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 quadratic 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.

Serdecznie dziękuję
Promotorowi dr. inż. Wojciechowi Chmielowi za pomoc merytoryczną przy pisaniu niniejszej pracy oraz mojej mamie Annie Kultys za sprawdzenie tekstu pod względem stylistycznym

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, liczba możliwych rozwiązań rośnie bardzo szybko. Już dla stosunkowo małej liczby 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 oraz przedstawione są w nim rezultaty eksperymentów w postaci różnych tabel i wykresów.

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

Ostatni, dziesiąty 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 z 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 liczbę placówek do takiej samej liczby 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 [11],

– właściwa lokalizacja fabryk [11],

– organizacja biur, oddziałów szpitalnych [11],

– wyważanie turbin w silnikach odrzutowych [11].

2.3. Model matematyczny

Model matematyczny zagadnienia przydziału kwadratowego może być przedstawiony w następujący sposób [11]:

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, liczba połączeń, liczba 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.

Mimo, że algorytmy aproksymacyjne pozwalają przeważnie na znalezienie rozwiązania jedynie przybliżonego w pewnym sensie przejawiają podobieństwo do algorytmów dokładnych: również polegają na uchwyceniu istoty problemu i następnie na znalezieniu sposobu, który pozwoli na jej wykorzystanie [20].

Ogromna liczba 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 często 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 liczby 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 [15]:
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, liczbę 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. Zwracane przez algorytm pozycje cząstek mogą być opisane współrzędnymi o powtarzających się wartościach. Dlatego nie jest możliwe wykorzystanie tego algorytmu w najprostszy sposób, czyli taki, w którym wartość *i-tej* współrzędnej cząstki określa przydział do *i-tej* lokalizacji obiektu o wartości tej wartości. Taki sposób mógłby spowodować wielokrotny przydział tego samego obiektu do wielu miejsc i pominięcie w ogóle innych obiektów.

Jedno z możliwych zastosowań algorytmu PSO dla problemu QAP zostało zaproponowane w publikacji [15]. 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 [4].

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 określonych sytuacjach możliwe jest wykonanie ruchu zabronionego. Zdefiniowana jest specjalna funkcja, zwana funkcją aspiracji, która pozwala obliczyć, czy zabroniony ruch będzie jednak opłacalny [18]. Najczęstszym przypadkiem dopuszczenia do użycia zabronionego ruchu jest sytuacja, gdy jego wykonanie pozwoli na uzyskanie lepszego rozwiązania w stosunku do dotychczas wypracowanych.

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.

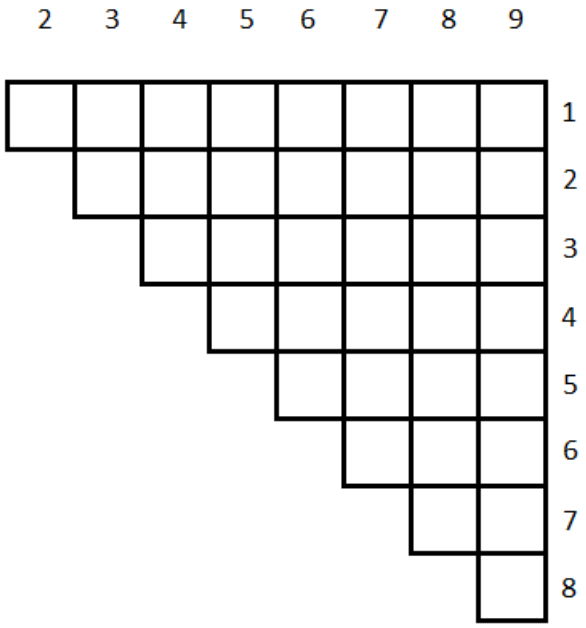
3.2.2. Pseudokod algorytmu Tabu Search

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 ruchów dozwolonych i zabronionych;
    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 [17]:



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 [1].

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 liczbę 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 [3]:

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

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 [3]:

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

gdzie ρ jest współczynnikiem wyparowywania feromonów zostawianych przez mrówki, a Δ_{ij}^k określone jest wzorem [3]:

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

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 [3]:

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

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 [3].

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 [3]:

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

a uaktualnienie śladu feromonów, który jest przez tę mrówkę zostawiany, uzyskiwane jest z równania [3]:

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

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 [10]:

– 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 [8].

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 [6]. 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 [6]. Inną ważną zaletą algorytmów genetycznych jest to, że stosowanie ich nie wymaga spełniania założeń dotyczących przestrzeni poszukiwań, jakimi są

przykładowo ciągłość czy różniczkowalność [6]. 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 we-

dług 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, sprawdzane jest w oparciu o prawdopodobieństwo krzyżowania, czy należy je uwzględnić jako rodzica w procesie krzyżowania rozwiązań. W zależności od przyjętych założeń, może zaistnieć potrzeba doboru dodatkowych rozwiązań w przypadku, gdy jest ich niewystarczająca liczba, 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 [6].Sposób działania tego operatora może pozwolić 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 [7], 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 z ilu elementów składa się 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. Przykładowo, zdekodowany do postaci dziesiętnej ciąg liczb *5, 3, 7, 1, 1, 2, 1* po uszeregowaniu rosnąco wygląda w następujący sposób: *1, 1, 1, 2, 3, 5, 7*,

co reprezentuje następujący przydział obiektów: *1, 2, 3, 4, 5, 6, 7*. Po uwzględnieniu, która liczba z ciągu reprezentuje który obiekt, otrzymuje się ostatecznie następującą permutację, reprezentującą rozwiązanie problemu QAP: *6, 5, 7, 1, 2, 4, 3*.

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 cyfry 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 ← 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 liczbą 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:

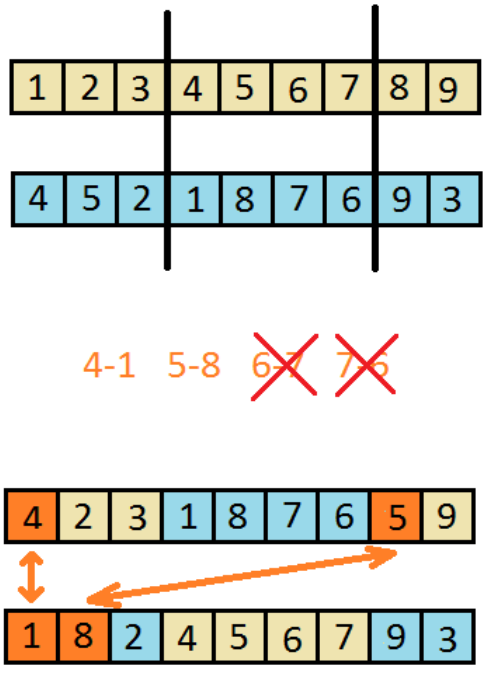
$$\eta_{min} = 2 - \eta_{max},\ 1 \leq \eta_{max} \leq 2. \tag{5.3}$$

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

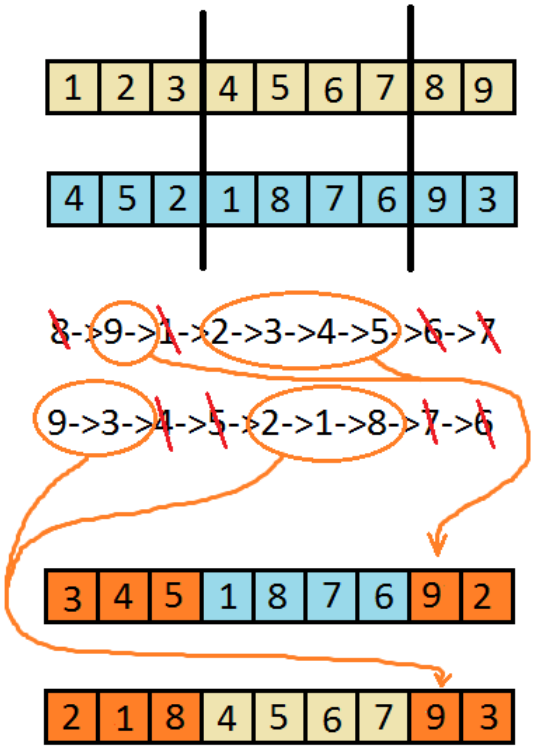
Oprócz proponowanego przez autorów operatora krzyżowania cyklicznego, zostały również wykorzystane operatory PMX oraz OX. Operator PMX, czyli operator krzyżowania z częściowym odwzorowaniem, polega na zamianie w wybranym fragmencie genów pomiędzy rodzicami i utworzeniu na tej podstawie listy odwzorowań. Fragment ten nazywany jest sekcją kojarzenia. 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 [7], 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#.

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.

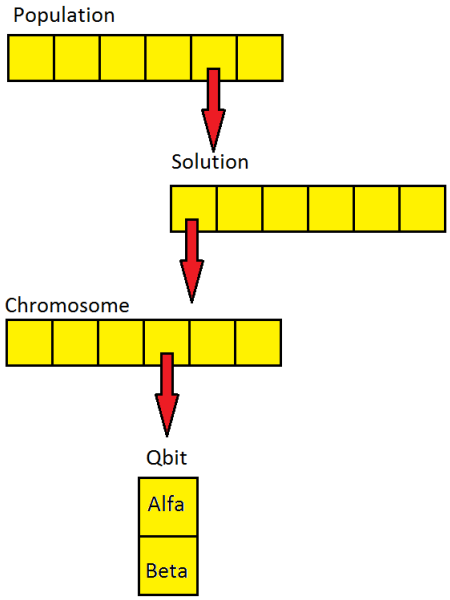
Interfejs *IEvolutionAlgorithm* dziedziczy po interfejsie *IOptimisationAlgorithm*. Na bazie tych interfejsów powstała klasa *QgAlgorithm* zawierająca metody pozwalające na ustawienie i uruchomienie algorytmu oraz pozwalająca na zwrócenie rezultatów działania algorytmu.

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. Funkcja testująca działanie algorytmu

Oprócz klas związanych z samym algorytmem NPQGA została napisana klasa pozwalająca na przetestowanie algorytmu. Składa się ona z dwóch funkcji: testującej oraz wczytującej instancje problemu QAP i zapisującej wartości macierzy przepływu i odległości do opisanego wyżej singletona.

Po wywołaniu funkcji testującej algorytm użytkownik proszony jest o podanie, dla ilu wartości testowanego parametru chce przeprowadzić testy oraz ile razy powtórzyć eksperyment dla danej wartości. Następnie należy podać, który parametr algorytmu (operator selekcji, operator krzyżowania, prawdopodobieństwo krzyżowania, prawdopodobieństwo mutacji, operator bramki kwantowej) ma być testowany. Po tym wyborze użytkownik proszony jest o podanie kolejnych wartości wybranego parametru algorytmu. Kolejnym krokiem jest wybór instancji testowych, dla których mają zostać przeprowadzone eksperymenty, a następnie należy ustawić pozostałe parametry algorytmu, których wartość będzie stała podczas wszystkich testów. Ostatnim krokiem jest podanie szablonu nazwy pliku, do którego zostaną zapisane rezultaty testów.

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. Istnieje także możliwość uruchomienia algorytmu bez bramki kwantowej.

W obrębie testowania różnych wartości jednego parametru dla jednej instancji testowej populacja startowa generowana jest raz i służy jako punkt wyjścia dla każdego z testów.

Aplikacja została napisana w sposób niepozwalający na przyjęcie nieprawidłowych danych.

Po zakończeniu testów użytkownik ma możliwość ponownego rozpoczęcia eksperymentów lub zamknięcia aplikacji.

6.4. Rezultaty działania aplikacji

Wyniki działania eksperymentów zapisywane są w pliku o nazwie w formacie [szablon nazwy]_[parametr testowany] i rozszerzeniu *xls* bądź *xlsx*, w zależności od wersji oprogramowania *Microsoft Excel* zainstalowanej na komputerze użytkownika aplikacji. W osobnych arkuszach danego pliku znajdują się rezultaty testów dla poszczególnych wartości instancji testowych.

Na rezultaty eksperymentów składają się następujące wartości:

- wartość funkcji celu najlepszego rozwiązania z populacji startowej,
- wartość funkcji celu najlepszego rozwiązania znalezionego podczas wszystkich przeprowadzonych testów dla danej wartości testowanego parametru,
- średnia wartość najlepszych rozwiązań zwróconych przez algorytm w powyższych testach,
- numer iteracji, w której najlepsze rozwiązanie zostało znalezione,
- średnia wartość numerów iteracji najlepszych rozwiązań w poszczególnych testach,
- wartość błędu względnego najlepszego rozwiązania.

6.5. Szczegóły związane z implementacją poszczególnych elementów algorytmu

Kilka szczegółów związanych z działaniem algorytmu nie zostało poruszonych w publikacji [7]. 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 zajścia 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 znajdować 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 działania bramki kwantowej i nazywane jest najlepszym, jest faktycznie najlepszym rozwiązaniem, ale uzyskanym w poprzednim pokoleniu. Operator bramki kwantowej używany jest dopiero na populacji poddanej już działaniu operatorów krzyżowania i mutacji w danej iteracji algorytmu. Z tego powodu istnieje możliwość, że dopasowanie zapamiętanego rozwiązania będzie gorsze niż dopasowanie niektórych rozwiązań poddawanych działaniu bramki kwantowej. Wynika to z faktu, iż w efekcie działania operatorów mutacji i krzyżowania mogą powstać rozwiązania lepsze niż te, które dotychczas istniały w populacji. 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 uruchamiania funkcji testującej sprawdzane jest na początku, czy wymienione foldery istnieją. Jeśli nie, są tworzone i użytkownik jest o tym powiadamiany. Następnie funkcja kończy swoje działanie.

Próba uruchomienia testów, gdy folder z instancjami jest pusty, gdy nie udało się odczytać danych z pliku z danymi dla problemu QAP, bądź nastąpił błąd podczas zapisu danych do pliku z rezultatami działania algorytmu, kończy się zgłoszeniem błędu, po czym funkcja kończy swoje działanie.

Dla operatorów OX i PMX przyjęto, że sekcja kojarzenia jest nie mniejsza niż 40% całego rozwiązania.

6.6. Pseudokod algorytmu zaimplementowanego w aplikacji

Poniżej przedstawiony jest pseudokod algorytmu, który został zaimplementowany w utworzonej aplikacji, z uwzględnieniem opisanych w poprzednim rozdziale modyfikacji:

Wczytaj parametry algorytmu;
Zainicjuj populację;
wyznacz permutacyjna postać rozwiązań w populacji;
policz dopasowanie rozwiązań;
zapamiętaj najlepsze rozwiązanie;
while $t < t_{max}$ **do**
    $t \leftarrow t + 1$;
    if *wybrana selekcja rankingowa* **then**
        dokonaj selekcji rankingowej;
    end
    else
        dokonaj selekcji ruletkowej;
    end
    if *wybrano operator krzyżowania CX* **then**
        dokonaj operacji krzyżowania przy pomocy operatora CX;
    end
    else if *wybrano operator krzyżowania OX* **then**
        dokonaj operacji krzyżowania przy pomocy operatora CX;
    end
    else
        dokonaj operacji krzyżowania przy pomocy operatora PMX;
    end
    if *wybrano oryginalną wersję operatora bramki kwantowej* **then**
        aktualizuj rozwiązania przy pomocy oryginalnej wersji bramki kwantowej;
    end
    else if *wybrano zmodyfikowaną wersję operatora bramki kwantowej* **then**
        aktualizuj rozwiązania przy pomocy zmodyfikowanej wersji bramki kwantowej;
    end
    zapamiętaj najlepsze rozwiązanie;
end

Algorithm 5: Algorytm NPQGA

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 liczba 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 dotychczas znalezionym rozwiązaniem. Wykorzystane w eksperymentach instancje testowe pochodzą ze strony [21] 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 wybrano jedenaście instancji 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 Manhattanie, natomiast macierz przepływu jest wygenerowana na podstawie równomiernego rozkładu na przedziale *[1,12]*. Znane jest optymalne rozwiązanie i jego wartość wynosi 1652.

Następnie zostały wybrane cztery instancje: *Lipa20a*, *Lipa50b*, *Lipa80a* i *Lipa90a*. Ich autorami są Y. Li i P.M. Pardalos. Podobnie jak powyżej, nazwy instancji pochodzą od ich rozmiarów i nazwisk autorów. Instancje te nie mają konkretnego przełożenia na rzeczywistość, zostały wygenerowana w generatorze

problemów. Optymalne wartość dla tych problemów wynoszą odpowiednio 3683, 1210244, 253195 i 360630.

Szóstą 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.

Siódma z instancji nazywa się *esc64a* i jej autorami są B. Eschermann i H.J. Wunderlich. Dane znajdujące się w niej pochodzą z aplikacji testującej pewne sekwencyjne układy samotestujące. Optymalna wartość dla tego problemu wynosi 116.

Następnie została wybrana grupa trzech instancji *chr15b*, *chr18b* i *chr25a* autorstwa N. Christofidesa i E. Benaventa. Optymalne wartości dla tych problemów wynoszą odpowiednio 7990, 1534 i 3796.

Ostatnia, jedenasta instancja testowa nosi nazwę *bur26a* i jej autorami są R.E. Burkard i J. Offermann. Wartości w pierwszej macierzy związane są ze średnim czasem pisania na maszynie stenograficznej, a macierz druga zawiera częstotliwość występowania pewnych par liter w różnych językach. Wartość optymalnego rozwiązania dla tego problemu wynosi 5426670.

7.2. Scenariusze testowe

Z racji tego, iż stosunkowo duża liczba 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 uzasadnione jest uśrednianie wyników z wielu prób. 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 trzech 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 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 [7], 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 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 wielokrotne powtórzenie 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.

7.2.3. Prawdopodobieństwo krzyżowania

Algorytm zaimplementowany w aplikacji pozwala na ustawienie minimalnego i maksymalnego prawdopodobieństwa krzyżowania. Ustawienie obu parametrów na tę samą wartość powoduje, że praw-

dopodobieństwo 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.

7.2.4. 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).

7.2.5. Operator selekcji

Każda z dwóch zaimplementowanych metod selekcji ma swoje dobre i złe strony. Należało jednak sprawdzić, która z metod jest bardziej przydatna w przypadku postawionego problemu. Dodatkowo metoda rankingowa pozwala na ustawienie parametru η,należało więc 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.

7.2.6. 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 w testach opisanych wcześniej. 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 rezultatów uzyskanych w przeprowadzonych eksperymentach.

8. Eksperymenty obliczeniowe

W niniejszym rozdziale przedstawione zostały rezultaty przeprowadzonych eksperymentów. Na podstawie danych zawartych w zamieszczonych tabelach została przeprowadzona analiza, której wyniki zostały zawarte w kolejnym, dziewiątym rozdziale.

8.1. Bramka kwantowa

W testach, których rezultaty znajdują się poniżej, zmianie ulegała wersja bramki kwantowej, pozostałe parametry algorytmu były stałe i ich wartości znajdują się w tableli 8.1:

liczba iteracji	rozmiar populacji	min. prawd. krzyżowania	maks. prawd krzyżowania	operator krzyżowania	prawd. mutacji	selekcja	liczba testów
300	50	0,8	0,8	CX	0,05	ruletkowa	3

Tablica 8.1: Stałe parametry eksperymentu związanego z bramka kwantową

Poniżej znajdują się tabele z rezultatami otrzymany dla każdej z instancji testowej. Oznaczenia w pierwszym wierszu tabeli to odpowiednio: wartość funkcji celu najlepszego rozwiązania z populacji początkowej, wartość funkcji celu najlepszego z rozwiązań znalezionego w ogóle podczas wszystkich testów dla danej wartości testowanego parametru, średnia wartość funkcji celu dla najlepszych rozwiązań w każdym z testów, numer iteracji, w której znaleziono najlepsze rozwiązanie, średnia wartość numerów iteracji najlepszych rozwiązań w każdym z testów oraz błąd względny najlepszego z rozwiązań w stosunku do rozwiązania optymalnego.

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
brak	5684727	5571084	5576314,667	12	17,66666667	0,0266119
oryginalna	5684727	5531474	5552662,333	12	102,6666667	0,019312765
zmodyfikowana	5684727	5496454	5509412	116	171,3333333	0,012859452

Tablica 8.2: Rezultaty dla instancji bur26a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
brak	37898	24242	25478,66667	3	55	2,034042553
oryginalna	37898	14828	16308	111	146,3333333	0,855819775
zmodyfikowana	37898	15174	17303,33333	43	51	0,899123905

Tablica 8.3: Rezultaty dla instancji chr15b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
brak	2950	2522	2671,333333	3	5,333333333	0,644067797
oryginalna	2950	1882	1964	14	157	0,226857888
zmodyfikowana	2950	1672	1777,333333	43	61,33333333	0,089960887

Tablica 8.4: Rezultaty dla instancji chr18b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
brak	16356	11828	12355,33333	5	103,6666667	2,115911486
oryginalna	16356	11054	11379,33333	114	162,6666667	1,912012645
zmodyfikowana	16356	7716	8720,666667	75	95,66666667	1,032665964

Tablica 8.5: Rezultaty dla instancji chr25a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
brak	228	148	173,3333333	15	17,66666667	0,275862069
oryginalna	228	138	150,6666667	32	121,6666667	0,189655172
zmodyfikowana	228	116	118,6666667	78	125,6666667	0

Tablica 8.6: Rezultaty dla instancji esc64a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
brak	1768	1702	1724	1	6,333333333	0,030266344
oryginalna	1768	1684	1688,666667	18	45	0,01937046
zmodyfikowana	1768	1684	1700	28	35,33333333	0,01937046

Tablica 8.7: Rezultaty dla instancji had12

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
brak	127190	113420	116506,6667	8	13,66666667	0,278692221
oryginalna	127190	110950	112093,3333	96	211	0,250845547
zmodyfikowana	127190	99100	102046,6667	112	180	0,117249154

Tablica 8.8: Rezultaty dla instancji kra32

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
brak	3865	3853	3859,666667	0	3,666666667	0,046158023
oryginalna	3865	3831	3840,333333	23	30,33333333	0,040184632
zmodyfikowana	3865	3803	3819	43	62,66666667	0,032582134

Tablica 8.9: Rezultaty dla instancji lipa20a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
brak	1541214	1512985	1519929	19	63	0,25014873
oryginalna	1541214	1510388	1512682,667	96	176,3333333	0,248002882
zmodyfikowana	1541214	1486615	1490356,667	151	199	0,228359736

Tablica 8.10: Rezultaty dla instancji lipa50b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
brak	258029	257445	257612,6667	9	18,33333333	0,016785482
oryginalna	258029	257363	257495,6667	34	180	0,01646162
zmodyfikowana	258029	256700	256796	259	273,6666667	0,013843085

Tablica 8.11: Rezultaty dla instancji lipa80a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
brak	366925	366249	366354	21	23,33333333	0,015581066
oryginalna	366925	366271	366355,6667	50	140,6666667	0,015642071
zmodyfikowana	366925	365195	365309,6667	266	280,6666667	0,012658403

Tablica 8.12: Rezultaty dla instancji lipa90a

8.2. Operator krzyżowania

W eksperymencie dotyczącym operatora krzyżowania porównywane były rezultaty otrzymane przy użyciu różnych operatorów krzyżowania. Niezmienne parametry algorytmu ustawione były na te same wartości przedstawione w poniższej tabeli:

liczba iteracji	rozmiar populacji	min. prawd. krzyżowania	maks. prawd krzyżowania	bramka kwantowa	prawd. mutacji	selekcja	liczba testów
300	50	0,8	0,8	zmodyfikowana	0,05	ruletkowa	3

Tablica 8.13: Stałe parametry eksperymentu związanego z operatorem krzyżowania

W poniższych tabelach znajdują się rezultaty testów dla poszczególnych instancji testowych:

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
CX	5717454	5509912	5512025,667	63	184,6666667	0,015339425
OX	5717454	5545330	5575050	9	158	0,02186608
PMX	5717454	5499264	5536580,333	95	183,6666667	0,013377265

Tablica 8.14: Rezultaty dla instancji bur26a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
CX	31212	17554	18410,66667	55	100	1,196996245
OX	31212	17304	20116,66667	66	148,3333333	1,165707134
PMX	31212	19892	21148	118	175,6666667	1,489612015

Tablica 8.15: Rezultaty dla instancji chr15b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
CX	3550	1808	1865,333333	59	108,3333333	0,178617992
OX	3550	2216	2262	53	117,6666667	0,444589309
PMX	3550	1668	1840,666667	231	263,6666667	0,087353325

Tablica 8.16: Rezultaty dla instancji chr18b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
CX	14380	9124	9658,666667	84	102	1,403582719
OX	14380	11790	12020	151	208	2,105900948
PMX	14380	11288	11499,33333	43	68,66666667	1,973656481

Tablica 8.17: Rezultaty dla instancji chr25a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
CX	244	124	126	119	155	0,068965517
OX	244	186	192	30	128	0,603448276
PMX	244	162	164,6666667	58	186,3333333	0,396551724

Tablica 8.18: Rezultaty dla instancji esc64a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
CX	1812	1678	1685,333333	32	43,33333333	0,015738499
OX	1812	1686	1690,666667	30	118,6666667	0,020581114
PMX	1812	1690	1693,333333	69	150,6666667	0,023002421

Tablica 8.19: Rezultaty dla instancji had12

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
CX	126660	96620	97800	109	209	0,089289741
OX	126660	114210	116090	99	167	0,287598647
PMX	126660	111780	112970	258	274,3333333	0,260202931

Tablica 8.20: Rezultaty dla instancji kra32

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
CX	3877	3831	3843	48	72	0,040184632
OX	3877	3841	3847	75	143,6666667	0,04289981
PMX	3877	3844	3845	38	51,33333333	0,043714363

Tablica 8.21: Rezultaty dla instancji lipa20a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
CX	1543705	1487038	1493044,333	212	257	0,228709252
OX	1543705	1520666	1522054,333	20	52	0,256495384
PMX	1543705	1519178	1520194,667	16	122	0,25526588

Tablica 8.22: Rezultaty dla instancji lipa50b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
CX	258140	256810	256962,3333	256	277	0,014277533
OX	258140	257752	257764,3333	180	211	0,017997986
PMX	258140	257697	257705,6667	67	167	0,017780762

Tablica 8.23: Rezultaty dla instancji lipa80a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
CX	366995	365281	365551,6667	278	288	0,012896875
OX	366995	366456	366485,3333	7	191,6666667	0,016155062
PMX	366995	366474	366554,3333	31	167,6666667	0,016204975

Tablica 8.24: Rezultaty dla instancji lipa90a

8.3. Prawdopodobieństwo krzyżowania

W testach związanych z wpływem prawdopodobieństwa krzyżowania badano, jak różne wartości tego prawdopodobieństwa przekładają się na otrzymywane rezultaty. Pozostałe parametry były stałe i przyjęły poniższe wartości:

liczba iteracji	rozmiar populacji	bramka kwantowa	operator krzyżowania	prawd. mutacji	selekcja	liczba testów
300	50	zmodyfikowana	CX	0,05	ruletkowa	3

Tablica 8.25: Stałe parametry eksperymentu związanego z bramka kwantową

W tabelach znajdujących się poniżej zamieszczone są rezultaty testów związanych z prawdopodobieństwem krzyżowania. Wartości prawdopodobieństwa krzyżowania opisane są w formacie wartość min/max:

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,8/0,8	5711143	5496746	5511766,333	70	77	0,01291326
0,6/1	5711143	5490231	5504724	88	201	0,011712708
0,6/0,6	5711143	5486502	5501058	108	182,6666667	0,011025546
0,4/0,8	5711143	5482314	5510755,667	77	106,3333333	0,010253802
0,4/0,4	5711143	5481896	5513744,667	99	143	0,010176775
0,2/0,6	5711143	5505873	5519636,333	159	215	0,014595138

Tablica 8.26: Rezultaty dla instancji bur26a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,8/0,8	37702	17042	17710	27	52	1,132916145
0,6/1	37702	18938	20533,33333	34	44,66666667	1,370212766
0,6/0,6	37702	17892	18357,33333	39	83	1,239299124
0,4/0,8	37702	15906	19572	87	155,3333333	0,990738423
0,4/0,4	37702	13810	17276,66667	38	70,33333333	0,728410513
0,2/0,6	37702	17066	18640,66667	54	63	1,1359199

Tablica 8.27: Rezultaty dla instancji chr15b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,8/0,8	3284	1786	1816,666667	41	46	0,164276402
0,6/1	3284	1724	1748	50	74	0,123859192
0,6/0,6	3284	1730	1782,666667	41	125	0,127770535
0,4/0,8	3284	1690	1722,666667	58	119,3333333	0,101694915
0,4/0,4	3284	1642	1758	43	45,66666667	0,070404172
0,2/0,6	3284	1824	1882,666667	39	40	0,18904824

Tablica 8.28: Rezultaty dla instancji chr18b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,8/0,8	14972	8852	9461,333333	84	100,6666667	1,331928346
0,6/1	14972	8918	9278	85	216,6666667	1,349315068
0,6/0,6	14972	8634	9420	76	170,3333333	1,274499473
0,4/0,8	14972	9108	9724	82	168	1,399367756
0,4/0,4	14972	9276	9670,666667	59	96,66666667	1,443624868
0,2/0,6	14972	8506	9172,666667	83	167,3333333	1,240779768

Tablica 8.29: Rezultaty dla instancji chr25a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,8/0,8	250	120	126,6666667	90	119	0,034482759
0,6/1	250	118	118	99	190	0,017241379
0,6/0,6	250	120	122,6666667	110	135	0,034482759
0,4/0,8	250	120	122,6666667	99	142,6666667	0,034482759
0,4/0,4	250	120	124,6666667	185	224,6666667	0,034482759
0,2/0,6	250	122	127,3333333	95	109,6666667	0,051724138

Tablica 8.30: Rezultaty dla instancji esc64a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,8/0,8	1772	1662	1689,333333	9	21,33333333	0,006053269
0,6/1	1772	1670	1675,333333	20	33,66666667	0,010895884
0,6/0,6	1772	1682	1690,666667	27	46,33333333	0,018159806
0,4/0,8	1772	1682	1688,666667	9	24,66666667	0,018159806
0,4/0,4	1772	1680	1689,333333	16	68,66666667	0,016949153
0,2/0,6	1772	1672	1687,333333	10	33	0,012106538

Tablica 8.31: Rezultaty dla instancji had12

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,8/0,8	124520	98000	100270	94	148,3333333	0,104847802
0,6/1	124520	96800	98530	60	162,3333333	0,091319053
0,6/0,6	124520	98110	100293,3333	75	163	0,106087937
0,4/0,8	124520	97520	98430	93	176	0,099436302
0,4/0,4	124520	98220	98933,33333	102	117,3333333	0,107328072
0,2/0,6	124520	98580	101100	88	143,6666667	0,111386697

Tablica 8.32: Rezultaty dla instancji kra32

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,8/0,8	3896	3806	3815	170	234,6666667	0,033396687
0,6/1	3896	3787	3813	46	60,66666667	0,02823785
0,6/0,6	3896	3807	3809,666667	52	111,6666667	0,033668205
0,4/0,8	3896	3817	3837,333333	51	69	0,036383383
0,4/0,4	3896	3808	3810,333333	89	142,3333333	0,033939723
0,2/0,6	3896	3822	3831,333333	9	64,66666667	0,037740972

Tablica 8.33: Rezultaty dla instancji lipa20a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,8/0,8	1541782	1486907	1489416,333	182	216,3333333	0,228601009
0,6/1	1541782	1485129	1492526,667	139	177,6666667	0,227131884
0,6/0,6	1541782	1479032	1490646,333	214	238,6666667	0,222094057
0,4/0,8	1541782	1487758	1490308,667	202	242,6666667	0,229304173
0,4/0,4	1541782	1481221	1486103,667	219	249	0,223902783
0,2/0,6	1541782	1486806	1487716	154	210	0,228517555

Tablica 8.34: Rezultaty dla instancji lipa50b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,8/0,8	258062	256781	256835,6667	216	258	0,014162997
0,6/1	258062	256555	256811	250	269,3333333	0,013270404
0,6/0,6	258062	256618	256695,6667	227	250	0,013519224
0,4/0,8	258062	256707	256785,6667	226	273	0,013870732
0,4/0,4	258062	256821	256924,6667	241	244	0,014320978
0,2/0,6	258062	256830	256919,6667	236	267,6666667	0,014356524

Tablica 8.35: Rezultaty dla instancji lipa80a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,8/0,8	367043	365258	365389,3333	277	283,6666667	0,012833098
0,6/1	367043	365377	365549,6667	282	285	0,013163076
0,6/0,6	367043	365211	365393,3333	249	281	0,01270277
0,4/0,8	367043	365422	365548	286	288	0,013287857
0,4/0,4	367043	365265	365387,6667	271	285	0,012852508
0,2/0,6	367043	365491	365527,3333	278	290,6666667	0,013479189

Tablica 8.36: Rezultaty dla instancji lipa90a

8.4. Prawdopodobieństwo mutacji

W poniższej tabeli znajdują się wartości parametrów, które nie zmieniały się podczas testów dotyczących prawdopodobieństwa mutacji:

liczba iteracji	rozmiar populacji	min. prawd. krzyżowania	maks. prawd krzyżowania	bramka kwantowa	operator. krzyżowania	selekcja	liczba testów
300	50	0,8	0,8	zmodyfikowana	CX	ruletkowa	3

Tablica 8.37: Stałe parametry eksperymentu związanego z prawdopodobieństwem mutacji

Poniżej znajdują się tabele z rezultatami testów związanych z wpływem prawdopodobieństwa mutacji na otrzymywane wyniki:

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,4	5711735	5483153	5497864	148	195	0,010408409
0,05	5711735	5491005	5502819,333	166	197	0,011855337
0	5711735	5487411	5499076	96	108,3333333	0,011193052

Tablica 8.38: Rezultaty dla instancji bur26a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,4	33308	16040	18794,66667	26	43	1,007509387
0,05	33308	14050	16928,66667	39	64	0,75844806
0	33308	15152	18228	54	76,33333333	0,896370463

Tablica 8.39: Rezultaty dla instancji chr15b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,4	3418	1660	1750,666667	34	65,33333333	0,082138201
0,05	3418	1828	1915,333333	51	63,66666667	0,191655802
0	3418	1768	1808	42	51,33333333	0,152542373

Tablica 8.40: Rezultaty dla instancji chr18b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,4	14228	7250	8993,333333	113	166,6666667	0,909905163
0,05	14228	8994	9823,333333	78	89,66666667	1,369336143
0	14228	8654	9313,333333	76	77,33333333	1,279768177

Tablica 8.41: Rezultaty dla instancji chr25a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,4	246	120	123,3333333	99	152,3333333	0,034482759
0,05	246	116	118,6666667	93	137	0
0	246	118	120,6666667	92	128,3333333	0,017241379

Tablica 8.42: Rezultaty dla instancji esc64a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,4	1764	1684	1698,666667	5	13	0,01937046
0,05	1764	1666	1681,333333	22	39,33333333	0,008474576
0	1764	1664	1680	27	32	0,007263923

Tablica 8.43: Rezultaty dla instancji had12

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,4	126810	98170	100883,3333	69	111,6666667	0,106764374
0,05	126810	97950	99770	97	120,3333333	0,104284104
0	126810	96430	98010	80	105	0,087147689

Tablica 8.44: Rezultaty dla instancji kra32

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,4	3890	3830	3831,666667	68	71	0,039913114
0,05	3890	3814	3830,666667	63	90	0,03556883
0	3890	3827	3835,333333	52	59	0,039098561

Tablica 8.45: Rezultaty dla instancji lipa20a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,4	1523711	1484764	1488391,333	129	183,3333333	0,226830292
0,05	1523711	1485442	1488679,333	172	215,6666667	0,22739051
0	1523711	1478320	1486365,333	178	253,6666667	0,221505746

Tablica 8.46: Rezultaty dla instancji lipa50b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,4	258081	256604	256788	233	257,6666667	0,013463931
0,05	258081	256682	256794,6667	246	272,3333333	0,013771994
0	258081	256708	256813	226	251,6666667	0,013874682

Tablica 8.47: Rezultaty dla instancji lipa80a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
0,4	367035	365162	365355,6667	283	289	0,012566897
0,05	367035	365272	365318,6667	220	270,6666667	0,012871919
0	367035	365201	365373,6667	290	294	0,012675041

Tablica 8.48: Rezultaty dla instancji lipa90a

8.5. Operator selekcji

Poniższa tabela zawiera wartości parametrów algorytmu, które były stałe podczas eksperymentów badających wpływ operatora selekcji na otrzymywane rezultaty:

liczba iteracji	rozmiar populacji	min. prawd. krzyżowania	maks. prawd krzyżowania	bramka kwantowa	operator. krzyżowania	prawd. mutacji	liczba testów
300	50	0,8	0,8	zmodyfikowana	CX	0,05	3

Tablica 8.49: Stałe parametry eksperymentu związanego z operatorem selekcji

Znajdujące się poniżej tabele przedstawiają rezultaty testów przeprowadzonych dla wybranych instancji testowych oraz dla różnych ustawień algorytmu w kwestii operatora selekcji:

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
ruletkowa	5672969	5508559	5517374,667	106	183	0,015090101
rankingowa, $\eta = 1$	5672969	5506490	5515610	73	93,33333333	0,014708836
rankingowa, $\eta = 1,5$	5672969	5480158	5483712,333	60	143,3333333	0,009856505
rankingowa, $\eta = 2$	5672969	5456714	5506032,667	79	184,6666667	0,00553636

Tablica 8.50: Rezultaty dla instancji bur26a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
ruletkowa	30036	15904	18153,33333	61	84,66666667	0,99048811
rankingowa, $\eta = 1$	30036	13516	19001,33333	26	68,66666667	0,691614518
rankingowa, $\eta = 1,5$	30036	14974	17978,66667	6	39	0,874092616
rankingowa, $\eta = 2$	30036	12864	16804,66667	40	59,66666667	0,610012516

Tablica 8.51: Rezultaty dla instancji chr15b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
ruletkowa	3516	1772	1868,666667	42	48,66666667	0,155149935
rankingowa, $\eta = 1$	3516	1680	1700	47	53,66666667	0,09517601
rankingowa, $\eta = 1,5$	3516	1614	1728,666667	47	53	0,052151239
rankingowa, $\eta = 2$	3516	1688	1748,666667	50	51,66666667	0,100391134

Tablica 8.52: Rezultaty dla instancji chr18b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
ruletkowa	14510	8124	8770,666667	45	128,3333333	1,140147524
rankingowa, $\eta = 1$	14510	9140	9426	71	149	1,407797682
rankingowa, $\eta = 1,5$	14510	8786	9903,333333	73	168	1,314541623
rankingowa, $\eta = 2$	14510	7940	8378,666667	79	112,3333333	1,091675448

Tablica 8.53: Rezultaty dla instancji chr25a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
ruletkowa	248	122	125,3333333	87	132	0,051724138
rankingowa, $\eta = 1$	248	120	122	105	159,6666667	0,034482759
rankingowa, $\eta = 1,5$	248	122	125,3333333	103	109,6666667	0,051724138
rankingowa, $\eta = 2$	248	118	120	92	102,3333333	0,017241379

Tablica 8.54: Rezultaty dla instancji esc64a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
ruletkowa	1762	1690	1698	17	34,66666667	0,023002421
rankingowa, $\eta = 1$	1762	1672	1676,666667	25	65	0,012106538
rankingowa, $\eta = 1,5$	1762	1686	1689,33333322	22	30	0,020581114
rankingowa, $\eta = 2$	1762	1680	1686,666667	22	48,66666667	0,016949153

Tablica 8.55: Rezultaty dla instancji had12

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
ruletkowa	127600	93960	98133,33333	94	151,3333333	0,059301015
rankingowa, $\eta = 1$	127600	97170	99633,33333	73	110	0,095490417
rankingowa, $\eta = 1,5$	127600	98550	98890	159	201,3333333	0,111048478
rankingowa, $\eta = 2$	127600	94820	98433,33333	108	221	0,068996618

Tablica 8.56: Rezultaty dla instancji kra32

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
ruletkowa	3885	3839	3845,333333	56	135,3333333	0,042356774
rankingowa, $\eta = 1$	3885	3823	3832,333333	62	124,6666667	0,03801249
rankingowa, $\eta = 1,5$	3885	3818	3823	50	65,33333333	0,036654901
rankingowa, $\eta = 2$	3885	3827	3833,333333	50	55,66666667	0,039098561

Tablica 8.57: Rezultaty dla instancji lipa20a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
ruletkowa	1529495	1481090	1487070,667	222	268	0,223794541
rankingowa, $\eta = 1$	1529495	1477465	1484398	170	225,3333333	0,220799277
rankingowa, $\eta = 1,5$	1529495	1484140	1490860,667	228	250	0,226314694
rankingowa, $\eta = 2$	1529495	1485568	1487855,667	127	178,6666667	0,227494621

Tablica 8.58: Rezultaty dla instancji lipa50b

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
ruletkowa	258039	256823	256847,3333	236	256	0,014328877
rankingowa, $\eta = 1$	258039	256635	256769,3333	292	295	0,013586366
rankingowa, $\eta = 1,5$	258039	256670	256766	283	290	0,0137246
rankingowa, $\eta = 2$	258039	256683	256745,3333	283	291,3333333	0,013775943

Tablica 8.59: Rezultaty dla instancji lipa80a

	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
ruletkowa	367017	365153	365358,3333	270	282,6666667	0,01254194
rankingowa, $\eta = 1$	367017	365214	365433,6667	249	269,3333333	0,012711089
rankingowa, $\eta = 1,5$	367017	365409	365462,3333	265	282,6666667	0,013251809
rankingowa, $\eta = 2$	367017	365296	365355	234	270,3333333	0,012938469

Tablica 8.60: Rezultaty dla instancji lipa90a

8.6. Najlepsze parametry

W tabeli poniżej znajdują się wartości parametrów, które zostały uznane na podstawie wcześniejszych eksperymentów za najlepsze. Uzasadnienie ich wyboru znajduje się kolejnym rozdziale:

bramka kwantowa	operator krzyżowania	min. prawd. krzyżowania	maks. prawd krzyżowania	prawd. mutacji	operator selekcji
zmodyfikowana	CX	0,6	0,6	0,4	rankingowa, $\eta = 2$

Tablica 8.61: Najlepsze wartości testowanych parametrów

Tabela znajdująca się poniżej zawiera rezultaty eksperymentu dla każdej z instancji z parametrami algorytmu ustawionymi na najlepsze wartości:

instancja	f_{init}	f_{best}	$\overline{f_{best}}$	n	$\overline{n}$	δ
bur26a	5715964	5480950	5500345,333	107	140,6666667	0,010002451
chr15b	37200	15514	16730	38	39	0,941677096
chr18b	3464	1718	1776	36	44	0,119947849
chr25a	15420	8904	9217,333333	108	202,6666667	1,345626976
esc64a	206	116	121,3333333	98	149,3333333	0
had12	1790	1676	1695,333333	28	32,66666667	0,014527845
kra32	126930	101670	102790	100	118	0,146223224
lipa20a	3885	3816	3825,666667	53	57,66666667	0,036111865
lipa50b	1538103	1487146	1490913,667	169	207,3333333	0,22879849
lipa80a	257950	256698	256759,3333	264	278,3333333	0,013835186
lipa90a	366979	365277	365396	248	269,6666667	0,012885783

Tablica 8.62: Rezultaty testu dla wszystkich instancji testowych dla najlepszych wartości parametrów algorytmu

9. Analiza uzyskanych wyników

Stwierdzenie, która wartość wybranego parametru algorytmu jest lepsza, jest zadaniem trudnym. Przyjęto więc, że w pierwszej kolejności decyzję należy podjąć w oparciu o wartość błędu względnego, a w przypadku gdy różnice są niewielkie, wybór najlepszej wartości testowanego parametru podejmowany jest na podstawie średniej wartości numerów iteracji algorytmu, w których uzyskano najlepsze rozwiązania.

Należy oczywiście podkreślić, iż stwierdzenie, że wybrana wartość parametru jest najlepsza, dotyczy tylko wartości testowanych. Nie jest wiadome, czy inna wartość, która nie była sprawdzana eksperymentalnie nie pozwoliłaby na uzyskanie lepszych rezultatów. Natomiast wartości te starano się dobrać w taki sposób, by można było odpowiedzieć na pytanie, czy, przykładowo, testowany parametr powinien przyjmować wartości większe czy mniejsze.

Można zauważyć, że dla pewnych instancji testowych wartości błędów względnych są duże i nie ma znaczenia, który parametr algorytmu był testowany. Sytuacja ta może wynikać z wielu powodów. Przyczyną niewątpliwie może być zły dobór parametrów, które nie podlegały aktualnie testowaniu. Być może dla jednej konkretnej instancji, dużo lepiej spisywałby się inny operator krzyżowania, pod warunkiem, że prawdopodobieństwo mutacji było inne. Być może instancja problemu skonstruowana jest tak, by istniało wiele minimów lokalnych o podobnej postaci do rozwiązania optymalnego.

9.1. Bramka kwantowa

Testowane były trzy warianty: brak bramki kwantowej, wersja bramki zaproponowana w [7] oraz wersja nazywana zmodyfikowaną, zaproponowana przez autora niniejszej pracy. Poniżej znajduje się tabela zawierająca informację na temat, ile razy dany wariant pozwolił na uzyskanie lepszych rezultatów:

brak bramki	wersja oryginalna	wersja zmodyfikowana
0	0	11

Tablica 9.1: Porównanie wersji bramki kwantowej

Powyższa tabela jednoznacznie pokazuje, przy opisanych wyżej kryteriach oceny, która „wartość”

bramki kwantowej jest najlepsza. Jednakże, dla niektórych instancji testowych różnice bywały niewielkie.

Przy analizie jedynie średniej wartości numerów iteracji, zauważyć można, że wartość była najmniejsza dla sytuacji, gdy bramka kwantowa w ogóle była niewykorzystywana, a największa dla wersji zmodyfikowanej. Świadczy to o tym, że bramka ta pozwala unikać zbyt szybkiej zbieżności do minimum lokalnego. Natomiast różnice na korzyść wersji zmodyfikowanej można tłumaczyć tym, że sposób jej działania ma lepszy wpływ na szersze przeszukiwanie przestrzeni rozwiązań - nakierowuje ona przetwarzane rozwiązania na rozwiązanie lepsze od nich, lecz w przeciwieństwie do wersji oryginalnej zmiana kąta jest dużo mniejsza. Dzięki temu ewentualna zbieżność do minimum lokalnego jest mniej prawdopodobna.

Należy też wspomnieć, że publikacja [7] zawierała opis algorytmu i jego wykorzystanie w problemie szeregowania zadań. W eksperymentach użyty algorytm był po wielu modyfikacjach względem swojej oryginalnej wersji i posłużył do rozwiązywania innego problemu. Być może to również miało wpływ na efekty uzyskiwane przy wykorzystani oryginalnej wersji bramki kwantowej. Możliwe jest, że wartości w tablicy *Look Up* dobrane były specjalnie pod kątem problemu szeregowania zadań i dla innych problemów wartości te niekoniecznie muszą być odpowiednie.

Dla jednej instancji - *esc64a* korzystając z bramki w wersji zmodyfikowanej udało się uzyskać rozwiązanie równe optymalnemu, gdzie średnia wartość numerów iteracji wynosiła około 126, a dla wersji oryginalnej przy wartości błędu około 0,19 średnia wartości iteracji wynosiła mniej więcej 122.

9.2. Operator krzyżowania

Kolejnym, drugim testowanym parametrem był operator krzyżowania. Testowane były operatory CX, OX oraz PMX. W poniższej tabeli znajduje się podsumowanie, ile razy dany operator okazał się najlepszy:

CX	OX	PMX
8	1	2

Tablica 9.2: Porównanie operatorów krzyżowania

Z powyższej tabeli wynika, że najlepszym operatorem okazał się operator CX, jednakże nie było tak w każdym przypadku. Dla instancji *chr15b* lepszy okazał się operator OX, natomiast dla instancji *bur26a* i *chr18b* PMX.

W przypadku operatorów krzyżowania trudno ocenić, dlaczego jeden działa lepiej od innych. Sam proces krzyżowania w problemie, którego rozwiązaniem jest permutacja, powoduje duże zmiany w rozwiązaniach potomnych. Oczywiście rozwiązania te mają cechy swoich rodziców, lecz sposób, w jaki informacja jest wymieniana, wprowadza duże modyfikacje. W permutacji ważna jest kolejność jej ele-

mentów, natomiast krzyżowanie nierzadko kolejność tę zaburza. Z tego wynika być może, że operator CX w mniejszym stopniu powoduje te modyfikacje.

W żadnym z przypadków nie udało się uzyskać rozwiązania optymalnego, choć często błąd względny był niewielki. Natomiast w przypadku instancji *chr25a* dla każdego z operatorów wartość błędu wynosiła grubo ponad 1, a dla operatora OX nawet ponad 2.

9.3. Prawdopodobieństwo krzyżowania

Algorytm NPQGA w swych założeniach pozwala na zmianę wartości krzyżowania w czasie. Wartość ta wraz z rosnącą liczbą wykonanych iteracji algorytmu malała do określonej wartości minimalnej, by osiągnąwszy ją nie zmieniać się już do końca działania algorytmu. Testy dotyczące wpływu prawdopodobieństwa krzyżowania na otrzymywane rezultaty miały na celu pokazanie, czy zmieniająca się jego wartość ma realny wpływ na rezultaty i jaka jego wartość jest najlepsza. Poniższa tabela pokazuje, ile razy dla jakiej wartości prawdopodobieństwa uzyskano najlepsze rezultaty:

0,8/0,8	0,6/1	0,6/0,6	0,4/0,8	0,4/0,4	0,2/0,6
1	4	2	0	3	1

Tablica 9.3: Porównanie wartości prawdopodobieństwa krzyżowania

Z powyższej tabeli wynika wprawdzie jednoznacznie, która wartość prawdopodobieństwa jest lepsza od innych, nie wynika jednak z niej, która wartość jest wyraźnie lepsza od innych. Sytuacja ta wydaje się być ciekawa - krzyżowanie jest zasadniczą częścią algorytmów genetycznych, a więc dobór właściwej wartości jego prawdopodobieństwa powinien być rzeczą bardzo ważną. Być może obecność bramki kwantowej powoduje to, że krzyżowanie traci na znaczeniu - nie jest już jedynym operatorem polepszającym wartość populacji z pokolenia na pokolenie.

Analizując tabelę 9.3 można dojść do wniosku, że najlepsze rezultaty pozwoli osiągnąć prawdopodobieństwo krzyżowania o wartości równiej mniej więcej 0,6. Warto jednak przy tym podkreślić, że różnice pomiędzy błędami względnymi wyliczonymi dla każdej sytuacji nie są duże.

9.4. Prawdopodobieństwo mutacji

Niewątpliwie, sposób w jaki przeprowadzana jest mutacja w algorytmie NPQGA powoduje, że jej wpływ na rozwiązania, które zostały jej poddane jest dużo większy niż w innych algorytmach genetycznych.

Poniższa tabela prezentuje, ile razy testowana wartość prawdopodobieństwa mutacji pozwoliła na uzyskanie lepszych rezultatów niż pozostałe:

0,4	0,05	0
5	3	3

Tablica 9.4: Porównanie wartości prawdopodobieństwa mutacji

Podsumowanie wyników przedstawione w powyższej tabeli może wydawać się lekko zaskakujące. Lepsze rezultaty uzyskano przeważnie w sytuacji, gdy wartość prawdopodobieństwa mutacji wynosiła 0,4. Wynikać z tego może fakt, że częstsza mutacja zwiększa prawdopodobieństwo zdekodowania rozwiązania w taki sposób, że jest ono bardziej podobne do rozwiązania lepszego, uwzględnianego w działaniu bramki kwantowej. Równocześnie, małe różnice w uzyskiwanych rezultatach dla wszystkich testowanych wartości prawdopodobieństwa mutacji mogą potwierdzać tezę, że mutacja ma dużo mniejszy wpływ na osiągane przez algorytm wyniki niż operacja krzyżowania.

Dla instancji testowej o nazwie *chr15b* uzyskany błąd względny dla każdego przypadku był duży, a dla wartości prawdopodobieństwa mutacji równego 0,4 osiągnął wartość przekraczającą 1. Udało się także raz znaleźć rozwiązanie optymalne. Sytuacja ta miała miejsce dla instancji *esc64a* i wartości prawdopodobieństwa mutacji równego 0,05.

9.5. Operator selekcji

Ostatnim testowanym parametrem algorytmu był operator selekcji. Testom podlegała selekcja ruletkowa oraz rankingowa dla różnych wartości parametru η. Poniższa tabela przedstawia, ile razy dany z wariantów okazał się lepszy od pozostałych:

selekcja ruletkowa	selekcja rankingowa, $\eta = 1$	selekcja rankingowa, $\eta = 1,5$	selekcja rankingowa, $\eta = 2$
2	2	2	5

Tablica 9.5: Porównanie operatorów selekcji

Z poniższej tabeli wynika, że najczęściej najlepsze rezultaty uzyskano, gdy wybrana była selekcja rankingowa, z parametrem η równym 2, czyli dla wartości największej możliwej i najbardziej premiującej rozwiązania lepsze od gorszych. Co ciekawe, dla dwóch instancji testowych mimo ustawienia parametru η na wartość 1 udało się uzyskać lepsze rezultaty niż dla wartości innych. W takim przypadku rozwiązania są wybierane losowo, gdyż każde z nich ma takie same prawdopodobieństwo bycia wybranym. Sytuacja ta mogła być wynikiem tego, że przy danym rozmiarze populacji różnice w prawdopodobieństwach wyboru rozwiązań dla parametru η różnego od 1 i dla selekcji ruletkowej są małe i rozwiązanie najgorsze w pokoleniu ma niewiele mniejszą szansę bycia wybranym niż najlepsze. Z tego powodu selekcja w każdym z możliwych przypadków przebiega dość podobnie.

9.6. Najlepsze parametry

Ostatni podsumowujący eksperyment miał na celu zweryfikowanie, czy uzyskane najlepsze wartości parametrów w każdym z wcześniejszych testów, ustawione razem poprawią uzyskiwane rezultaty. Poniższa tabela zawiera informacje na temat tego, ile razy dla danej instancji testowej udało się uzyskać poprawę, a ile razy wynik był gorszy w stosunku do poprzednich eksperymentów:

instancja testowa	ile razy lepszy wynik	ile razy gorszy wynik
bur26a	4	1
chr15b	2	3
chr18b	2	3
chr25a	1	4
esc64a	4	1
had12	5	0
kra32	0	5
lipa20a	3	2
lipa50b	0	5
lipa80a	1	4
lipa90a	2	3
Σ	24	31

Tablica 9.6: Porównanie wyników eksperymentów

Powyższa tabela nie pozwala stwierdzić, że ustawienie parametrów na wartości uznane w poprzednich eksperymentach za najlepsze, spowodowało ogólne polepszenie działania algorytmu. Jednakże nie pozwala ona również stwierdzić, że taki dobór parametrów spowodował pogorszenie wyników. Zaistniała sytuacja może mieć swoją przyczynę w tym, że pewne parametry dobrane razem dadzą gorsze wyniki niż, gdyby każdy z nich był ustawiony na swoją teoretycznie najlepszą wartość w osobnych testach. Należy jednak wspomnieć, że różnice pomiędzy wcześniejszymi testami a testem „podsumowującym" były niewielkie.

9.7. Podsumowanie eksperymentów obliczeniowych

Przeprowadzone eksperymenty przy użyciu napisanej aplikacji zweryfikowały poprawności jej działania - algorytm działa poprawnie, w większości przypadków otrzymane rezultaty były dość dobre. Jeden z testów pozwolił nawet na znalezienie rozwiązania optymalnego. Jednakże dla pewnych instancji testowych otrzymywane wyniki były złe, niezależnie od dobranych parametrów algorytmu.

Niewielkie różnice pomiędzy uzyskiwanymi rezultatami eksperymentów pozwalają stwierdzić, że choć istnieją różne wersje operatorów genetycznych, każda z nich pozwala na uzyskiwanie dobrych

rezultatów. Liczba parametrów algorytmu, które można zmienić oraz ich zakres, nie pozwolił na przetestowanie każdej możliwej opcji, dlatego należy jeszcze raz podkreślić, że wartości uznane za najlepsze, najlepszymi były tylko w obrębie danego eksperymentu, co niejako potwierdza eksperyment podsumowujący. Jednak rezultaty testów pozwalają na wyrobienie pewnej intuicji, dzięki której możliwy będzie właściwy dobór parametrów w przypadku rozwiązywania problemów, dla których nieznane jest rozwiązanie optymalne.

10. Podsumowanie i wnioski

Celem niniejszej pracy było dokonanie przeglądu algorytmów przybliżonych w kontekście problemu przydziału kwadratowego oraz prezentacja rezultatów eksperymentów przeprowadzonych przy pomocy napisanej aplikacji komputerowej wykorzystującej wybrany algorytm aproksymacyjny. Dla każdego z zaprezentowanych algorytmów został również przedstawiony możliwy sposób jego wykorzystania w celu rozwiązania problemu QAP. Sposoby te nieraz wymagały specyficznego podejścia do problemu QAP i stosowania ciekawych technik przekształcających problem wyjściowy na taki, który możliwy będzie do rozwiązania przy wykorzystania danego algorytmu. Częste trudności ze znalezieniem prostych sposobów wykorzystania niektórych algorytmów przybliżonych, by rozwiązać problem QAP czy też inny problem, którego rozwiązaniami są permutacje, pokazują, że szeroka gama dostępnych algorytmów przybliżonych jest potrzebna. Nie każdy znany algorytm nadaje się dobrze do rozwiązywania każdego możliwego problemu. Algorytmy mrówkowy i PSO nadają się dobrze do problemów binarnych lub całkowitoliczbowych, w których rozwiązanie nie jest permutacją. Natomiast z tymi rozwiązaniami, które są permutacją, dobrze i w prosty sposób radzą sobie algorytmy genetyczne i tabu search, choć kodowanie rozwiązań w opisywanym algorytmie NPQGA nie jest łatwe.

Ciekawym wnioskiem związanym z algorytmami aproksymacyjnymi jest to, że w dużej mierze swoje pochodzenie zawdzięczają one licznym obserwacjom otaczającego nas świata i próbom wykorzystania mechanizmów działających wokół nas w celu rozwiązywania wielu problemów. Należy tutaj podkreślić kunszt ich twórców, którzy niejednokrotnie po latach badań potrafili wykorzystać swoje obserwacje i stworzyć liczne algorytmy wzorując się na naturze.

Zapoznanie się ze sposobami działania wspomnianych algorytmów pozwoliło stwierdzić, że są one niezwykle potężnym narzędziem rozwiązywania zagadnień, dla których algorytmy dokładne nie miałby możliwości osiągnięcia sukcesu. Jednakże brak znajomości rozwiązania optymalnego rozwiązywanego problemu nie pozwala na jednoznaczną ocenę poprawności rezultatów, które algorytmy przybliżone pozwalają uzyskać. Weryfikacja poprawności znalezionych rozwiązań wiązać się musi z wielokrotnym powtarzaniem eksperymentów i prowadzeniem często żmudnych analiz statystycznych.

Proces tworzenia aplikacji oraz przeprowadzone dzięki niej eksperymenty pozwoliły na dogłębne zapoznanie się z ideą algorytmów genetycznych i ich modyfikacji. Natura działania algorytmów genetycznych nadaje się dobrze do wprowadzania różnorakich zmian i ulepszeń, czego ciekawym przykładem jest wykorzystany w programie algorytm NPQGA, który już będąc wariacją na temat algorytmów genetycznych również uległ licznym modyfikacjom. Zastosowanie bitów kwantowych do reprezenta-

cji rozwiązań problemu pozwoliło na uzyskiwanie wielu nieraz zaskakujących rezultatów. Obserwacja stanów kubitów może powodować, że dwa identyczne ciągi bitów kwantowych mogą reprezentować zupełnie inne rozwiązanie problemu. Na uniknięcie całkowitej losowości otrzymywanych rozwiązań ma wpływ nietypowy operator genetyczny, jakim jest bramka kwantowa. „Nakierowuje" ona rozwiązania na rozwiązania lepsze od nich, zwiększając prawdopodobieństwo, że ich kubity znajda się w stanie takim samym, jak analogiczne kubity rozwiązania lepszego.

Być może przykład algorytmu NPQGA jest jednym z lepszych obrazujących wpływ losowości (pseudolosowości) na uzyskiwanie rozwiązań w algorytmach przybliżonych. Natura działania wielu algorytmów aproksymacyjnych wymaga stosowania liczb losowych (pseudolosowych), co wiąże się ze stosowaniem różnorakich metod generowania takich liczb. Dlatego ważnym elementem są generatory liczb pseudolosowych - zbyt częste generowanie może doprowadzić do uzyskiwania gorszych wyników. Również element losowości powoduje, że algorytmy przybliżone w większości przypadków nie dadzą tych samych rezultatów dla tych samych ustawień początkowych. Korzystanie z tych algorytmów wiąże się więc z wielokrotnym powtarzaniem eksperymentów w celu zapewnienia większej poprawności uzyskiwanych rozwiązań. Dlatego przeprowadzane w ramach celów pracy eksperymenty polegały na wielokrotnym uruchamianiu algorytmu dla tej samej populacji startowej i tych samych jego parametrów oraz analizowaniu wyników uzyskanych na podstawie wielu testów.

Uzyskane rezultaty eksperymentów pozwoliły stwierdzić, że aplikacja została napisana w sposób poprawny, algorytm działa zgodnie z przewidywaniami, a uzyskiwane różnice w stosunku do znanych rozwiązań optymalnych nie są w większości sytuacji duże. Jednak dla pewnych instancji testowych różnice były spore. Podyktowane to mogło być wprowadzonymi do algorytmu modyfikacjom oraz faktem, że algorytm pierwotnie dedykowany był zagadnieniu szeregowania zadań. Nie bez wpływu również mogło być to, że wiele elementów dotyczących implementacji algorytmu nie zostało doprecyzowanych przez samych jego autorów, czego rezultatem było zastosowanie rozwiązań, które mogły faktycznie odbiegać od zamierzeń twórców algorytmu.

DODATEK A

Qbit
Alpha : double Beta : double
Qbit() Qbit(anotherQbit: Qbit) Qbit(alpha: double, beta: double) EvaluateState(): int ExecuteRotationGate(theta: double): void ExecuteNotGate(): void

Chromosome
PermuationValue : int Value : int Size: int this[index: int]: Qbit
Chromosome(size: int) Chromosome(anotherChromosome: Chromosome) DecodeChromosome(): int

Solution
Size: int this[index: int]: Chromosome Goal: double
Solution(size: int) Solution(anotherSolution: Solution) Solution(chromosomes: Chromosome[]) ToPermutation(): void PrintSolution(): void PrintSolution(file: System.IO.StreamWriter): void

Population
ProblemSize: int this[index: int]: ISolution CurrentSize: int
Population() Population(solutionsCount: int, problemSize: int) Population(anotherPopulation: IPopulation) Population(solutions: Isolution[]) SortAscending(): void SortDescending(): void Add(sol: ISolution): int Remove(idx: int): void GetEnumerator(): IEnumerator

MutationOperator
MutationProbability: double
MutationOperator(probability: double = 0.0) Execute(Population: IPopulation): ISolution

CxCrossoverOperator
CrossoverProbability: double
CxCrossoverOperator(probability: double = 0.0) Execute(population: Population): ISolution[] Execute(parentOne: Solution, parentTwo: Solution): Solution[]

OxCrossoverOperator
CrossoverProbability: double
OxCrossoverOperator(probability: double = 0.0) Execute(population: Population): ISolution[] Execute(parentOne: Solution, parentTwo: Solution): Solution[]

PmxCrossoverOperator
CrossoverProbability: double
PmxCrossoverOperator(probability: double = 0.0) Execute(population: Population): ISolution[] Execute(parentOne: Solution, parentTwo: Solution): Solution[]

RotationGateOperator
RotationGateOperator() ExecuteOriginal(population: IPopulation, best: Soultion): void ExecuteModified(population: IPopulation, best: Soultion)

QapData
Insstance: QapData
SetQapData(distance: doub;e[,], flow: double[,], size: int) GetDistance(): double[,] GetFlow(): double[,] GetSize(): int

SelectionOperator
EtaMax: double
SelectionOperator(etaMax: double = 2.0) RouletterMethod(population: IPopulation): Population RankingMethod(population: IPopulation): Population

QgAlgorithm
Population: IPopulation
QgAlgorithm() SetParameters(popSize: int, iterations: int, selMethod: SelectionMethdChosen, eta: double, crossMethod: CrossoverOperatorChosen, crossMax: double, crossMin: double, mutProb: double, rotVersion: RotationGateVersion, initNew: bool): bool EvaluateSolutionsolution: ISoultion: double InitRandomPopulation(): void Execute(): void Stop(): bool GetBestSolution(): ISolution GetBestInitialSolution(): Soultion GetBestIteration(): int

UserInterface
RunTest(algorithm: QgAlgorithm): void ReadQapFile(path: string): bool

Bibliografia

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

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

[3] 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.

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

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

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

[7] 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.

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

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

[10] Jones Gareth, *Genetic and Evolutionary Algorithms*, 1990.

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

[12] Kennedy James, Eberhart Russel, *Particle Swarm Optimization*, Materiały IEEE International Conference on Neural Networks, 4, 1942-1948, 1995.

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

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

[15] Liu Hongbo, Abraham Ajith, Zhang Jianying *A Particle Swarm Approach to Quadratic Assignment Problems*. [w:] Soft Computing in Industrial Applications, pod red. Saad A., Dahal K., Sarfraz M., Roy R., Springer, Berlin 2007.

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

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

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

[19] Stützle Thomas, Dorigo Marco, *ACO Algorithms for the Quadratic Assignment Problem*. [w:] D. Corne, M. Dorigo, F. Glover, New Ideas for Optimization, McGraw-Hill, 1999, 33-50.

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

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