej i z różnymi wybranymi operatorami krzyżowania.

7.2.4. Prawdopodobieństwo krzyżowania

Algorytm zaimplementowany w aplikacji pozwala na ustawienie minimalnego i maksymalnego prawdopodobieństwa krzyżowania. Ustawienie obu parametrów na tę samą wartość powoduje, że prawdopodobieństwo krzyżowania ma stałą wartość podczas całego działania algorytmu. Należało więc sprawdzić jaki wpływ na otrzymywane rezultaty mają różne wartości stałego prawdopodobieństwa, a także czy zmieniająca się jego wartość w trakcie działania algorytmu powoduje istotne zmiany w efektywności poszukiwania optimum. Testy polegały więc, na porównaniu rezultatów otrzymanych dla małej, średniej i dużej wartości prawdopodobieństwa, a także na porównaniu wyników otrzymanych przy stałym i zmieniającym się prawdopodobieństwie krzyżowania. Przyjęto, że stała wartość prawdopodobieństwa jest średnią arytmetyczną wartości minimalnej i maksymalnej prawdopodobieństwa w teście ze zmiennym prawdopodobieństwem.

7.2.5. Prawdopodobieństwo mutacji

Przyjęło się mówić, że operator mutacji ma drugorzędne znaczenie w kwestii poszukiwania rozwiązań w algorytmie genetycznym. Jednakże natura tego operatora w kontekście zaimplementowanej wersji algorytmu może powodować znaczące zmiany w postaci mutowanego rozwiązania. Eksperymenty badające wpływ prawdopodobieństwa mutacji na otrzymywane rezultaty polegały na testowaniu trzech sytuacji:

1. prawdopodobieństwo mutacji równe 0,
2. wartość prawdopodobieństwa mała (w stosunku do prawdopodobieństwa krzyżowania),
3. wartość prawdopodobieństwa duża (rząd wielkości prawdopodobieństwa krzyżowania).

7.2.6. Najlepsze parametry

Ostatni ze scenariuszy testowych jest niejako podsumowaniem scenariuszy wcześniejszych. Polegał on na poszukiwaniu rozwiązania problemu dla parametrów ustawionych na wartości, które na podstawie poprzednich testów uznane zostały za najlepsze (spośród wartości testowanych) i porównaniu otrzymanych wyników z wynikami uzyskanymi na podstawie działania algorytmu z ustawieniami najgorszymi. Test ten pozwala na sprawdzenie, czy najlepsze znalezione w testach poprzednich wartości parametrów algorytmu ustawione razem również pozwalają na uzyskiwanie dobrych rozwiązań.

Rozdział ósmy poświęcony jest rezultatom uzyskanym w opisanych wyżej testach, a rozdział dziewiąty analizom uzyskanych w przeprowadzonych eksperymentach rezultatom.

8. Eksperymenty obliczeniowe

9. Rezultaty działania algorytmu dla oraz przeprowadzone testy

10. Analiza uzyskanych wyników

11. Podsumowanie i wnioski

Algorytmy przybliżone są potężnym narzędziem do rozwiązywania problemów, dla których algorytmy dokładne nie miałby możliwości na osiągnięcie sukcesu.

Operator mutacji w postaci wykorzystanej w algorytmie NPQGA może powodować duże zmiany w postaci rozwiązania dla którego zaszła mutacja

Kwantowa idea algorytmu może powodować, że mimo dużego prawdopodobieństwa na znalezienie się w danym stanie, kubit może znaleźć się w stanie przeciwnym. Może to powodować znaczne pogorszenie rozwiązania.

Bibliografia

[1] Michalewicz Zbigniew, *Algorytmy genetyczne + struktury danych = programy ewolucyjne*, przeł. Zbigniew Nahorski, Wyd 3, Wydawnictwa Naukowo- Techniczne, Warszawa 2003.

[2] Gwiazda Tomasz Dominik, *ewolucyjne w rozwiązywaniu nieliniowych problemów decyzyjnych*, Wydawnictwa Naukowe Wydziału Zarządzania Uniwersytetu Warszawskiego, Warszawa 2002.

[3] Goldberg David E., *Algorytmy genetyczne i ich zastosowania*, przeł. Kazimierz Grygiel, Wyd 3, Wydawnictwa Naukowo - Techniczne, Warszawa 2003.

[4] Hiller Frederick S., Lieberman Gerald J., *Introduction to Operations Research*, Wyd 5, McGraw - Hill Publishing Company, Nowy Jork 1990.

[5] Michalewicz Zbigniew, Fogel David B., *jak to rozwiązać czyli nowoczesna metaheurystyka*, przeł. Aleksy Schubert, Joanna Schubert, Wydawnictwa Naukowo - Techniczne, Warszawa 2006.

[6] Filipowicz Bogusław, Chmiel Wojciech, Kadłuczka Piotr, *Ukierunkowane poszukiwanie przestrzeni rozwiązań w algorytmach rojowych*, Półrocznik Automatyka, z. 2m t. 13, Wydawnictwo AGH, Kraków 2009.

[7] Laboudi Zakaria, Chikhi Salim, *Comparison of Genetic Algorithm and Quantum Genetic Algorithm*, The International Arab Journal of Information Technology, nr 3, t. 9, 2012.

[8] Stützle Thomas, Dorigo Marco, *ACO Algorithms for the Quadratic Assignment Problem*.

[9] Dorigo, Marco, Di Caro, Gambardella, Gianni, Luca M. *Ant Algorithms for Discrete Optimization*.

[10] Kadłuczka Piotr, Chmiel Wojciech, *Efektywność algorytmu ewolucyjnego wykorzystującego warunkową wartość oczekiwaną funkcji celu*, Półrocznik Automatyka, z. 1-2, t. 9, Wydawnictwo AGH, Kraków 2005.

[11] Filipowicz Bogusław, Kwiecień Joanna, *Algorytmy stadne w optymalizacji problemów przydziału przy kwadratowym wskaźniku jakości (QAP)*, Półrocznik Automatyka, z. 2, t. 15, Wydawnictwo AGH, Kraków 2011.

[12] Stawowy Adam, *Heurystyki i metaheurystyki*, wykład z przedmiotu Inteligencja Obliczeniowa.

[13] Kennedy James, Eberhart Russel, *Particle Swarm Optimization*, 1995.

[14] Liu Hongbo, Abraham Ajith, Zhang Jianying *A Particle Swarm Approach to Quadratic Assignment Problems*.

[15] Glover Fred, *Tabu Search: A Tutorial*.

[16] Glover Fred, *Tabu Searcg - Part I*, ORSA Journal on Computing, nr 3, t. 1, 1989.

[17] Kennedy James, Eberhart Russel C., *Swarm Intelligence*, Morgan Kaufmann Publishers, San Francisco, 2001.

[18] Vizirani Vijay V., *Approximation Algorithms*, Springer, Berlin 2003.

[19] Jones Gareth, *Genetic and Evolutionary Algorithms*.

[20] Gu Jinwei, Gu Xingsheng, Gu Manzhan, *A novel parallel quantum genetic algorithm for stochastic job shop scheduling*, Journal of Mathematical Analysis and Applications, 63-81, 2009.

[21] Zbiór instancji testowych problemu QAP: http://anjos.mgi.polymtl.ca/qaplib/inst.html.

DODATEK A

DODATEK B