SPIS TREŚCI

WYKAZ NAJWAŻNIEJSZYCH SKRÓTÓW I OZNACZEŃ

CAD	**C**omputer **A**ided **D**esign Komputerowe wspomaganie projektowania
CAM	**C**omputer **A**ided **M**anufacturing Komputerowe wspomaganie wytwarzania
CETSP	**C**lose **E**nough **TSP** Problem komiwojażera z założeniem promienia bliskości
FI	**F**arthest **I**nsertion Metoda najdalszego włączenia
GA	**G**enetic **A**lgorithm Algorytm genetyczny - podejście ewolucyjne
GTSP	**G**eneralized **TSP** Uogólniona postać zadania komiwojażera
NN	**N**earest **N**eighbour Metoda najbliższego sąsiada
OX	**O**rdered **C**rossover Operator krzyżowania z zachowaniem względnej kolejności alleli
PCB	**P**rinted **C**ircuit **B**oard Obwód drukowany - płyta do montażu podzespołów elektronicznych
RBA	**R**ubber **B**and **A**lgorithm Algorytm rozwiązywania zadania TPP
RSM	**R**everse **S**equence **M**utation Operator mutacji - odwróconej sekwencji alleli
TPP	**T**ouring a sequence of **P**olygons **P**roblem Problem wyznaczanie ścieżki prowadzącej przez sekwencję wielokątów
TSP	**T**raveling **S**alesman **P**roblem Problem komiwojażera
TSPN	**TSP** with **N**eighbourhood Problem komiwojażera z sąsiedztwem (otoczeniem) punktów docelowych

WSTĘP

Niniejsza praca stanowi kontynuację pracy inżynierskiej Autora i poświęcona jest dalszemu rozwojowi oprogramowania *PCB CAM Processor* [31]. Praca porusza ważne, pominięte wcześniej kwestie związane z optymalizacją generowanych ścieżek narzędzia.

Pierwszy rozdział stanowi wprowadzenie w tematykę optymalizacji trajektorii robotów przemysłowych. Ukazuje różnorodność podejść do zaganiania oraz obszary zastosowań danych kryteriów optymalizacji. Kolejne rozdziały zawierają sformułowanie problemów optymalizacji występujących w omawianym oprogramowaniu oraz prezentują metody rozwiązywania owych zadań. Trzeci rozdział szczegółowo opisuje problematykę wyznaczanie najkrótszych ścieżek w zadaniu komiwojażera oraz prezentuje rzeczywiste implementacje tego problemu do optymalizacji procesu wiercenia. Ostatni rozdział pracy poświęcony jest bardzo ciekawemu problemowi, optymalizacji ścieżki narzędzia prowadzącej przez konturu podlegające zabiegom takim jak grawerowanie czy wycinanie. Oprócz opisu problematyki optymalizacji wspomnianych procesów, praca zawiera również szczegółowe implementacje różnorodnych podejść do zagadnienia oraz prezentuje wyniki poszczególnych metod wraz z ich wzajemnym porównaniem. W pracy poszukiwane są efektywne metody minimalizacji czasu obróbki, czyli takie, które w zadowalającym czasie są w stanie generować rozwiązania bliskie optymalnym.

Cel i zakres pracy:

- Charakterystyka zagadnienia,
- Metody konwencjonalne oraz metody sztucznej inteligencji w rozwiązywaniu zadań optymalizacji,
- Opracowanie oprogramowania wybranych metod,
- Przeprowadzenie badań doświadczalnych nad wydajnością i jakością badanych algorytmów optymalizacji,
- Podsumowanie i wnioski.

1. OPTYMALIZACJA TRAJEKTORII ROBOTÓW PRZEMYSŁOWYCH

Roboty przemysłowe znajdują dziś zastosowanie w niemal każdej dziedzinie przemysłu. Jest to możliwe dzięki olbrzymiemu zróżnicowaniu ich konstrukcji. Istnieją różne typy robotów przemysłowych. Jedna z klasyfikacji zaproponowana przez Spong et al. [33] dzieli roboty ze względu na różne kryteria takie jak: źródło zasilania lub sposób poruszania się poszczególnych par kinematycznych, strukturę kinematyczną, udźwig, przestrzeń roboczą, metody sterowania czy obszary zastosowań.

Obecnie kładzie się również olbrzymi nacisk na rozwój procesów produkcyjnych przyjaznych środowisku. Technologie proekologiczne są wprowadzane na każdym etapie życia produktu, od projektu, aż po jego utylizację. Prowadzonych jest wiele badań z zakresu robotyki mających na celu redukcje zużycia energii, szczególnie poprzez optymalizację ruchów robotów [36]. Constantinescu i Croft [10] pokazują, że optymalizacja poprzez wygładzanie trajektorii prowadzi do znacznego polepszenia wydajności oraz zmniejszenia zużycia energii.

Optymalizacja ruchu obiektu manipulowanego przez robota przemysłowego pomiędzy dwoma zadanymi pozycjami, obejmuje zarówno optymalizację ruchu manipulowanego obiektu (względnie efektora końcowego) względem podstawy robota, jak również wybór optymalnego posadowienia robota w przestrzeni roboczej. Umiejscowienie robota może być dobierane do aktualnej aplikacji lub sama aplikacja może być tak projektowana, aby zajmowała optymalne miejsce w przestrzeni roboczej wybranego robota. Spensieri et al. [32] wykazują eksperymentalnie, że optymalne umiejscowienie robota na stanowisku montażowym w branży samochodowej, zależnie od wykonywanych zadań może prowadzić do zredukowanie czasu operacyjnego nawet o 20%.

Celem tego typu optymalizacji jest minimalizacja lub maksymalizacja przynajmniej jednej z poniższych funkcji celu [28]:

- Minimalizacja czasu operacyjnego, względnie maksymalizacja produktywności robota przemysłowego, uwzględniając ograniczenia w postaci maksymalnych prędkości poszczególnych członów robota.
- Minimalizacji zużycia energii lub pracy mechanicznej wymaganej do wykonania zadania, co prowadzi do zmniejszenia obciążeń mechanicznych napędów i konstrukcji robota oraz do osiągnięcia gładkich trajektorii ruchu.
- Minimalizacja maksymalnej mocy chwilowej wymaganej do pracy robota.

- Minimalizacja maksymalnych sił i momentów generowanych przez napędy robota. Takie podejście prowadzi do zmniejszenia awaryjności i wydłużenia żywotności maszyny.

Często spotykanym podejściem jest optymalizacja wielokryterialna mająca na celu równoległą optymalizację kilku z powyższych funkcji celu. Taka realizacja pozwala osiągać znacznie lepsze rezultaty.

Do najczęściej opisywanych w literaturze kryteriów optymalizacji trajektorii robotów przemysłowych należą [28]: minimalizacja czasu przejazdu, minimalizacja zużycia energii lub zaangażowania napędów, minimalizacja szarpnięć oraz kryteria złożone takie jak np. minimalizacja czasu i energii, gdzie należy zachować kompromis pomiędzy czasem potrzebnym na przebycie wymaganej trajektorii, a energią zużywaną przez napędy lub równoczesna minimalizacja czasu i szarpnięć.

Po przeprowadzeniu badań z zakresu przemysłowych zastosowań, dostępnych metod optymalizacji trajektorii, Kim i Croft [17] stwierdzili, że wszelkie metody mogą mieć ograniczone zastosowanie lub być wręcz niemożliwe do wdrożenia, jeśli nie weźmie się pod uwagę kilku ważnych kwestii. Zwrócono uwagę na:

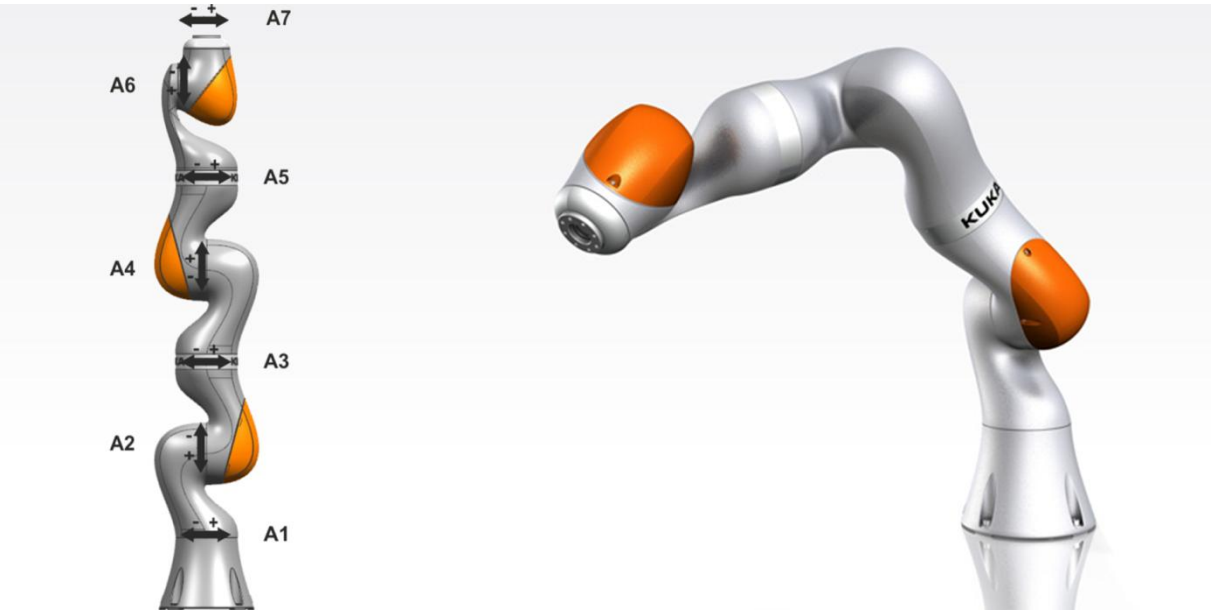
- Dokładność śledzenia trajektorii (wygładzanie śnieżki, może się w niektórych przypadkach wiązać ze znacznym pogorszeniem dokładności).
- Ważność kryterium minimalnego czasu, jako nadrzędnego w kontekście wydajności produkcji oraz dochodowości przedsiębiorstwa.
- Potrzeba optymalizowania cykli produkcyjnych w technice programowania *online*, czyli bezpośrednio na robocie. Zaawansowane metody optymalizacji, szczególnie wielokryterialnej wymagają dużych mocy obliczeniowych oraz mogą być bardzo czasochłonne, dlatego nie jest możliwa ich bezpośrednia implementacja w układach sterowania robotów.

1.1. Minimalno-czasowe planowanie trajektorii

Optymalne planowanie trajektorii pod względem minimalizacji czasu operacyjnego jest najczęściej poruszanym w literaturze podejściem do problemu [28]. Jest to oczywiście związane z tym, że to kryterium bezpośrednio przekłada się na skrócenie czasu produkcji i odpowiednio wzrost wydajności procesu.

Na przestrzeni wielu lat badań nad problemem wprowadzano mnóstwo technik oraz różnorodnych algorytmów optymalizacji. Niestety większość z nich zaniedbuje dynamikę napędów, co powoduje występowanie nieciągłości przebiegów czasowych momentów napędowych i przyspieszeń. Aby temu zapobiec, Constantinescu i Croft [10] proponują rozwiązanie problemu przy użyciu metody elastycznej tolerancji. Ograniczając maksymalne momenty obrotowe, uzyskują gładkie trajektorie, które stanowią optymalne pod względem czasu, lecz technicznie wykonalne ruchy. Gao et al. [14] wykorzystują ulepszone algorytmy uczenia maszynowego, które uwzględniają ograniczenia kinematyczne. Abu-Dakka et al. [2], po wyczerpującej analizie metod optymalizacji prezentowanych w literaturze, opowiadają się za minimalizacją czasu przy użyciu algorytmów genetycznych z uwzględnieniem ograniczeń kinematycznych, dynamicznych oraz obciążeniowych zamiast bezpośredniego skupiania się na minimalizacji szarpnięć i udarów.

W kontekście robotów przemysłowych o kinematyce nadmiarowej (tj. posiadających więcej niż 6 stopni swobody) stwierdzono, że tego typu konstrukcje posiadają pewne zalety w porównaniu z konwencjonalnymi robotami. Istnieją liczne badania poświęcone optymalizacji ich ruchów. Reiter et al. [30] prezentują metodę wyznaczania minimalnych czasowo oraz gładkich trajektorii typu *splajn B* przy użyciu standardowych metod optymalizacji oraz analitycznego podejścia do kinematyki odwrotnej i pewnej dekompozycji przestrzeni roboczej.



Rys. 1.1. Robot przemysłowy o kinematyce nadmiarowej - KUKA LBR iiwa [38]

Erdős et al. [12] przedstawiają model oraz metodę optymalizowania ruchów złączowych robotów o kinematyce nadmiarowej, które pracują w złożonym środowisku pracy z licznymi ograniczeniami geometrycznymi i technologicznymi. Metoda zakłada wyznaczanie zestawu dopuszczanych układów robota na drodze rozwiązania zadania kinematyki odwrotnej oraz wybór takiego układu, który spełnia wszelkie ograniczenia i zapewnia uzyskanie trajektorii minimalnej czasowo.

1.2. Minimalizacja zużycia energii

Optymalizacja trajektorii poprzez minimalizację energii posiada szereg zalet. Z jednej strony prowadzi do otrzymania gładkich krzywych, które są pożądane z uwagi na redukcję obciążeń mechanicznych napędów oraz konstrukcji robota. Z drugiej strony takie podejście prowadzi do oszczędzania energii, co jest istotne nie tylko ze względów ekonomicznych, ale może mieć również znaczenie tam, gdzie występują pewne ograniczenia energetyczne źródeł zasilania [28].

Tego typu podejście do zaganiania optymalizacji aplikacji przemysłowych znajduje ostatnio szerokie zastosowanie z uwagi na potrzebę zmniejszenia zużycia energii. Wielu współczesnych badaczy zajmuje się tą tematyką: Mashimo et al. [22], Meike et al. [24], Wigstrom et al. [37] oraz Gregory et al. [16]. W Swoich badaniach wykorzystują różnorodne algorytmy oraz metody planowania trajektorii, włączając w to modelowanie oraz symulacje komputerowe. Niektóre z tych badań wciąż nie uzyskują gładkich przebiegów przyspieszeń oraz momentów napędowych, natomiast w kilku następnych pracach badacze uniknęli tych problemów. Jedne z takich badań prowadzone przez Luo et al. [21], wykorzystują metodę interpolacji Langrange'a, aby wyrazić funkcję ruchu każdego złącza (pary kinematycznej obrotowej robota przemysłowego o strukturze szeregowej) i przeprowadzić proces planowania trajektorii. Na zadanie nakładane są ograniczenia pozycji oraz stabilności robota. Poprzez całkowanie wyprowadzonych równań ruchu obliczane są prędkości kątowe oraz przyspieszenia kątowe. Przytoczona metoda pozwala uzyskać gładkie przebiegi czasowe pozycji, przyspieszeń i momentów obrotowych, które spełniają nałożone ograniczenia. Wyznaczone tą metodą funkcje można uwzględnić w procesie obliczania zużycia energii i na drodze dalszej optymalizacji wyznaczyć optymalną pod względem energetycznym trajektorię ruchu robota. Fung i Cheng [13] w swoich pracach uzyskują trajektorię punkt po punkcie, opartą na minimalnej energii wejściowej robota. Model matematyczny zakłada opis

zagadnienia za pomocą wielomianów wysokiego stopnia, których współczynniki są dobierane za pomocą algorytmów genetycznych. Opis zakłada ograniczenia w postaci przemieszczeń kątowych, prędkości, przyspieszeń oraz szarpnięć.

1.3. Minimalizacja szarpnięć

Szarpnięcie jest trzecią pochodną przemieszczenia względem czasu (tj. pierwszą pochodną przyspieszenia) i wyraża szybkość zmian przyspieszeń, a co za tym idzie sił oraz momentów sił występujących w konstrukcjach mechanicznych. Minimalizacja szarpnięć prowadzi do wyznaczania gładkich trajektorii ruchu oraz zmniejszenia błędów pozycjonowania osi robota, a także do redukcji wibracji konstrukcji, co przekłada się na zmniejszenie zużycia i wydłużenie czasu eksploatacji maszyny.

Planowanie optymalnej trajektorii minimalizującej szarpnięcia odbywa się przy użyciu specjalizowanych metod, które można podzielić, ze względu na podejście, na dwie grupy [28]: metody wykorzystujące predefiniowane punkty przelotowe, zwane węzłami, punktami węzłowymi lub punktami kontrolnymi oraz metody niewykorzystujące takich punktów. Lin [19] przedstawia jedną z metod wykorzystującą punkty pośrednie, która w szybki i jednolity sposób pozwala wyznaczyć trajektorię minimalnych szarpnięć. Do wyznaczania trajektorii brane są pod uwagę początkowe, pośrednie i końcowe przemieszczenia oraz prędkości kątowe osi robota. Metoda wykorzystuje podejście oparte na optymalizacji roju cząstek (ang. PSO - Particle Swarm Optimization), które zdaniem autora, różni się od poprzednich prac poświęconych tej tematyce, ujednoliconą metodologią. Przeprowadzone symulacje komputerowe potwierdzają prawidłowe działanie metody dla robota o sześciu stopniach swobody.

1.4. Optymalizacja wielokryterialna

Podejście uwzględniające więcej niż jedno kryterium optymalizacji wydaje się być najodpowiedniejsze do zastosowań w procesie planowania trajektorii robota przemysłowego. Minimalizacja energii lub obciążeń dynamicznych bez uwzględnienia czasu rozmija się nieco z rzeczywistością. W realnych aplikacjach przemysłowych, czas stanowi kluczowe kryterium wydajności procesu produkcyjnego. Przytoczone dalej badania rozważają optymalizację pod względem kilku kryteriów, bez pominięcia czasu.

Stosując kombinację minimalizacji energii oraz czasu, która stanowi kompromis pomiędzy czasem potrzebnym na wykonanie zadanej trajektorii, a energią zużywaną przez napędy, Gleeson et al. [15] pokazują, że można osiągnąć zmniejszenie zużycia energii na poziomie 10% w standardowej aplikacji zrobotyzowanego stanowiska spawalniczego. Kolejne hybrydowe podejście mające na celu minimalizację czasu oraz szarpnięć zostało zastosowane w badaniach Liu et al. [20] w celu uzyskania płynnych i wydajnych ruchów robota. Podobne badania prowadzone przez Perumaal i Jawahar [27] prowadzą do określenia sposobów wyznaczania płynnej i optymalnej pod względem czasu trajektorii robota do zadań typu *pick and place*. Autorzy wskazują, że tego typu operacje, takie jak paletyzacja, depaletyzacja czy zgrzewanie punktowe mają znaczący udział wśród wszystkich aplikacji zrobotyzowanych.

Inne zastosowanie wielokryterialnego podejścia do procesu optymalizacji znalazło miejsce w złożonym zadaniu montażu w realnej aplikacji przemysłowej, wymagało ono optymalizacji relacji pomiędzy prędkością, a długością trajektorii. Zadaniem robota było wkręcenie ośmiu (wcześniej wstępnie wkręconych) śrub mocujących pewien element silnika spalinowego na głębokość 10 mm, a następnie dokręcenie momentem 25 Nm. Robot musiał wykonać to zadanie w czasie maksymalnie 33 s. Anton et al. [5] opracowali algorytm oparty na równaniach dynamiki robota, biorący pod uwagę obciążenie robota wynikające z zamocowanego na nim narzędzia oraz położenie punktów zatrzymania.

Bardzo kompleksowe podejście do problemu zaprezentowali Chiddarwar i Babu [8]. W swoich badaniach nad wyznaczaniem optymalnej trajektorii robota wzdłuż określonej ścieżki wprowadzili aż 6 funkcji celu, łącznie 67 ograniczeń i 48 zmiennych. Metoda bierze pod uwagę: czas, momenty napędowe, unikanie osobliwości, szarpnięcia, przyspieszenia osi oraz siły chwytania.

1.5. Optymalizacja sekwencji zadań

W wielu przemysłowych aplikacjach roboty wykonują pewne zestawy zadań. Kolejność ich wykonywania oraz sposób przemieszczania się robota pomiędzy tymi zadaniami mogą mieć znaczny wpływ na ogólną wydajność aplikacji. Niestety, w wielu przypadkach wciąż dokonuje się pewnego rodzaju ręcznej optymalizacji. Programiści robotów opracowując trajektorię kierują się doświadczeniem i swoistą intuicją [28]. Takie podejście może jednak nieść za sobą ryzyko błędów, a wielokrotne testowanie różnych

wariantów w celu znalezienia lepszego rozwiązania jest czasochłonne, czyli kosztowne. W ostatnich latach problemem zajęli się badacze próbujący znaleźć algorytmy umożliwiające generowanie optymalnej trajektorii. Zadanie nie jest jednak łatwe. Ponownie należy tu uwzględnić wiele czynników takich jak: unikanie kolizji, osobliwości, złożone funkcje celu, niejednoznaczność wykonania zadania (optymalne rozwiązanie danego zadania zależy od jego miejsca w sekwencji) czy umiejscowienie robota na stanowisku. Jak prezentują Alatartsev et al. [4], trudno jest porównać istniejące podejścia do zagadnienia, ponieważ poświęcone są różnym przypadkom, branżom oraz wykorzystują różne typy robotów, ich cechy i rodzaje ograniczeń.

Dzięki temu, że wiele zadań posiada pewną swobodę wykonania, istnieje szerokie pole do prowadzenia badań nad poszukiwaniem bardziej efektywnych sekwencji zadań. W jednym z takich badań Alatartsev et al. [3] proponują nowy, skuteczny sposób rozwiązania zadania sekwencjonowania. Zadanie zakłada istnienie pewnej grupy zamkniętych konturów, które mogą być poddawane takim operacją jak wycinanie, grawerowanie czy spawanie łukowe. Tak zdefiniowany kontur posiada pewien punkt będący jednocześnie początkiem i końcem operacji. Punkt może być swobodnie dobrany wzdłuż całego konturu. Zadanie wymaga wyznaczania takich punktów na każdym z konturów, aby po rozwiązaniu zadania komiwojażera (ang. TSP - Traveling Salesman Problem) pomiędzy tymi punktami, uzyskać jak najkrótszą ścieżkę.

Inny problem wyznaczania optymalnej sekwencji zadań oraz planowania trajektorii został opisany przez Kovács [18]. Problem odnosi się do aplikacji zrobotyzowanego spawania w technologii RLW (Remote Laser Welding), która zakłada wykonywanie spoin o długości od 15 do 30 mm. Każdą spoinę można potraktować jako zadanie do wykonania w pewnej sekwencji. Robot z danego ustawienia jest w stanie wykonać pewną grupę spoin, która znajduje się w zasięgu lasera oraz spełnia pewne dodatkowe ograniczenia m. in. kąt padania wiązki lasera. Zadanie jest bardzo złożone, ponieważ wymaga takiego prowadzenia głowicy laserowej, aby zapewnić jej możliwość wykonania wszystkich spoin. Należy wyznaczyć pełną trajektorię głowicy, uwzględniając pozycję, orientację oraz prędkość poruszania. W obszarach gdzie do wykonania jest więcej spoin głowica musi się poruszać wolniej, a tam gdzie liczba spoin jest niewielka może przyspieszyć. Równolegle z planowaniem trajektorii ruchu głowicy należy jeszcze rozwiązać szereg zadań typu TSPND (ang. Traveling Salesman Problem with Neighbourhoods and Duration visits), czyli zadanie komiwojażera uwzględniające czas pobytu oraz pewne otoczenie zamiast ścisłego punktu docelowego.

Podsumowując, uzyskujemy pewną sekwencję ruchu głowicy i równoległą sekwencję względnego ruchu plamki laserowej.

1.6. Podsumowanie

Istnieją różnorodne podejście do problematyki optymalizacji trajektorii robotów przemysłowych. Ich wspólną cechą jest oczywiście maksymalizacja produktywności, którą można osiągnąć przede wszystkim poprzez minimalizację czasu operacyjnego. Należy jednak pamiętać o pewnych ograniczeniach konstrukcyjnych i uwzględnić również minimalizację obciążeń dynamicznych oraz zużycia energii. Rzeczywiste aplikacje przemysłowe wymagają złożonych podejść wielokryterialnych lub bazujących na sekwencji zadań, dlatego te dwie przedstawione wcześniej grupy podejść zasługują na szczególną uwagę.

W kolejnym rozdziale zostanie omówiony problem, który jest przedmiotem tej pracy. Problem wpisuje się w optymalizację sekwencji zadań pod względem minimalizacji czasu. Jednocześnie poszukiwane są metody, które same też muszą być optymalne, czyli dawać zadowalający wynik w krótkim czasie. Zatem, mamy tu do czynienia z wieloma kryteriami skupionymi wokół czasu. Poszukujemy optymalnego algorytmu minimalizującego czas wykonania zadań, który wyznacza optymalną ich sekwencję przy jednoczesnym doborze jednego z dostępnych wariantów dla każdego z zadań.

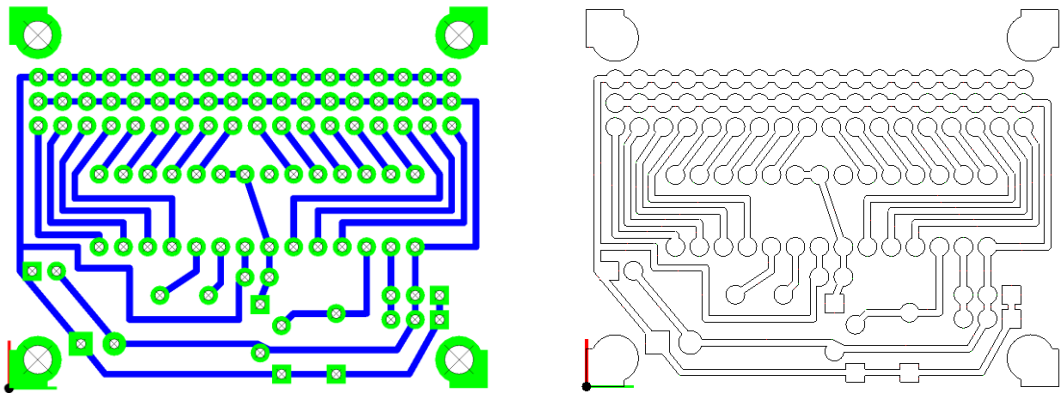
2. SFORMUŁOWANIE PROBLEMU OPTYMALIZACJI

Niniejszy rozdział stanowi przedstawienie problemów optymalizacji występujących podczas generowania programów obróbkowych przez oprogramowanie *PCB CAM Processor*, stanowiące przedmiot poprzedniej pracy autora [31]. Omawia również pokrótce samo oprogramowanie, aby nakreślić kontekst dalszych rozważań.

2.1. Oprogramowanie *PCB CAM Processor*

Program został opracowany z myślą o szybkim i tanim prototypowaniu prostych obwodów drukowanych PCB (ang. Printed Circuit Board) o montażu przewlekanym. Do wykonania prototypu wystarczy kawałek laminatu PCB, kilka narzędzi (frez grawerski i wiertła) oraz dowolna maszyna sterowana numerycznie wyposażona we wrzeciono. Próby z powodzeniem prowadzone były z wykorzystaniem robota KUKA oraz tokarki z układem sterowania Sinumerik [31].

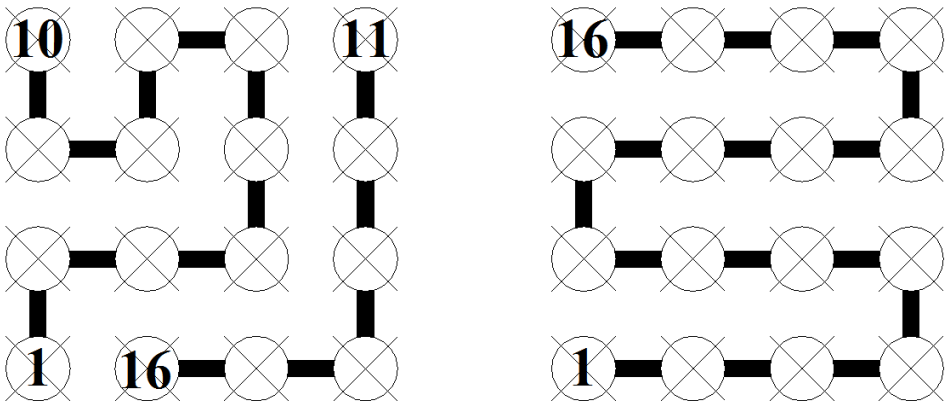
Oprogramowanie składa się z dwóch części: CAD oraz CAM. Moduł CAD umożliwia projektowanie obwodu oraz zapewnia wszelkie możliwości edycyjne takie jak zmiany parametrów czy pozycji umieszczanych obiektów. Moduł CAM, dzięki zastosowaniu specjalnych szablonów obróbkowych umożliwia generowanie programów obróbkowych na niemal dowolny typ maszyny sterowanej komputerowo. Szablony w prosty i przejrzysty sposób opisują język danej maszyny. Użytkownik definiuje w nich, w jaki sposób w danym języku zapisuję się np.: interpolację liniową lub kołową. Ostatnim krokiem jest wygenerowanie programów dla maszyny. Istnieje również możliwość podglądu wyniku oraz symulacji utworzonych sekwencji. Powstają dwa pliki tekstowe z programem, po jednym dla operacji wiercenia i grawerowania.



Rys. 2.1. Przykładowy model oraz jego obliczony kontur [31]

Podczas generowania programów obróbkowych rozwiązywane są dwa kolejne zadania: wyznaczanie kolejności wykonywania poszczególnych otworów oraz obliczanie wspólnych konturów dla operacji grawerowania. Wszystkie pola lutownicze oraz łączące je ścieżki są połączone i stanowią pola o wspólnym potencjale elektrycznym. Metodą grawerowania można niejako odciąć te pola od reszty laminatu PCB - rys. 2.1.

Wprowadzona dotychczas metoda optymalizacji zakłada poruszanie się do najbliższego niewykonanego dotąd elementu i jest metodą klasy *najbliższego sąsiedztwa* (ang. NN - Nearest Neighbour). Kontury składają się z elementarnych form geometrycznych zwanych prymitywami [31]. Ścieżka narzędzia rozpoczyna się na początku jednego z nich i kolejno prowadzi przez wszystkie prymitywy tworzące kontur. Następnie, narzędzie musi opuścić dany kontur i jałowym ruchem przemieścić się nad początek najbliższego niewykonanego dotąd prymitywu, należącego do innego konturu. Taka procedura wykonywana jest do momentu wykonania wszystkich konturów. Takie podejście nie zapewnia uzyskania dobrego rozwiązania, dlatego do wyznaczania kolejności otworów zostało wprowadzone pewne ulepszenie. Niektóre modele (rys. 2.1.), zawierają równomierne siatki otworów. Optymalne wykonanie takiego układu metodą NN jest niemożliwe, co obrazuje rysunek 2.2. Można jednak zaproponować, aby spośród najbliższych sąsiadów (tak samo odległych) wybierać tego, który nie wymaga zmiany kierunku ruchu.



Rys. 2.2. Losowa NN oraz optymalna NN kolejność wykonywania otworów w siatce 4x4 [31]

Stosowane dotychczas proste metody generują jedynie pseudo optymalne rozwiązania. Problem wymaga szerszego spojrzenia na zagadnienie. W dalszej części pracy zostaną szczegółowo sformułowane klasy problemów występujących w procesie generowania programów obróbkowych. Zaproponowane zostaną również lepsze metody oraz ich porównanie do metody NN.

2.2. Optymalizacja procesu wiercenia

Podczas wiercenia otworów na płaszczyźnie płytki PCB narzędzie porusza się pomiędzy zdefiniowanymi punktami, w których wykonywane są otwory. Każdy z tych punktów odwiedzany jest dokładnie jeden raz. Narzędzie posiada również jakąś pozycję bazową, z której rozpoczyna się cały proces i do której wraca po wykonaniu wszystkich otworów. Tak zdefiniowane zadanie wyznaczania sekwencji operacji ściśle wpisuje się w klasę problemu TSP, czyli problemu komiwojażera.

Problem komiwojażera został po raz pierwszy opisany i zbadany przez irlandzkiego matematyka Williama Hamiltona w XVIII w. [23]. Mimo wielu lat badań nad problemem nie udało się znaleźć metody znajdującej rozwiązanie dokładne w czasie wielomianowym, tj. czasie będącym wielomianem zmiennej *n*, czyli rozmiaru zadania. Problem, należy bowiem do klasy problemów NP-trudnych, tzn. takich, dla których nie istnieją takie algorytmy [35].

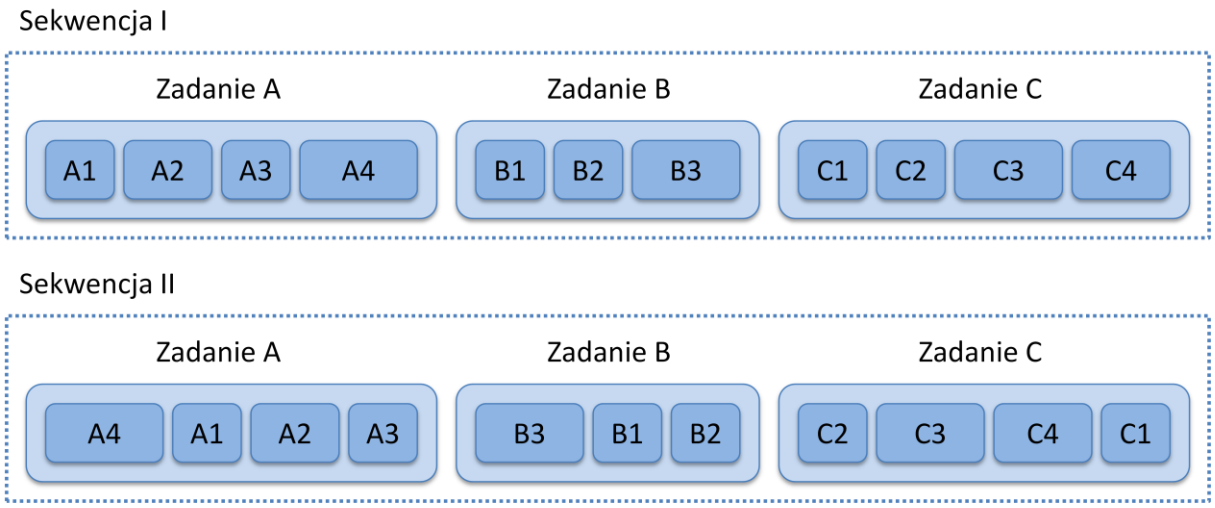
Zadanie można opisać przy użyciu teorii grafów. Zakładając, ze możliwy jest przejazd między każdą parą otworów oraz koszt takiej drogi stanowi jedynie odległość pomiędzy punktami, mamy do czynienia z pełnym grafem nieskierowanym. Zadanie sprowadza się do tzw. symetrycznego problemu komiwojażera (ang. sTSP - symmetric Traveling Salesman Problem). Grafem nieskierowanym postaci $G = (V, E)$ nazywamy strukturę złożoną ze skończonego zbioru *n* wierzchołków $V = \{v_1,...,v_i,...,v_n\}$ oraz skończonego zbioru *m* krawędzi $E = \{e_1,...e_j,...,e_m\}$ łączących nieuporządkowane pary wierzchołków. Dla grafu pełnego istnieją krawędzie łączące każdą parę wierzchołków.

Drogą lub inaczej ścieżką w grafie nazywamy ciąg niejednakowych krawędzi wyodrębnionych z grafu. Droga, której początek i koniec pokrywają się, nazywana jest cyklem. Cykl, który przebiega przez wszystkie wierzchołki, przy czym wierzchołki różnią się, nosi nazwę cyklu Hamiltona. W tak postawionym problemie rozwiązanie polega na znalezieniu najkrótszego z cykli Hamiltona.

2.3. Optymalizacja procesu grawerowania konturów

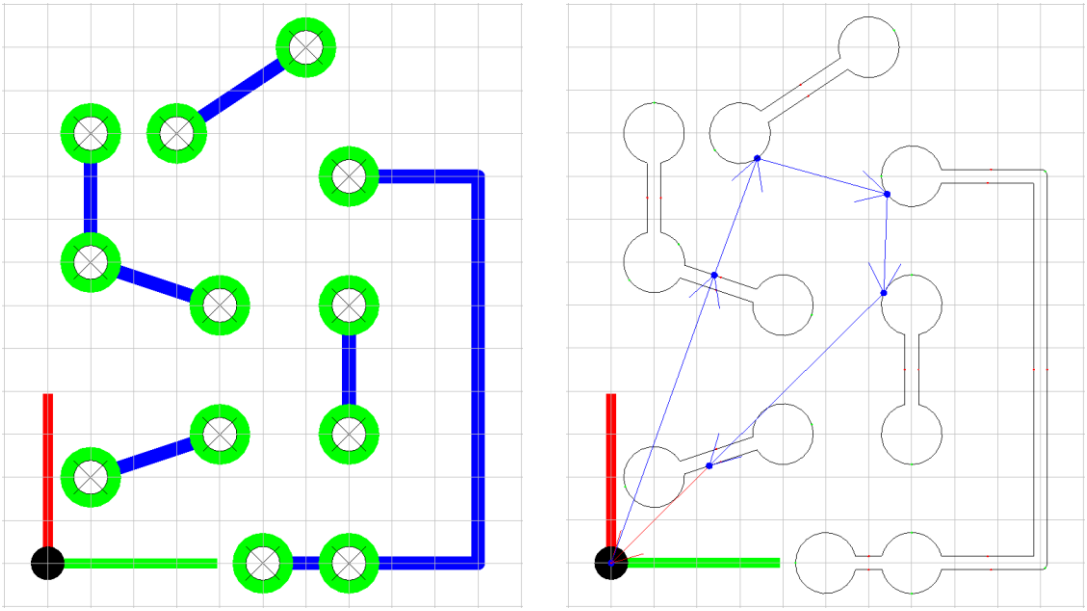
Podstawą wyprowadzenia problemu jest założenie, że wyznaczone przez oprogramowanie kontury są nierozłączne i stanowią pewnego rodzaju elementarne zadania. Każde z takich zadań można podzielić na kroki, które muszą być kolejno wykonane, aby ukończyć dane zadanie. Optymalna kolejność kroków jest prosta do wyznaczenia, natomiast

istnieje wiele wariantów wykonania danego zadania, zależnie od tego, który z kroków zostanie przyjęty jako pierwszy. Daje to pewnego rodzaju swobodę wyboru. Choć koszt pojedynczego zadania nie zależy od wyboru kroku startowego, to koszt (czas) wykonania całej sekwencji zadań jest już od niego uzależniony. Rysunek 2.3 pokazuje schematycznie opis problemu.



Rys. 2.3. Różnice w wyborze kroku startowego dla dwóch identycznych sekwencji

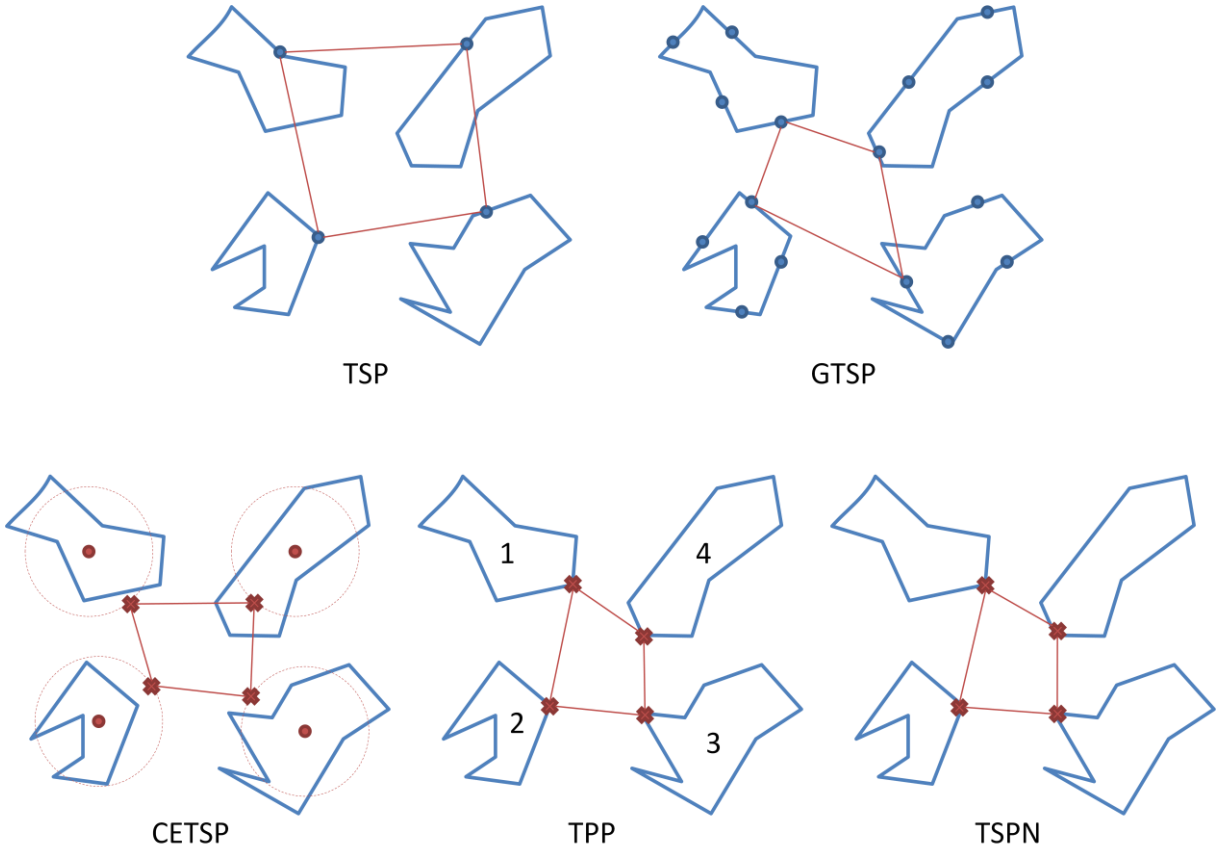
Obie sekwencje zawierają trzy zadania wykonane w kolejności: A, B, C. Zadania A oraz C składają się z czterech kroków, a zadanie B z trzech. Sekwencje są identyczne na poziomie wyboru kolejności zadań, natomiast różnią się w obrębie wyboru kroku startowego dla poszczególnych zadań. Koszty przejścia między kolejnymi zadaniami: A, B oraz B, C w obu przypadkach różnią się, zatem łączne koszty sekwencji I oraz II również nie są tożsame.



Rys. 2.4. Prosty model - poszukiwane rozwiązanie zadania

Oprogramowanie *PCB CAM Processor* generuje rozwiązanie w postaci zbioru konturów podlegających zabiegowi grawerowania. Kontur należy wykonać od pewnego punktu startowego i kolejno krok po kroku przeprowadzić narzędzie wzdłuż kolejnych prymitywów tworzących kontur, by finalnie powrócić do punku startowego. Zadanie polega na optymalnym uszeregowaniu konturów (zadań) oraz równoczesnym doborze punktu startowego (kroku startowego). Rysunek 2.4 przedstawia prosty model oraz poszukiwane rozwiązanie opisanego zadania.

Tego typu problem można przedstawić formalnie, jako pewne uogólnienie problemu komiwojażera TSP. Istnieje wiele klas problemów pochodnych TSP. Poniżej przedstawiono porównanie klasycznego zadania TSP oraz jego uogólnionych form.



Rys. 2.5. Porównanie zadania TSP oraz jego uogólnień

Przedstawione na rysunku 2.5 kontury można nazwać regionami. Zgodnie z założeniami zadania komiwojażera poszukujemy najkrótszej drogi przechodzącej przez wszystkie regiony, przy czym każdy należy odwiedzić dokładnie jeden raz. W klasycznym podejściu do TSP każdy region musi posiadać zdefiniowany wcześniej punkt docelowy, np. miejsce gdzie komiwojażer będzie wystawiał swoje towary.

Istnieje pewne proste uogólnienie: GTSP [34] (ang. Generalized TSP), które zakłada, że w każdym z regionów (miast), istnieje wiele punktów docelowych, które są w pewnym sensie równoznaczne. Należy odwiedzić jeden z takich punktów, aby spełnić warunki zadania. Punkty muszą być wcześniej znane, np. lista dostępnych w mieście targowisk. Dyskretyzacja konturów ma jednak szereg wad. Zapewnienie wymaganej dokładności wiąże się z wprowadzeniem do zadania dużej liczby dodatkowych punktów, co bardzo utrudnia zadanie. Już samo zadanie TSP jest bardzo trudne, a liczba możliwych kombinacji rośnie jak *n!*. Dla 20 miast mamy 19! / 2, czyli około $6 \cdot 10^{16}$ możliwych cykli Hamiltona. Uzyskanie kompromisu pomiędzy dokładnością, a nakładem obliczeniowym prowadzi do wprowadzenia zaledwie kilku punktów w każdym z obszarów, a co za tym idzie z pogodzeniem się z dużymi błędami. Tak sformułowany problem nie nadaje się do rzeczywistych zastosowań i stanowi jedynie przedmiot rozważań teoretycznych.

Kolejnym sposobem zamodelowanie zadania jest CETSP (ang. Close Enough TSP) [25]. Tak sformułowane zadanie zakłada, że w każdym z regionów istnieje pewien punkt, który nie musi zostać osiągnięty dokładnie, lecz z pewnym przybliżeniem. Wystarczy zbliżyć się do wyznaczonych punktów na dystans określony pewnym promieniem. Taki model zupełnie nie nadaje się do rozwiązywania problemu grawerowania konturów.

Model TPP (ang. Touring a sequence of Polygons Problem) [11], czyli problem poszukiwania ścieżki prowadzącej przez sekwencje wielokątów jest odpowiedni do zadania grawerowania. Problem zakłada istnienie zdefiniowanej wcześniej sekwencji regionów. Celem jest wyznaczenie takich punktów w każdym z nich, aby poprowadzona przez nie ścieżka była jak najkrótsza. Odwołując się do rysunku 2.3, problem sprowadza się do poszukiwania optymalnego kroku startowego dla każdego z zadań danej sekwencji. Nie istnieje tu możliwość wyznaczenia optymalnej sekwencji, jedynie optymalizacja już istniejącej.

Najbardziej ogólnym zadaniem jest tzw. TSP z sąsiedztwem - TSNP (ang. TSP with Neighbourhood) [7]. Zadanie zakłada znalezienie optymalnej sekwencji odwiedzania regionów, przy czym punkty w poszczególnych regionach są dobierane tak, aby zminimalizować drogę.

Z przedstawionej dyskusji wariantów zadania TSP wynika, że występujący w programie *PCB CAM Processor* problem wyznaczania optymalnej ścieżki dla operacji grawerowania jest problemem klasy TSPN.

3. OPTYMALIZACJA WIERCENIA - METODY I WYNIKI

3.1. Spotykane metody rozwiązywania TSP

Można powiedzieć, że każdy proces optymalizacji polega na przeszukiwaniu pewnej wielowymiarowej, często niemożliwej do zobrazowania, przestrzeni rozwiązań dopuszczalnych w celu znalezienia najlepszego (względem pewnych kryteriów) rozwiązania. Zadanie klasy TSP należy no problemów kombinatorycznych. Przestrzeń rozwiązań dopuszczalnych jest skończona (problem nie jest ciągły, lecz dyskretny), niestety jej rozmiar jest olbrzymi. Z tego powodu, dla zadań o większej liczbie wierzchołków nie można sprawdzić wszystkich możliwych rozwiązań. Pozostają, zatem dwie możliwości: zastosowanie pewnej udoskonalonej techniki przeglądu przestrzeni rozwiązań dającej dokładne (optymalne) rozwiązanie lub zadowolenie się rozwiązaniem przybliżonym (leżącym dość blisko rozwiązania optymalnego).

3.1.1. Metody dokładne

Ta grupa metod pozwala uzyskać optymalne rozwiązanie, natomiast nie gwarantuje uzyskania wyniku w czasie wielomianowym. Czas obliczeń nie zależy jedynie od wielkości zadania, ale również od samej jego postaci. W skrajnych wypadkach, metody z tej grupy mogą wymagać przejrzenia niemal całej przestrzeni rozwiązań, co jest równoznaczne z brakiem możliwości obliczenia zadania. Na przestrzeni wielu lat badań nad problemem powstało wiele takich algorytmów jak: metoda podziału i ograniczeń czy metody typu *dziel i zwyciężaj*. Według Matai et al. [23], najlepszym obecnie narzędziem do rozwiązywani zadań TSP jest *Concorde* [6], który jest bezpłatnie dostępny na stronie internetowej wydziału matematyki uniwersytetu Waterloo [39].

3.1.2. Metody przybliżone

W wielu rzeczywistych zastosowaniach musimy się zadowolić rozwiązaniem przybliżonym, które można uzyskać w zadowalającym czasie obliczeniowym. Istnieje kilka możliwych podejść przybliżonych. Pierwszą, najprostszą grupą są algorytmy konstruujące ścieżkę według pewnego schematu:

- Metoda najbliższego sąsiedztwa - konstrukcja drogi od punktu startowego przez kolejne najbliższe nieodwiedzone dotąd punkty.

- Metoda zachłanna - budowa drogi zawierającej tylko najkrótsze krawędzie grafu.
- Metoda włączenie - konstrukcja drogi poprzez dołączanie kolejnych punktów pośrednich na trasie, według pewnej ustalonej zasady.
- Metoda Christofidesa - wyszukana heurystyka oparta m.in. na budowie tzw. minimalnego drzewa rozpinającego grafu.

Kolejną grupę metod stanowią heurystyki lokalnych poszukiwań lub inaczej metody *poprawy drogi*. Takie podejście polega na ulepszeniu istniejącego już rozwiązania uzyskanego np. za pomocą opisanych wcześniej metod konstrukcji rozwiązania. Najczęściej spotykane są tzw. metody *k-optymalne* (szczególnie z *k* równym 2 lub 3), polegające na usunięciu *k* krawędzi grafu i próbie połączenia ścieżki w inny (lepszy) sposób. Wadą tych metod jest znajdowanie lokalnych minimów zadania oraz ścisła zależność wyniku od postaci rozwiązania początkowego.

Rozwiązaniem problemu nieodporności, tj. skłonności do wpadania w lokalne minimum, jest zastosowanie algorytmów wykorzystujących losowość, takich jak metoda symulowanego wyżarzania (ang. Simulated Annealing - SA). Metody tego typu są wstanie przeszukiwać duże przestrzenie rozwiązań. Podejście SA bazuje na obserwacji procesu krystalizacji. W miarę obniżania się temperatury maleją drgania tworzącej się stopniowo sieci krystalicznej. W odniesieniu do poszukiwania rozwiązania, w miarę upływu czasu (kolejnych iteracji algorytmu), pole poszukiwań rozwiązania się zawęża. Maleje prawdopodobieństwo dużego oddalenie się od bieżącego rozwiązania. Innym podejściem poprawiającym odporność algorytmów *k-optymalnych* jest przeszukiwanie tabu (ang. Tabu Search - TS). Algorytm bazuje na poszukiwaniu *2-optymalnym*, lecz równocześnie przechowuje pewną listę złych rozwiązań, które są w pewnym sensie zabronione - tabu. Istnieją różne metody implementacji listy tabu [23].

Kolejną grupę stanowią algorytmy ewolucyjne, które naśladują występujący w przyrodzie proces doboru naturalnego. Algorytmy genetyczne (ang. Genetic Algorithm - GA), są również metodami poszukiwań losowych, przy czym kolejne rozwiązania powstają na bazie najlepszych wyznaczonych do tej pory rozwiązań. Można również zastosować pewne podejścia hybrydowe, łączące zdolność GA do szerokiej eksploracji przestrzeni rozwiązań z możliwościami skutecznych poszukiwań lokalnych. Szersze spojrzenie na GA oraz jego szczegółowe implementacje znajdują się w rozdziale 3.5.

Istnieje jeszcze inne podejścia do TSP bazujące na obserwacji przyrody. Algorytmy mrówkowe symulują kolonie mrówek poruszających się po wyznaczonych ścieżkach. Mrówki podczas swoich wędrówek rozkładają feromony, które są sygnałem dla innych mrówek, mówiącym o atrakcyjności danej drogi. Im więcej osobników wybierze daną ścieżkę tym wyższe jest stężenie feromonu. Tak zamodelowana kolonia mrówek, poruszając się po grafie znajduje w końcu atrakcyjną (krótką, bliską optymalnej) ścieżkę.

Przedstawione metody nie wyczerpują tematu optymalizacji TSP. W ciągu wielu dekad badań nad problemem, powstało wiele bardzo złożonych heurystyk. Badacze stosowali również metody: rojów cząstek, koloni pszczół [9] czy sztucznych sieci neuronowych.

3.1.3. Podsumowanie

W niniejszej pracy poszukiwane są stosunkowo szybkie i łatwe w implementacji metody nadające się do rzeczywistego problemu optymalizacji zadań wiercenia oraz grawerowania. Kontekst zadania TSP jest poruszany jako wstęp do poszukiwania metod rozwiązywania bardziej ogólnego zadania TSPN. Rozbudowane metody, wyspecjalizowane do zadania TSP niekoniecznie nadają się również do modelowania TSPN. W dalszych rozważaniach opisane są wybrane metody, które wydają się wykazywać pewien uniwersalizm w odniesieniu do zadań kombinatorycznych i mogą służyć do rozwiązywania obu stawianych w pracy problemów: TSP oraz TSPN.

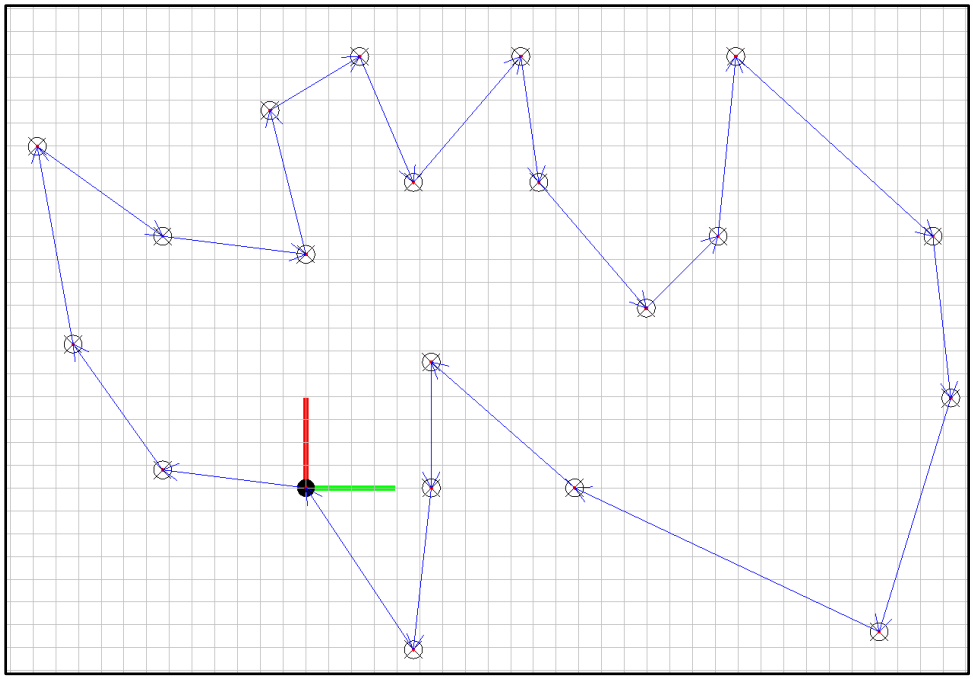
3.2. Metoda dokładna - podziału i ograniczeń

Metoda podziału i ograniczeń została zaimplementowana dla porównania z metodami przybliżonymi. Jest ona dość skomplikowana w implementacji oraz wymaga dużego nakładu obliczeń. Już po przekroczeniu $n = 20$ czas potrzebny na uzyskanie wyniku zaczyna znacząco rosnąć, co dyskwalifikuje ją do rzeczywistych zastosowań.

Podstawą zapisu zadania w tej metodzie jest tzw. macierz sąsiedztwa wyrażająca wagi łuków łączących wierzchołki (odległości pomiędzy punktami zadania). Każdy cykl Hamiltona zawiera dokładnie jeden łuk z każdego wiersza i jeden z każdej kolumny takiej macierzy. Istnieje możliwość obliczenia tzw. *dolnego ograniczenia* (ang. Lower Bound - LB) poprzez redukcję tej macierzy. Od każdego wiersza, a następnie od każdej kolumny należy odjąć pewną stałą tak, aby w każdym wierszu i w każdej kolumnie znajdowało się przynajmniej

jedno zero, a pozostałe elementy były nieujemne. Takie postępowanie redukuje długość wszystkich cykli o pewną stałą, natomiast względne relacje miedzy nimi oraz informacja o tym, który z nich jest optymalny pozostaje. Suma wszystkich odjętych od wierszy i kolumn liczb stanowi dolne ograniczenie długości jakiegokolwiek rozwiązania [35].

Algorytm polega na podziale zbioru rozwiązań na dwa podzbiory przy użyciu pewnej wyróżnionej krawędzi grafu: jeden zbiór zawiera rozwiązania zawierające tą krawędź, a drugi złożony jest z rozwiązań, które jej nie zawierają. Celem takiego postępowania jest redukcja rozmiaru zadania. Po każdej podziale obliczane jest dolne ograniczenie. Kolejnym krokiem jest ocena, który podzbiór posiada mniejsze LB i gdzie dalej będzie trwało poszukiwanie rozwiązania (podział zbioru). Jeśli mniejsze LB występuje w zbiorze zawierającym daną krawędź, to istotnie zmniejsza się rozmiar zadania, gdyż z macierzy sąsiedztwa zostaje wykreślony wiersz i kolumna, na których przecięciu leży rozważana krawędź. W przeciwnym wypadku, kiedy wybrany zostaje zbiór rozwiązań niezawierających rozważanej krawędzi, rozmiar zadania nie maleje, a krawędź zostaje zablokowana (w rozważaniach teoretycznych można postawić znak ∞ w miejsce tej krawędzi w macierzy sąsiedztwa, co oznacza, że krawędź nie istnieje). W takim przypadku ponownie musi zostać wybrana krawędź do podziału zbioru. Takie postępowanie polegające na wyborze krawędzi i próbie redukcji zadania wykonywane jest, aż do uzyskania całego cyklu Hamiltona. Kolejne podziały wskazują krawędzie, z których składa się droga optymalna.



Rys. 3.1. Rozwiązanie zadania o rozmiarze *n = 20* metodą podziału i ograniczeń

3.3. Proste heurystyki przybliżone

Metody przedstawione w tym rozdziale pozwalają bardzo szybko uzyskać rozwiązania przybliżone. Z uwagi na niską jakość tych rozwiązań należy je jednak traktować jako rozwiązania początkowe dla dalszych metod, których celem jest polepszenia istniejącego rozwiązania. Uzyskiwane wyniki są oczywiście znacznie lepsze od rozwiązań generowanych losowo. Kolejne metody silnie zależą od postaci rozwiązania początkowego, dlatego podawanie na wejście rozwiązania losowego może nie być właściwym podejściem.

3.3.1. Algorytm najbliższego sąsiada

Wyznaczanie rozwiązania metodą najbliższego sąsiada zostało już zaimplementowane w starszej wersji oprogramowania *PCB CAM Processor* [31]. Dzięki stosowaniu pewnej dodatkowej zasady nadającej priorytet kierunkowi wzdłuż osi *X*, algorytm nie błądzi w równomiernych siatkach, które stosunkowo często pojawiają się w projektach PCB. Rozszerzony na potrzeby tej pracy algorytm przedstawiono w postaci pseudokodu bazującego na składni języka *C++*. Listing 3.1 prezentuje ogólną zasadę metody NN.

Listing 3.1. Algorytm najbliższego sąsiada - NN

```
Input:  InPoints  - wej. tablica punktów
Output: OutPoints - wyj. tablica, wynik

InPoints[0] -> OutPoints //dodanie pierwszego punktu

while( InPoints.Count > 0 )
{
    BestI = 0
    LastP = OutPoints.Last
    D = Dist( InPoints[0], Last )

    for( i = 1; i < InPoints.Count; i++ )
    {
        CandP = InPoints[i]
        NextD = Dist( CandP, LastP )
        if( NextD < D ||
            Xpr && NexdD == D && CandP.Y == LastP.Y ||
            Ypr && NexdD == D && CandP.X == LastP.X )
        {
            D = NextD
            BestI = i
        }
    }
    InPoints[BestI] -> OutPoints //dodanie kolejnego punktu
}
```

Wejściem algorytmu jest tablica punktów *InPoints*, wyjściem tablica *OutPoints*, która jest początkowo pusta. W pierwszym kroku następuje przepisanie pierwszego punktu (o indeksie 0) z tablicy wejściowej do wyjściowej. Dalej wykonywana jest pętla *while* aż do wyczerpania się punktów w tablicy wejściowej, czyli do momentu, aż wszystkie elementy zostaną przeniesione do wyniku. W pętli występuje pomocnicza zmienna *BestI*, która określa indeks najlepszego znalezionego punktu, który zostanie dodany do wyniku. Zmienna *LastP* zawiera ostatnio dodany punkt, czyli aktualny koniec ścieżki, natomiast *D* określa odległość pomiędzy pierwszym z punktów tablicy *InPoints*, a ostatnim ścieżki *LastPoint*. Kolejnym krokiem jest wybór najbliższego względem *LastPoint* punktu z tablicy *InPoint* i przeniesienie go do wyniku. W tym celu następuje przegląd wszystkich dostępnych punktów. Zmienna *CandP* zawiera aktualnie sprawdzany punkt. Obliczany jest *NextD*, czyli dystans pomiędzy *CandP*, końcem ścieżki *LastP*. Jeśli nowy punkt znajduje się bliżej niż wcześniej wybrany, to znaczy, że $NextD < D$, należy zapamiętać indeks tego punktu oraz odległość do końca ścieżki. W kolejnych iteracjach należy powtarzać takie postępowanie, aby znaleźć punkt znajdujący się najbliżej. Po zakończaniu przeglądu (pętli *for*) następuje dodanie wybranego punktu do wyniku.

W powyższym opisie celowo pominięto opis rozbudowanej instrukcji warunkowej wewnątrz pętli for, aby nie zaciemniać ogólnej idei algorytmu. Jak wspomniano, poprzednia wersja oprogramowania *PCB CAM Processor* umożliwiała stosowanie priorytetu kierunku osi *X* w przypadku regularnych siatek, to znaczy, kiedy odległości w każdym z kierunków były takie same. W takim przypadku o tym, który punkt znajduje się najbliżej decyduje kolejność sprawdzania, czyli umiejscowienie w tablicy wejściowej. Na potrzeby tej pracy rozszerzono algorytm o możliwość sterowania wyborem priorytetu osi. Istnieje możliwość zastosowania priorytetu względem osi *X* (zmienna $Xpr = true$), względem osi *Y* (zmienna $Ypr = true$) lub wykonania bez priorytetu ustawiając obie zmienne na *false*. Wspomniana instrukcja warunkowa działa następująco. Jeśli sprawdzany punkt nie znajduje się bliżej od poprzednio wybranego, ale ustawiony jest priorytet *Xpr*, to jeśli odległość nowego punktu jest taka sama oraz posiada on taką samą współrzędną *Y* jak ostatni punkt ścieżki, to jest to lepszy punkt i jego indeks zostaje zapamiętany do kolejnej iteracji. Analogicznie wygląda interpretacja priorytetu osi *Y*. Wspomniane warunki równościowe są oczywiście użyte dla uproszczenia. W rzeczywistości program wykorzystuje skalę podobieństwa, która określa równość dwóch liczb, jako ich bezwzględną różnicę nieprzekraczającą pewnej ustalonej wartości. Problem został szczegółowo opisany w rozdziale 6 pracy [31].

Kolejną modyfikacją heurystyki NN zaproponowaną w tej pracy jest możliwość wielokrotnego uruchomienia konstrukcji ścieżki, wybierając za punkt startowy kolejno wszystkie możliwe punkty. Na koniec wybierane jest najkrótsze spośród zaproponowanych rozwiązań. Listing 3.2 zawiera pseudokod opisanej metody.

Listing 3.2. Algorytm najbliższego sąsiada - wielokrotna konstrukcja ścieżki

```
Input:  InitSol - wej. tablica punktów - rozwiązanie początkowe
Output: BestSol - wyj. tablica, wynik - najlepsze rozwiązanie

BestSol = InitSol                          // Inicjalizacja rozwiązania
for( i = 0; i < InitSol.Count; i++ )       // Pętla dla wszystkich punktów
{
    CandSol = NN( InitSol, InitSol[i] ) // Wykonanie alg. NN dla InitSol

    if( CandSol.Len < BestSol.Len )        // Jeśli CandSol lepsze
    {
        BestSol = CandSol                  // Przepisanie CandSol do BestSol
    }
}
```

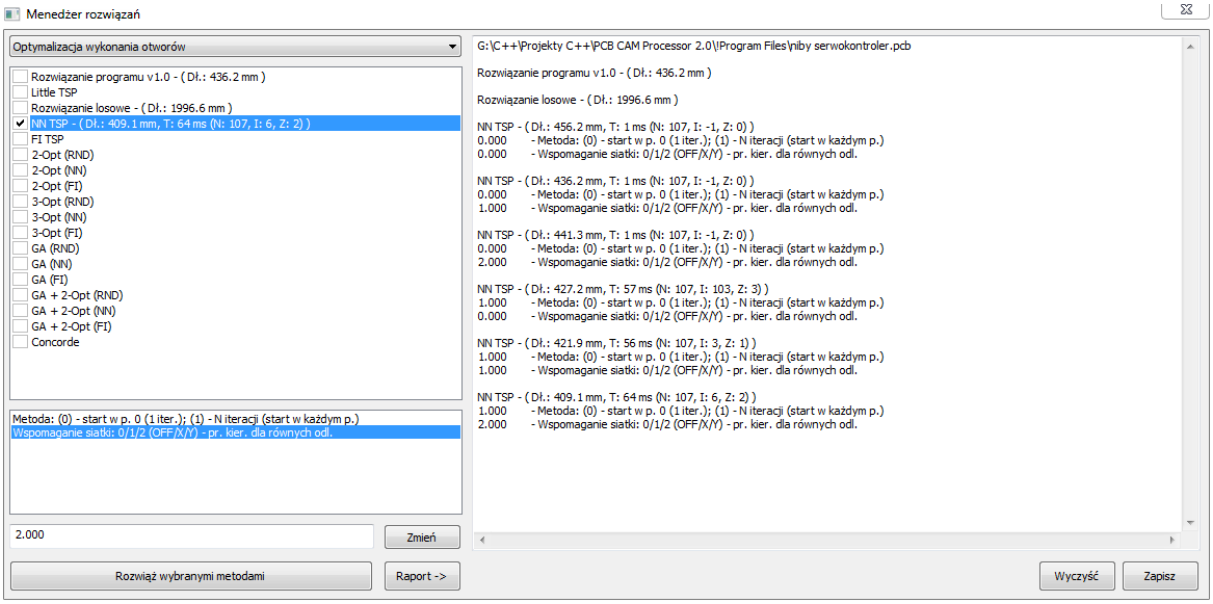
Wejściem algorytmu jest rozwiązanie początkowe *InitSol*, wyjściem najlepsze uzyskane rozwiązanie *BestSol*. Na początku następuje wstępne zainicjowanie rozwiązania *BestSol* przy użyciu *InitSol*. W kolejnym kroku wykonywana jest pętla *for* dla wszystkich punktów znajdujących się w *InitSol*. Wewnątrz pętli wykonywana jest procedura *NN*, która przyjmuje za argumenty tablicę punktów oraz punkt, który ma stanowić początek ścieżki. Procedura stanowi implementację algorytmu z listingu 3.1. Wynikiem jej działania jest rozwiązanie, które jest zapisywane do pomocniczej tablicy *CandSol*. Jeśli długość rozwiązania *CandSol* jest mniejsza od *BestSol,* to zostaje ono zapamiętane jako najlepsze. Takie postępowanie kontynuowane jest, aż do utworzenia i sprawdzenia wszystkich możliwych ścieżek. Efektem końcowym jest najlepsza z możliwych do uzyskania ścieżek metodą *NN*.

Wspomniane wcześniej warianty wykonania algorytmu zostały w programie zaimplementowane poprzez wprowadzenie *parametrów solvera*. Każda oprogramowana metoda (*solver*) posiada unikalny zestaw ustawień. Poniżej przedstawiono dostępne parametry solvera NN wraz dopuszczalnymi wartościami i opisem działania.

Metoda:	
0	Start w punkcie *zero* (zdefiniowany punkt startowy) - jedna iteracja
1	Start w każdym możliwym punkcie - *n* iteracji

Wspomaganie siatki:	
0	Wyłączone
1	Priorytet ruchu wzdłuż osi *X*
2	Priorytet ruchu wzdłuż osi *Y*

Wybór odpowiedniego *solvera* oraz sterowanie jego parametrami jest możliwe z poziomu menedżera rozwiązań. Okienko zawiera wszystkie dostępne metody rozwiązywania dla obu występujących w programie problemów. Poniżej przedstawiono wygląd okienka dla zadania optymalizacji wykonywania otworów.

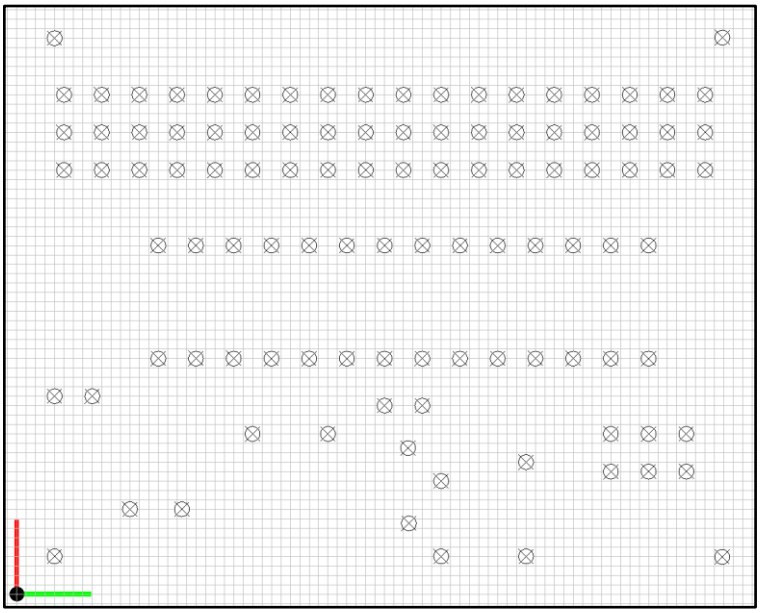


Rys. 3.2. Okno menedżera rozwiązań dla zadania optymalizacji wykonywania otworów

Okienko menedżera rozwiązań zawiera listę dostępnych metod rozwiązywania zadania, na której można wybrać pewne metody, następnie przy użyciu przycisku *Rozwiąż wybranymi metodami* obliczyć zadanie. Poniżej rozwiązań znajduje się lista dostępnych parametrów wskazanej metody. Do edycji wybranych parametrów służy pole edycyjne znajdujące się pod listą. Ostatnim elementem okna jest raport, który zawiera historyczne informację na temat używanych metod oraz ich rezultatów.

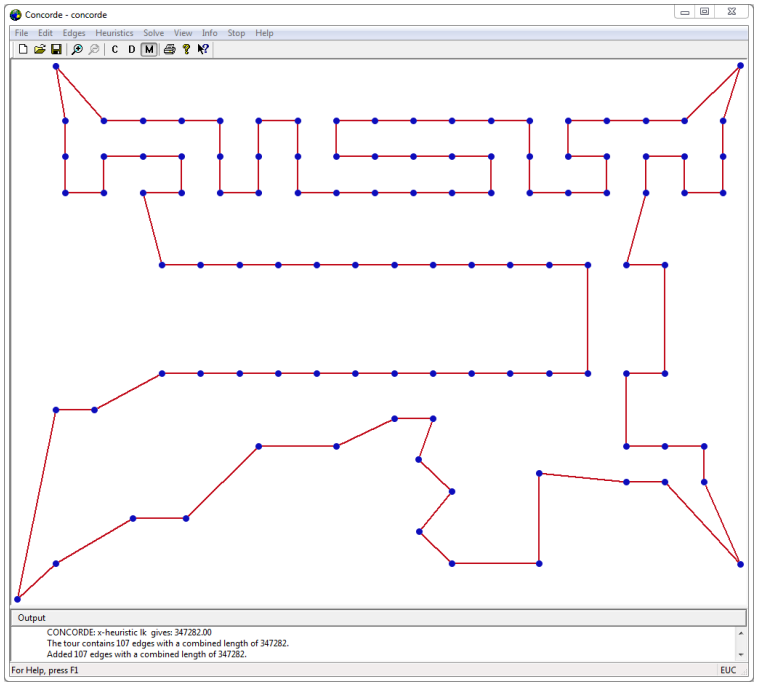
Poszukiwanie drogi metodą najbliższego sąsiada zależy od wyboru punktu startowego. Dla modeli zawierających równomierne siatki otworów istotne jest również wybór priorytetu kierunku ruchu. W dalszej części tego rozdziału przedstawiono wyniki obliczeń drogi dla modelu *serwokontrolera* (projekt autorski) oraz zadania z biblioteki przykładów TSPLIB, zawierającej zadania TSP o znanym rozwiązaniu optymalnym. Dzięki odniesieniu do znanych

instancji zadań można wyznaczyć skuteczność danej metody poprzez określenie błędu, jaki popełnia w danym zadaniu. Do porównania wykorzystano oprogramowanie *Concorde*, wspomniane w rozdziale 3.1.1, które jest obecnie najlepszym *solverem* TSP umożliwiającym wyznaczanie rozwiązań dokładnych [23].



Rys. 3.3. Układ otworów modelu serwokontrolera - projekt własny

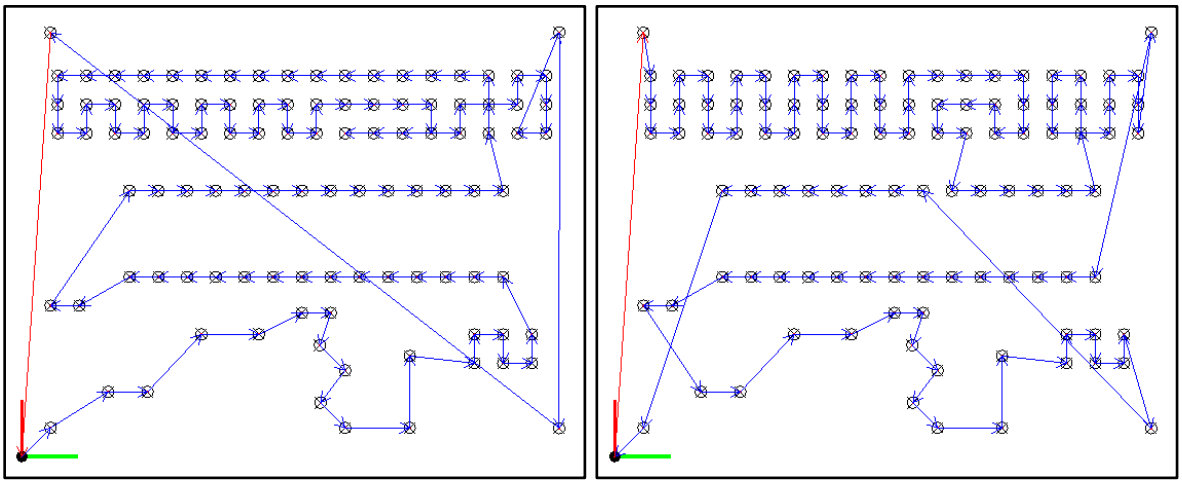
Na rysunku 3.3 przedstawiono pierwsze z zadań wykorzystane do oceny skuteczności opisanej metody. Model zawiera 106 otworów. Poniżej przedstawiono porównawcze rozwiązanie uzyskane w programie *Concorde*.



Rys. 3.4. Rozwiązanie zadania uzyskane w programie *Concorde*

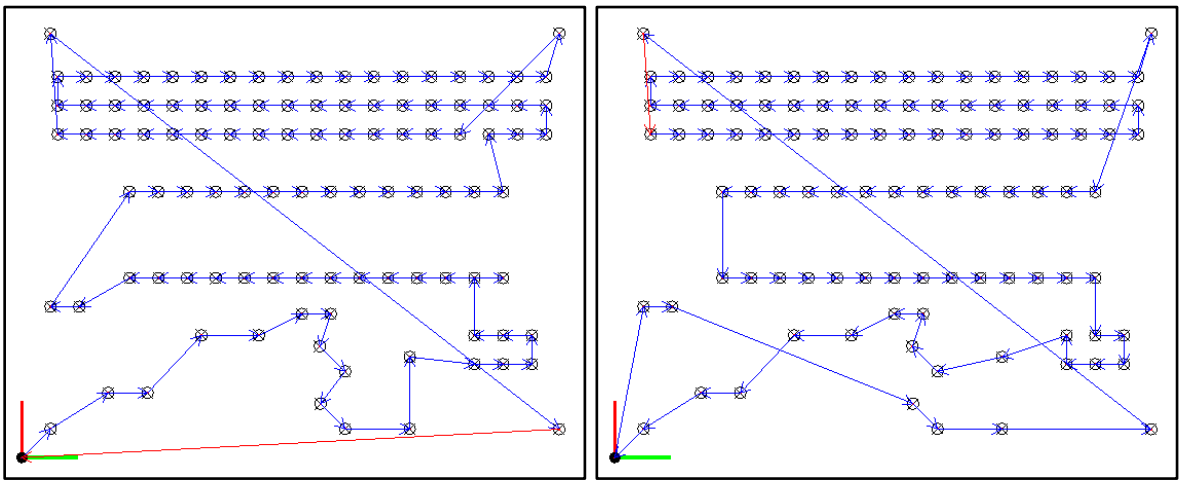
Program *Concorde* w czasie około 150 ms znajduje rozwiązanie optymalne zadania. Długość obliczonej ścieżki wynosi 347.3 mm.

Na kolejnych ilustracjach zostały pokazane rozwiązania zadania uzyskane metodą najbliższego sąsiedztwa przy różnych ustawieniach metody.



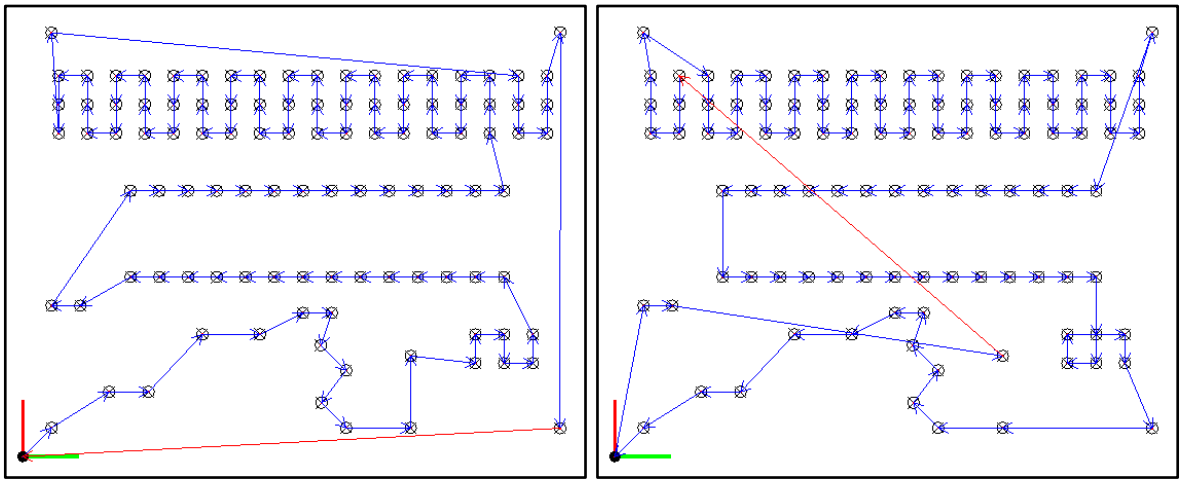
Rys. 3.5. Rozwiązania metodą NN, brak wspomagania siatki (z lewej 1 iteracja, z prawej *n* iteracji)

Rysunek 3.5 przedstawia wyniki obliczeń ścieżki uzyskane bez wykorzystania wspomagania siatki. Po lewej znajduje się wynik wykonania jednej iteracji, dla której punkt o współrzędnych (0, 0) został przyjęty jako początkowy. Prawa ilustracja prezentuje wynik wykonania 107 iteracji. Za każdym razem inny punkt został ustawiony jako punkt startowy, a następnie najlepszy wariant został zaprezentowany jako wynik końcowy. Czerwona strzałka wskazuje ostatnie przesunięcie na ścieżce. Jedna iteracja algorytmu zajęła poniżej 1 ms, a długość ścieżki wyniosła 456.2 mm, natomiast po wielu iteracjach uzyskano długość 427.2 mm w czasie 42 ms.



Rys. 3.6. Rozwiązania metodą NN, priorytet osi *X* (z lewej 1 iteracja, z prawej *n* iteracji)

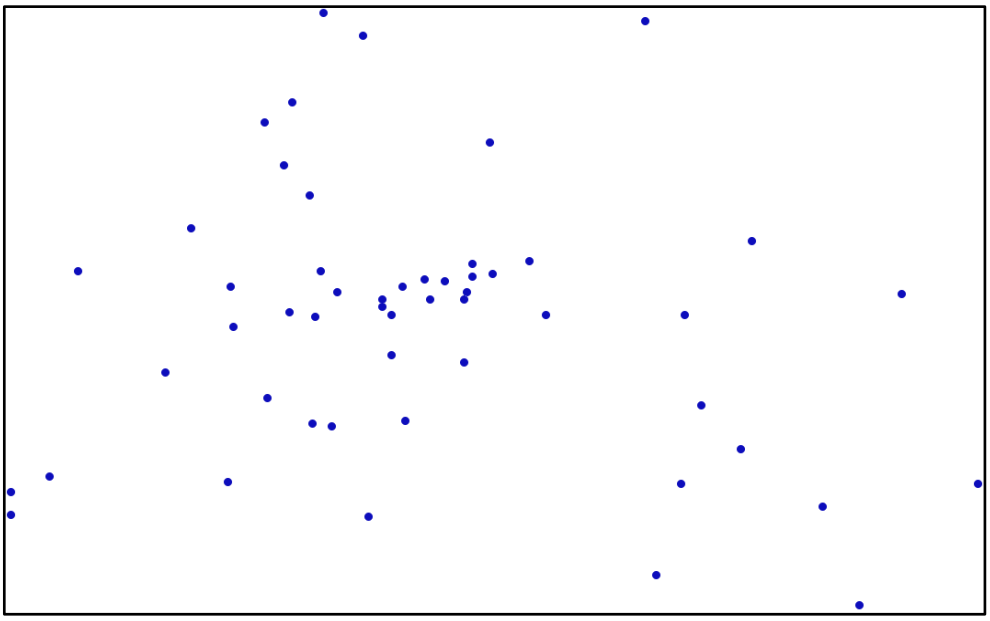
Na rysunku 3.6 przedstawiono wyniki metody NN z ustawionym priorytetem kierunku osi *X* dla siatek. Lewa ilustracja prezentuje wynik jednej iteracji algorytmu. Długość ścieżki wyniosła 436.2 mm w czasie poniżej 1 ms. Po wykonaniu wielu iteracji osiągnięto ścieżkę długości 421.9 mm w czasie 56 ms - prawa ilustracja.



Rys. 3.7. Rozwiązania metodą NN, priorytet osi *Y* (z lewej 1 iteracja, z prawej *n* iteracji)

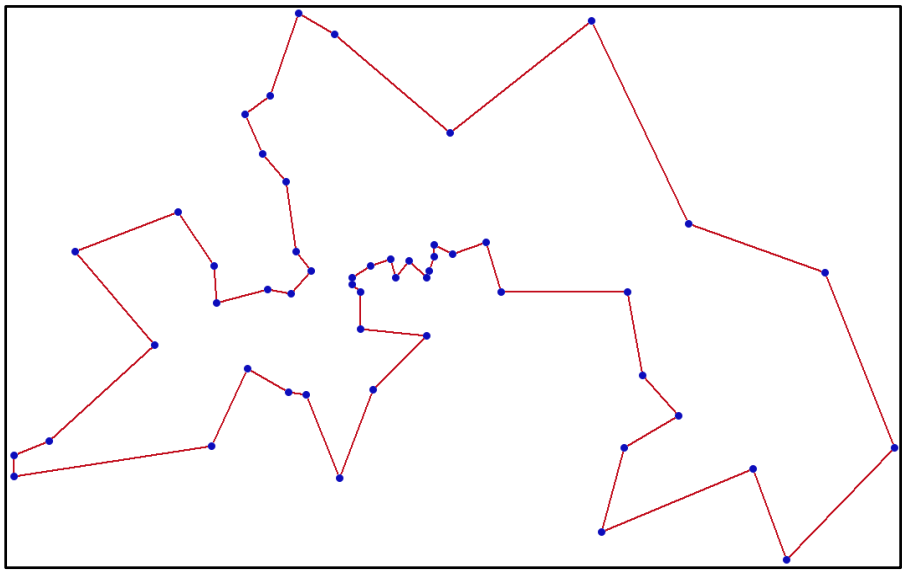
Przedstawione na rysunku 3.7 wyniki zostały uzyskane dla ustawienia priorytetu osi *Y* dla siatek. Jedna iteracja osiągnęła 441.3 mm długości ścieżki w czasie poniżej 1 ms, natomiast wiele iteracji algorytmu uzyskało ścieżkę długości 409.1 mm w czasie 73 ms.

Drugim modelem użytym do walidacji algorytmu było zadanie *Berlin52* pochodzące z biblioteki przykładów znajdującej się na stronie internetowej [40]. Zadanie zawiera 52 punkty, które są odniesieniem do konkretnych adresów w Berlinie.



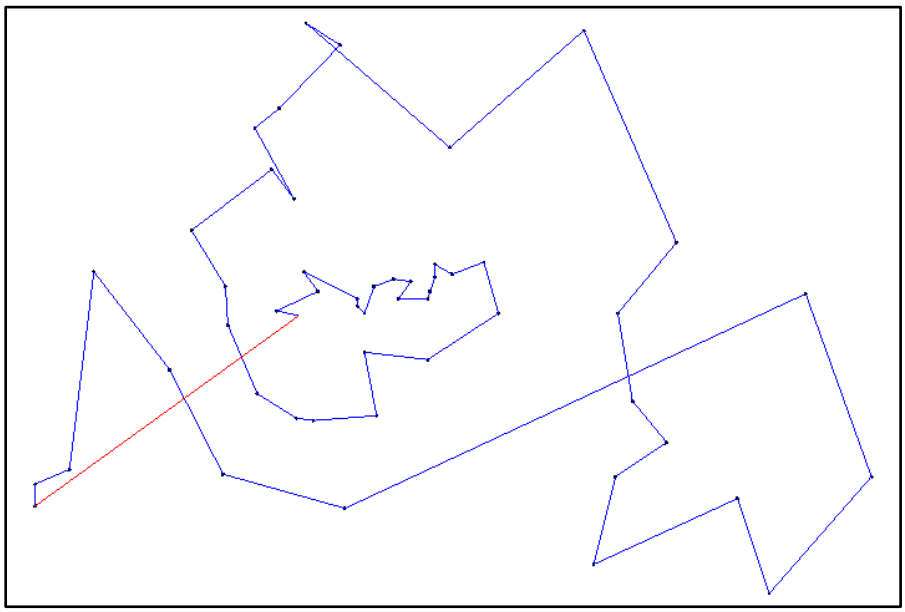
Rys. 3.8. Zadanie *Berlin52* - 52 lokalizacje w Berlinie

Na rysunku 3.8 zaprezentowano postać zadania *Berlin52*, które mimo stosunkowo niskiej liczby punktów stanowi spore wyzwanie dla algorytmów poszukujących najkrótszej ścieżki. Już na pierwszy rzut oka widać, że człowiekowi byłoby trudno wyznaczyć jakiekolwiek dobre (tzn. bliskie optymalnemu) rozwiązanie. Poniżej przestawiono optymalne rozwiązanie uzyskane w programie *Concorde*. Długość ścieżki wynosi 7544.4 m i została obliczona w czasie 80 ms.



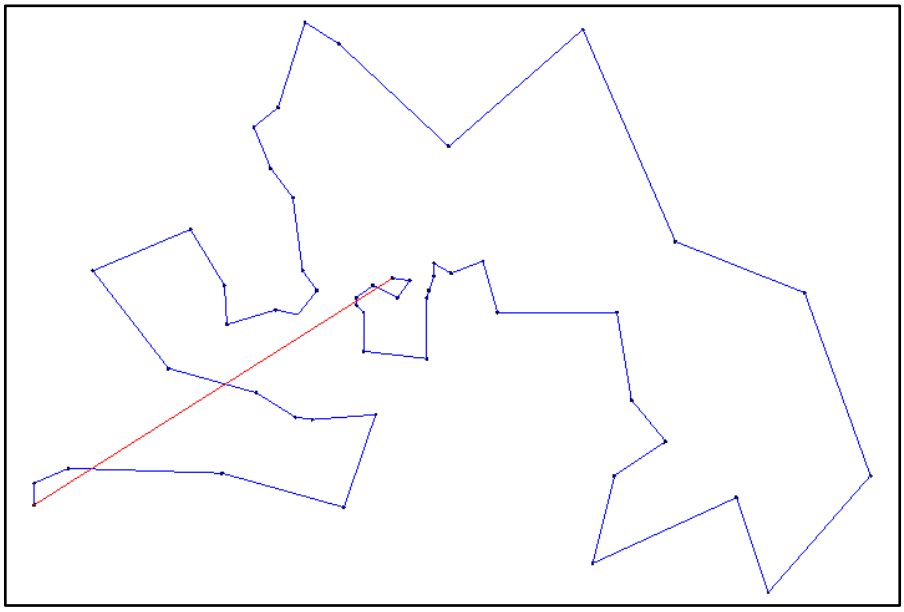
Rys. 3.9. Optymalne rozwiązanie zadania *Berlin52* uzyskanie w programie *Concorde*

Na kolejnych ilustracjach przedstawiono wyniki obliczeń zadania uzyskane metodą NN. Wspomaganie siatki zostało wyłączone, ponieważ nie ma ono zastosowania w tak nieregularnym układzie punktów.



Rys. 3.10. Rozwiązanie metodą NN - jedna iteracja

Rysunek 3.10 przedstawia rozwiązanie zadania Berlin52 uzyskane metodą NN przy jednej iteracji algorytmu. Jako punkt startowy został wybrany pierwszy punkt ze zbioru opisującego zadanie. Ścieżkę o długości 8980.9 m obliczono w czasie poniżej 1 ms.



Rys. 3.11. Rozwiązanie metodą NN - 52 iteracje

Zaprezentowane na rysunku 3.11 rozwiązanie zadania zostało uzyskane po 52 iteracjach algorytmu NN w ciągu 13 ms, a długość ścieżki wyniosła 8182.2 m.

W tabeli 3.1 zestawiono wyniki wszystkich wariantów metody NN dla obu obliczanych zadań. Obok długości uzyskanej ścieżki oraz czasu wykonania metody, podano również błąd względem rozwiązania optymalnego, wyrażony w procentach.

Tabela 3.1. Zestawienie wyników metody NN

Zadanie	Liczba iteracji	Długość ścieżki	Czas	Błąd
Płyta PCB	1	456.2 mm - brak priorytetu	< 1ms	31.4%
		436.2 mm - priorytet osi *X*	< 1ms	25.6%
		441.3 mm - priorytet osi *Y*	< 1ms	27.1%
	107	427.2 mm - brak priorytetu	42 ms	23.0%
		421.9 mm - priorytet osi *X*	56 ms	21.5%
		409.1 mm - priorytet osi *Y*	73 ms	17.8%
Berlin52	1	8980.9 m	< 1 ms	19.0%
	52	8182.2 m	13 ms	8.5%

Wnioski:

- Uzyskane wyniki wskazują, że metoda najbliższego sąsiedztwa nie powinna być brana pod uwagę jako samodzielna metoda obliczeniowa.
- Mimo zastosowania pewnych ulepszeń w postaci zwielokrotnionego uruchamiania oraz wsparcia dla wykonywania siatek, rozwiązania generowane przez tą metodą mają duże błędy.
- Zastosowanie priorytetu osi *Y* dla zadania płyty PCB, przy sprawdzeniu wszystkich możliwych punktów startowych daje najlepszy rezultat.
- Stosowany wcześniej na stałe, priorytet osi *X* [31] okazał się błędnym założeniem. Możliwość zmiany priorytetu może mieć korzystny wpływ na wynik.
- Stosując ulepszoną metodę NN udało się skrócić ścieżkę z 436.2 mm (poprzednia wersja *PCB CAM Processor*) do 409.1 mm dla zadania płyty PCB.
- Wykazano, że dla modeli zawierających siatki otworów, używanie priorytetu jest zasadne. Brak priorytetu zawsze pogarsza wynik.
- Rozwiązanie zadania *Berlin52* metodą wielokrotnego doboru punktu startowego daje znakomite rozwiązanie początkowe dla metod polepszających rozwiązanie. Duża część tego rozwiązania jest optymalna (porównanie rys. 3.9 oraz 3.11), a błąd wynosi zaledwie 8.5%.

3.3.2. Algorytm włączenia

Kolejną prostą heurystyką konstrukcji rozwiązania jest metoda włączenia. W tej metodzie wybierany jest pewien punkt startowy *s*. Następnie z pozostałych, nieodwiedzonych dotąd punktów, zgodnie z pewnym kryterium (omówionym dalej), wybierany jest kolejny punkt *p*. Tak powstała droga zawiera dwa wierzchołki, a kolejność ich odwiedzania jest następująca: *s, p, s*. W kolejnych krokach algorytm wyszukuje najlepszy z pośród dostępnych punktów i próbuje włączyć go do ścieżki w najlepszy możliwy sposób, tzn. by nowa ścieżka była jak najkrótsza. Nowy wierzchołek *k*, można włączyć do drogi *s, p, s* na dwa sposoby. Powstałe drogi mają postać: *s, k, p, s* lub *s, p, k, s*. Takie postępowanie jest kontynuowane, do momentu włączenia wszystkich punktów do ścieżki.

Istnieje wiele kryteriów wyboru punktów włączanych do ścieżki. Spotykane w literaturze są między innymi takie podejścia jak [35]:

- wybór losowego punktu - *swobodne włączenie*,
- wybór punktu leżącego najbliżej cyklu - *włączenie najbliższego wierzchołka*,
- porównanie kosztu włączenia wszystkich punktów we wszystkie możliwe miejsca i wybór tego o najniższym koszcie - *najtańsze włączenie*,
- wybór punktu leżącego najdalej od cyklu - *włączenie najdalszego wierzchołka*.

Wszechstronne badania prowadzone nad powyższymi metodami [35] wykazują, że najlepszym podejściem jest *włączenie najdalszego wierzchołka* (ang. Farthest Insertion - FI).

W oprogramowaniu *PCB CAM Processor* została zaimplementowana metoda najdalszego włączenia, która dodatkowo została rozszerzona o możliwość wielokrotnego uruchamiania dla różnych punktów startowych. Pseudokod metody FI został zaprezentowany na listingach 3.3 oraz 3.4.

Listing 3.3. Algorytm najdalszego włączenia - FI (metoda jednokrotna)

```
Input:  InitSol - wej. tablica punktów - rozwiązanie początkowe
Output: BestSol - wyj. tablica, wynik - najlepsze rozwiązanie

BestSol.AddFarthestPair( InitSol )      // Dodanie najdalszej pary

while( InitSol.Count > 0 )
{
    BestSol.AddFarthestPoint( InitSol ) // Dodanie najdalszego punktu
}
```

Listing 3.4. Algorytm najdalszego włączenia - FI (metoda wielokrotna)

```
Input:  InitSol - wej. tablica punktów - rozwiązanie początkowe
Output: BestSol - wyj. tablica, wynik - najlepsze rozwiązanie

InitSolCopy = InitSol // Zachowanie kopii rozwiązania początkowego
for ( i = 0; i < InitSol.Count; i++ )
{
    CandSol.AddPoint( InitSol[i] )
    while ( InitSol.Count > 0 )
    {
        CandSol.AddFarthestPoint( InitSol )
    }

    if ( CandSol.Len < BestSol.Len )
    {
        BestSol = CandSol
    }
    InitSol = InitSolCopy // Przywrócenie postaci rozw. pocz.
}
```

Algorytm FI został zaimplementowany w dwóch wersjach. Wejściem obu z nich jest tablica punktów *InitSol* - rozwiązanie początkowe stanowiące listę wszystkich punktów. Wyjściem algorytmów jest *BestSol*, czyli tablica zawierająca najlepsze rozwiązanie. Listing 3.3 prezentuje metodę jednokrotną, gdzie początkiem ścieżki jest para najdalej oddalonych punktów. Druga metoda przyjmuje za początek ścieżki kolejne punktu ze zbioru i generuje *n* rozwiązań próbnych. Pseudokod zaprezentowano na listingu 3.4.

Metoda jednokrotna rozpoczyna się od wywołania procedury *AddFarthestPair*, która dodaje do rozwiązania *BestSol* parę najodleglejszych punktów z *InitSol*. Następnie, aż do wyczerpania punktów z *InitSol*, wykonywane jest dodawanie do ścieżki najodleglejszego punktu z *InitSol*. Realizuje to procedura *AddFarthestPoint*.

Metoda wielokrotna działa w pętli *for*, dzięki której konstrukcja ścieżki uruchamiana jest *n* razy, od każdego punktu startowego. Ścieżka jest w tym przypadku konstruowana w tablicy *CandSol* i stanowi rozwiązanie próbne. W kolejnym kroku realizowane jest, podobnie jak dla metody jednokrotnej, dodawanie punktów do ścieżki procedurą *AddFarthestPoint*. Rozwiązanie próbne zostaje poddane ocenie i jeśli jest lepsze od *BestSol*, to zastępuje poprzednie rozwiązanie. Tablica *InitSolCopy* stanowi kopię rozwiązania początkowego. Z uwagi na to, że każda procedura budująca ścieżkę przenosi punkt z rozwiązania *InitSol* do *CandSol*, tablica *InitSol* zostaje opróżniona i musi zostać ponownie wypełniona przed wykonaniem kolejnej iteracji algorytmu.

Pseudokody procedur i funkcji pomocniczych algorytmu zostały przedstawione na listingach od 3.5 do 3.8.

Listing 3.5. Procedura *AddFarthestPair*

```
BestI = 0, BestJ = 0, MaxD = 0.0

for ( i = 0; i < InitSol.Count -1; i++ )
{
    for ( j = i+1; j < InitSol.Count; j++ )
    {
        D = Dist( InitSol[i], InitSol[j] )
        if ( D > MaxD )
        {
            MaxD = D
            BestI = i
            BestJ = j
        }
    }
}
InitSol[BestI] -> BestSol
InitSol[BestJ] -> BestSol
```

Procedura *AddFarthestPair* składa się z dwóch zagnieżdżonych pętli *for*, które generują wszystkie możliwe pary zmiennych *i* oraz *j*, będących indeksami tablicy punktów rozwiązania *InitSol*. Wewnątrz pętli następuje obliczenie odległości pomiędzy każdą parą punktów. Jeśli ta odległość jest większa od dotychczas znalezionej, następuje zapisanie tej wielkości jako *MaxD*, czyli największego dystansu pomiędzy każdą parą punktów. Równocześnie odbywa się zapamiętanie aktualnych indeksów punktów stanowiących parę. Zapisywane są zmienne *BestI* oraz *BestJ*. Po wykonaniu wszystkich iteracji pętli, to znaczy po sprawdzeniu wszystkich możliwych par, następuje przepisanie ustalonych punktów z tablicy *InitSol* do tablicy rozwiązania *BestSol*.

Listing 3.6. Procedura *AddFarthestPoint*

```
BestI = 0, MaxD = 0.0

for ( i = 0; i < InitSol.Count; i++ )
{
    D = BestSol.MaxDist( InitSol[i] )
    if ( D > MaxD )
    {
        MaxD = D
        BestI = i
    }
}
BestSol.BestInsert( InitSol[BestI] )
```

Procedura *AddFarthestPoint* służy dodawaniu do rozwiązania najdalszego punktu ze zbioru jeszcze nieodwiedzonych, przy czym punkt dodawany jest w najlepsze z możliwych w ścieżce miejsc, zgodnie z opisaną wcześniej zasadą. Wewnątrz pętli *for*, służącej do przeglądu wszystkich punktów *InitSol*, następuje obliczanie maksymalnej odległości punktu *InitSol[i]* od aktualnej ścieżki *BestSol* funkcją *MaxDist*. Jeśli obliczona wartość jest większa od dotychczas znalezionej, to zostaje zapamiętana w zmiennej *MaxD*. Zapisywany jest również indeks *BestI* znalezionego punktu. Dodanie wybranego punktu do ścieżki realizuje procedura *BestInsert* opisana dalej.

Listing 3.7. Funkcja *MaxDist*

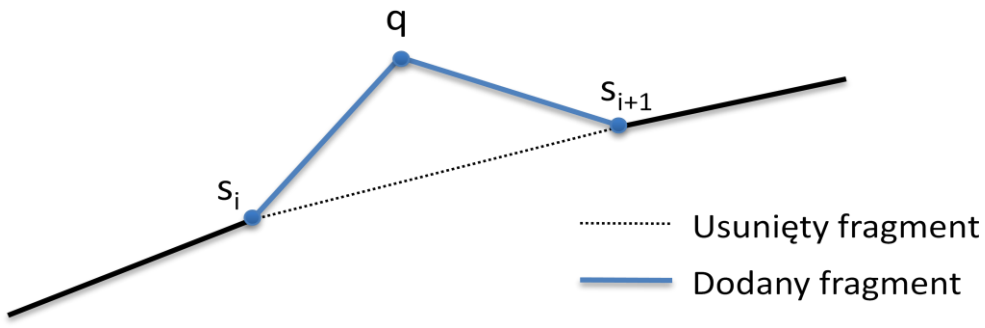
```
MaxD = 0.0

for ( i = 0; i < BestSol.Count; i++ )
{
    D = Dist( BestSol[i], Point )
    if ( D > MaxD )
    {
        MaxD = D
    }
}
return MaxD
```

Przedstawiona na listingu 3.7 funkcja *MaxDist* przyjmuje za argument punkt *Point*. Funkcja oblicza maksymalną odległość pomiędzy *Point*, a punktami tablicy *BestSol*. W pętli *for* przeglądane są wszystkie punkty tablicy *BestSol*, obliczana jest odległość od *Point* i jeśli jest ona większa niż dotychczas znaleziona, to zostaje zapisana w zmiennej *MaxD*. Po sprawdzeniu wszystkich punktów funkcja zwraca zmienną *MaxD*.

Listing 3.8. Procedura *BestInsert*

```
Input:  Point - dodawany punkt

MinCost = MaxDbl
BestI = 0, NextI, n = BestSol.Count

for ( i = 0; i < n; i++ )
{
    NextI = ( i == n-1 ) ? 0 : i+1

    InsertCost = Dist( BestSol[i], Point ) +
                 Dist( Point      ,BestSol[NextI] ) -
                 Dist( BestSol[i], BestSol[NextI] )

    if ( InsertCost < MinCost )
    {
        MinCost = InsertCost
        BestI = i
    }
}
BestSol.Insert( Point, BestI ) // Włączenie punktu do ścieżki po BestI
```

Procedura *BestInsert* stanowi najważniejszą cześć algorytmu FI. Wejściem procedury jest wybrany wcześniej punkt, który ma zostać włączony do cyklu w najlepszym możliwym miejscu. Poszukiwane jest miejsce w cyklu, gdzie koszt włączenia *MinCost* jest najmniejszy. Do tego celu służy pętla *for*. Miejsce włączenia oznaczone zmienną *i* stanowi tak naprawdę dwa kolejne punkty *i* oraz *i+1* między którymi ma się znaleźć nowy punkt.

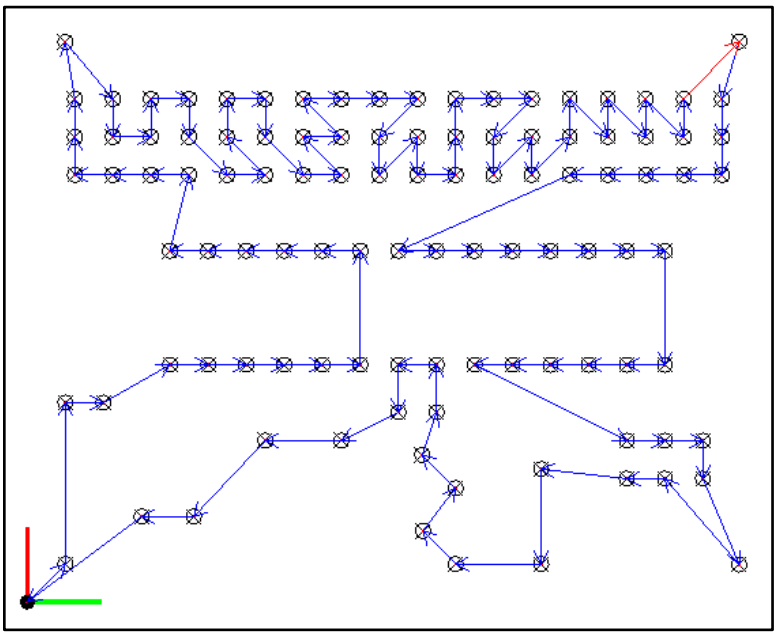


Rys. 3.12. Dodanie punktu nowego punktu do cyklu

Z uwagi na fakt, że droga jest cyklem, a jej interpretacja stanowi tablicę punktów o indeksach od *0* do *n-1*, kolejnym indeksem po *n-1* jest *0*. Obliczenie następnego indeksu realizuje

wyrażenie warunkowe, które przypisuje zmiennej *NextI* wartość *0*, jeśli *i == n-1*, a w przeciwnym wypadku wartość *i+1*. W każdej iteracji pętli *for* obliczany jest koszt włączenia (*InsertCost*) nowego punktu pomiędzy punkty o indeksach *i* oraz *NextI*. Jak pokazano na rysunku 3.12, włączenie punktu *q* pomiędzy s_i oraz s_{i+1} wiąże się z usunięciem ze ścieżki fragmentu (s_i, s_{i+1}) oraz dodaniem do niej fragmentów (s_i, *q*) oraz (*q*, s_{i+1}). Kosz włączenia można określić jako przyrost długości drogi, czyli różnicę między sumą długości nowych fragmentów, a długością usuniętego fragmentu. Po obliczeniu *InsertCost* następuje porównanie go z najmniejszym dotąd znalezionym kosztem *MinCost* i jeśli wynik jest korzystniejszy, to następuje zapamiętanie *InsertCost* w *MinCost* oraz zapisanie bieżącej lokalizacji w postaci zmiennej *BestI*. Po sprawdzeniu wszystkich możliwych miejsc w cyklu, następuje dodanie punktu *Point* do cyklu *BestSol* w miejsce o indeksie kolejnym do BestI.

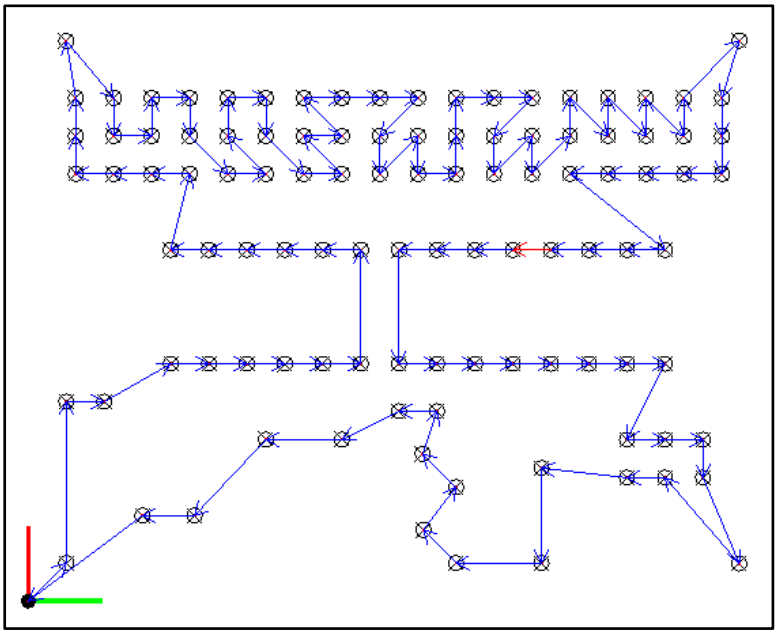
Podobnie jak dla metody NN walidacja heurystyki FI została przeprowadzana na dwóch modelach: płyty PCB - rys. 3.3 oraz zadania *Berlin52* - rys. 3.8. Poniżej przedstawiono wyniki obliczeń metody FI.



Rys. 3.13. Rozwiązanie metodą FI - jedna iteracja

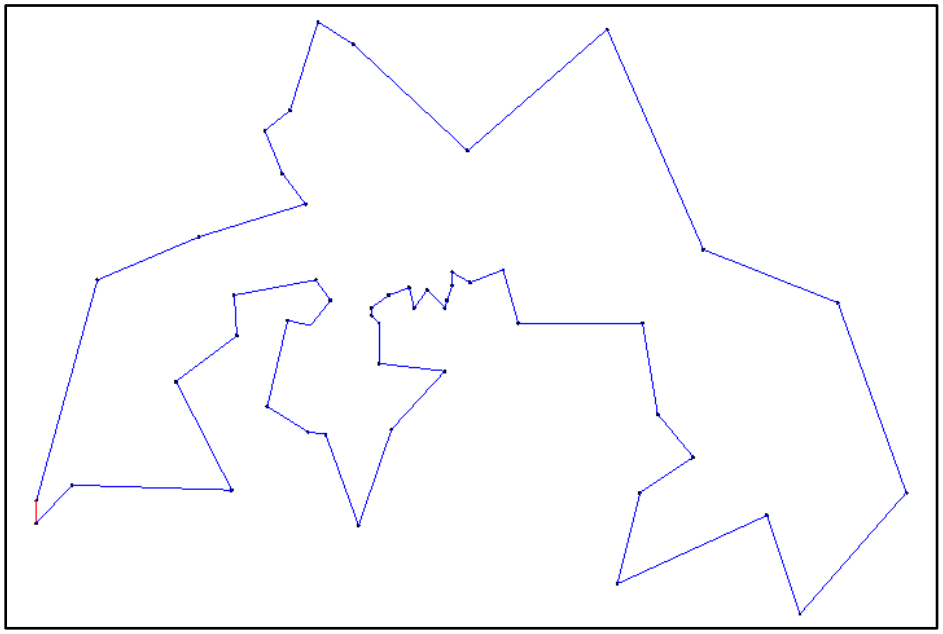
Rysunek 3.13 przedstawia wyniki obliczeń ścieżki dla zadania płyty PCB, uzyskane metodą najdalszego włączenia, przy wykonaniu jednej iteracji algorytmu (patrz listing 3.3). Drogę początkową stanowi tu para najodleglejszych punktów. Długość obliczonej ścieżki wynosi 376.5 mm i została uzyskana w czasie 25 ms.

Wyniki obliczeń tego samego zadania metodą FI wielokrotną, uruchamianą 107 razy dla każdego możliwego punktu startowego (patrz listing 3.4) prezentuje rysunek 3.14. Długość ścieżki wynosi 365.2 mm i została uzyskana w czasie 724 ms.



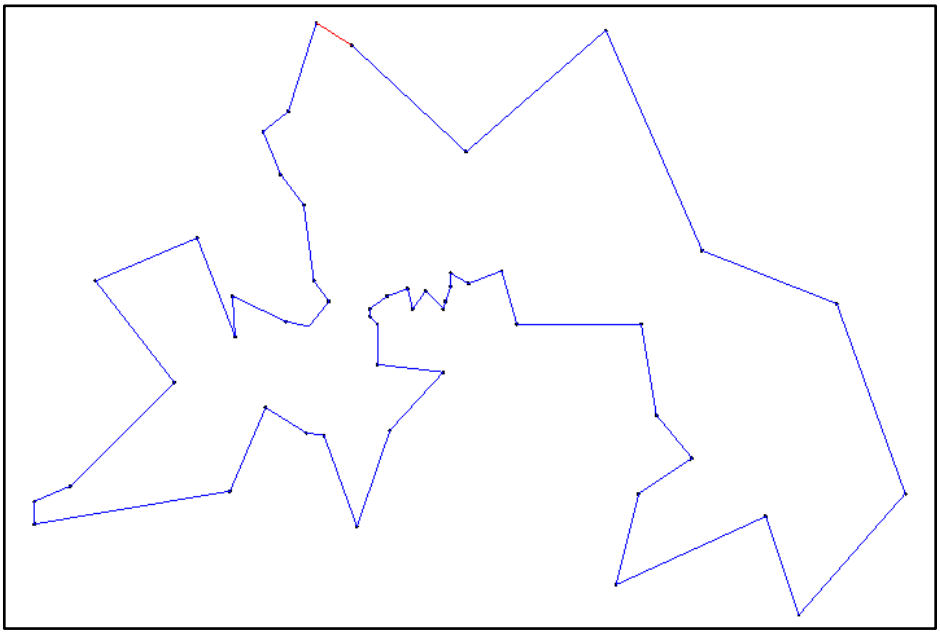
Rys. 3.14. Rozwiązanie metodą FI - wiele iteracji

Kolejne zadanie wzięte do wyznaczenia skuteczności prezentowanych metod to *Berlin52*. Wyniki obliczeń tego zadania metodą FI jednokrotną przedstawiono na rysunku 3.15. Wyznaczona ścieżka ma długość 7783.0 m i została osiągnięta w czasie 4 ms.



Rys. 3.15. Rozwiązanie zadanie *Berlin52* metodą FI - jedna iteracja

Rysunek 3.16 przedstawia wyniki obliczeń tego samego zadania z użyciem wielokrotnej metody FI. Po wykonaniu 52 iteracji, w czasie 77 ms, algorytm znalazł rozwiązanie o długości 7630.6 m.



Rys. 3.16. Rozwiązanie zadanie *Berlin52* metodą FI - wiele iteracji

W tabeli 3.2 zestawiono wyniki obu wariantów metody FI dla obu obliczanych zadań. Obok długości uzyskanej ścieżki oraz czasu wykonania metody, podano również błąd względem rozwiązania optymalnego, wyrażony w procentach.

Tabela 3.2. Zestawienie wyników metody FI

Zadanie	Liczba iteracji	Długość ścieżki	Czas	Błąd
Płyta PCB	1	376.5 mm	25 ms	8.41%
	107	365.2 mm	724 ms	5.15%
Berlin52	1	7783.0 m	4 ms	3.16%
	52	7630.6 m	77 ms	1.14%

Wnioski:

- Metoda najdalszego włączenia osiąga znacznie lepsze rezultaty niż metoda NN. W szczególności widoczny jest brak przecięć ścieżki, który występuje w każdym wyniku generowanym przez NN.
- Wielokrotne uruchomienie FI dale lepsze wyniki niż pojedyncza iteracja.

- Dla dużych zadań należy wziąć pod rozwagę opłacalność metody wielokrotnej, gdyż wzrost czasu obliczeniowego może być nie adekwatny do polepszenia rezultatu. Dla zadania płyty PCB uzyskano wynik, którego błąd względem rozwiązania optymalnego jest niższy o 3.26 punktu procentowego, a czas obliczeniowy wydłużył się niemal 30 krotnie.
- Dla zadania Berlin52 wynik uzyskany metodą wielokrotną jest bardzo dobry. Błąd względem rozwiązania optymalnego wynosi zaledwie 1.14%.
- Z uwagi na szybkość obliczeń oraz na brak przecięć ścieżki w wyniku metoda może być stosowana jako samodzielna lub służyć generowaniu rozwiązań początkowych dla metod polepszających.

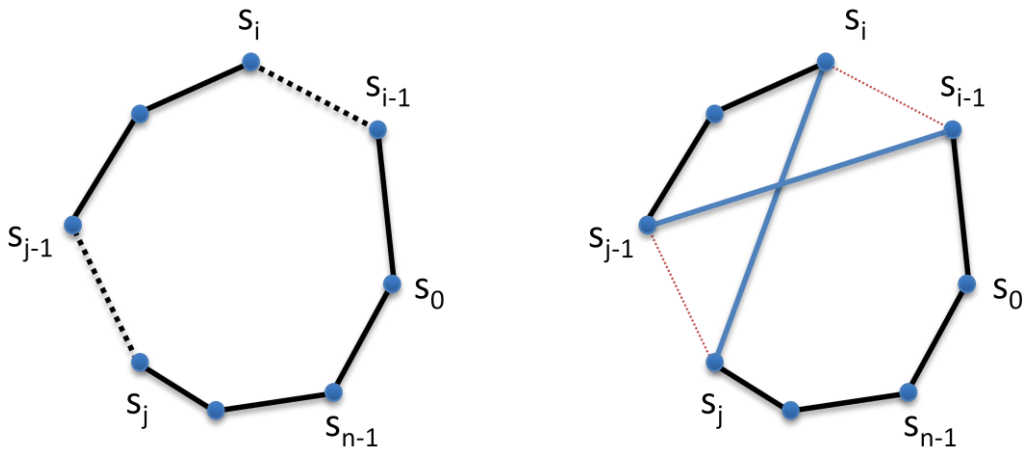
3.4. Metody poszukiwań lokalnych

Metody klasy *k-optymalnej* należą do najczęściej stosowanych przy rozwiązywaniu zadań optymalizacji kombinatorycznej, a w szczególności zadań TSP i podobnych. Są to metody szukające lepszego rozwiązania bazującego na już istniejącym. Algorytm zakłada usunięcie k krawędzi grafu i połączenie wierzchołków na nowo przy użyciu k innych krawędzi. Jeśli nowe rozwiązanie jest lepsze od poprzedniego to zostaje zapamiętane, a proces poszukiwania uruchamiany jest od początku. Algorytm kończy się, kiedy nie da się już poprawić rozwiązania. Im wyższy indeks k tym więcej możliwości rekonfiguracji grafu, co prowadzi do lepszych wyników, ale zwiększa nakład obliczeniowy. Badania wykazują, że powyżej $k = 3$, wzrost nakładu obliczeniowego jest nieadekwatny wzrostu dokładności rozwiązania [35]. Z tego powodu najczęściej stosuje się metody o k równym 2 lub 3. Istnieją natomiast pewne wyszukane heurystyki takie jak Lin-Kernighan, które wykorzystują poszukiwania o stopniu $k = 4$. Heurystyka LK podczas rozwiązywania stosuje zamiennie algorytmy stopnia 2, 3 lub 4 zależnie od warunków zadania. Metoda doboru odpowiedniej wartości k jest jednak dość złożona.

3.4.1. Algorytm *2-opt*

Metoda poszukiwania lepszego rozwiązania poprzez usuwanie dwóch krawędzi i rekonfigurację cyklu Hamiltona jest najprostsza do implementacji z pośród wszystkich

metod *k-optymalnych*, ponieważ istnieje jedynie jeden możliwy przypadek konstrukcji nowej drogi. Rysunek 3.17 prezentuje postać możliwej zamiany krawędzi cyklu.



Rys. 3.17. Rekonfiguracja dwóch krawędzi cyklu Hamiltona

Pseudokod metody poszukiwań *2-opt* zaprezentowano na listingu 3.9. Zaproponowana implementacja wykonuje przegląd wszystkich możliwych par krawędzi, które mogą zostać poddane zamianie.

Listing 3.9. Algorytm poszukiwania *2-opt*

```
Input/Output: BestSol - rozwiązanie początkowe podlegające poprawie

n = BestSol.Count

do // Główna pętla, wykonywana dopóki udaje się poprawić wynik
{
    Imroved = false
    for ( i = 0; i < n - 2 && !Imroved; i++ )
    {
        PrevI = ( i == 0 ) ? n-1 : i-1;
        for ( j = i + 2; j < n && !Imroved; j++ )
        {
            Imroved = NewTour( i, PrevI, j )
        }
    }
}
while( Imroved )
```

Wejściem (i zarazem wyjściem) algorytmu *2-optymalnego* jest rozwiązanie uzyskane wybraną metodą konstrukcji ścieżki, taką jak opisane wcześniej: NN czy FI lub rozwiązanie losowe. Podczas pracy algorytm usiłuje poprawić uzyskane wcześniej rozwiązanie. Program zawiera główną pętlę *do ... while*, która wykonywana jest dopóki znajdowane jest lepsze rozwiązanie. Wewnątrz tej pętli znajdują się dwie zagnieżdżone pętle *for*, które odpowiadają za przegląd wszystkich możliwych par krawędzi, które mogą zostać zastąpione. Z uwagi na

zapis cyklu w postaci tablicy o indeksach od *0* do *n-1* konieczne jest wprowadzenie zmiennej *PrevI* stanowiącej wartość indeksu poprzedzającego *i*. Wyrażenie warunkowe nadaje tej zmiennej wartość *n-1* w przypadku, gdy *i == 0*, a w przeciwnym wypadku ustala ją na *i-1*. Wewnętrzna pętla *for* startuje z indeksem *j = i+2* aby zachować minimalny odstęp pomiędzy punktami *i* oraz *j*. Analizując rysunek 3.17 można zauważyć, że przyjęcie indeksu *j = i+1* nie ma sensu, ponieważ wynikowa droga będzie taka sama. Za konstrukcję nowej ścieżki odpowiada procedura *NewTour*, która operuje na rozwiązaniu *BestSol* oraz przyjmuje za argumenty indeksy *i*, *i-1* oraz *j*. Wynikiem jej pracy jest odpowiedź na pytanie, czy ścieżka powstała przez usunięcie z *BestSol* krawędzi wyznaczonych przez wskazane indeksy jest lepsza od dotychczasowej. Jeśli tak jest, procedura zastępuje *BestSol* poprawioną wersją. Odpowiedź *NewTour* zostaje zapisana w zmiennej *Imroved*, która służy do sterowania pętlami. Pętle *for* działają do momentu znalezienia pierwszego poprawionego rozwiązania, natomiast główna pętla algorytmu kończy się, kiedy pętle *for* wykonają się do końca, nie znajdując lepszego rozwiązania. Listing 3.10 zawiera pseudokod procedury *NewTour*.

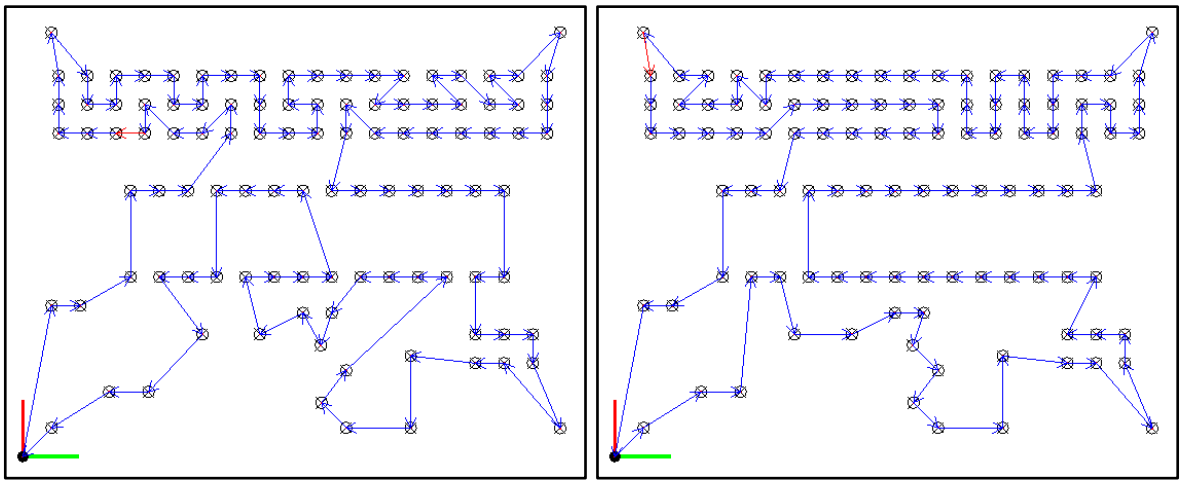
Listing 3.10. Procedura *NewTour*

```
Input:  i, PrevI, j - indeksy podziału cyklu
Output: czy poprawiono rozwiązanie BestSol

n = BestSol.Count
CandLen = BestSol.Len

// Usunięte krawędzie
CandLen -= Dist( BestSol[PrevI], BestSol[i] )
CandLen -= Dist( BestSol[j-1],   BestSol[j] )
// Dodane krawędzie
CandLen += Dist( BestSol[PrevI], BestSol[j-1] )
CandLen += Dist( BestSol[i],     BestSol[j]   )

if ( CandLen < BestSol.Len ) // Ocena rozwiązania
{
    // Kompozycja rozwiązania
    for ( h = 0;   h < i;     h++ ) TempSol.AddPoint( BestSol[h] )
    for ( h = j-1; h > PrevI; h-- ) TempSol.AddPoint( BestSol[h] )
    for ( h = j;   h < n;     h++ ) TempSol.AddPoint( BestSol[h] )

    BestSol = TempSol
    return true
}
return false
```

Wejściem procedury *NewTour* są dwa indeksy wyznaczające punktu podziału cyklu oraz pomocniczy indeks *PrevI* wprowadzony, aby nie trzeba było ustalać jego wartości wewnątrz procedury. Zmienna *CandLen* służy do wyznaczenia długości ścieżki rozwiązania próbnego. Na początku przyjmuje wartość długości rozwiązania *BestSol*. Zgodnie

z rysunkiem 3.17, długość nowej ścieżki może zostać obliczona poprzez odjęcie długości usuwanych krawędzi: (s_{i-1}, s_i) oraz (s_{j-1}, s_j), a następnie dodanie długości nowych krawędzi: (s_{i-1}, s_{j-1}) oraz (s_i, s_j). Po obliczeniu *CandLen* następuje ocena hipotetycznego rozwiązania próbnego i jeśli jest ono lepsze, to zostaje skonstruowane w trzech kolejnych pętlach *for*. Pętle przepisują punkty z *BestSol* do tymczasowej tablicy *TempSol* kolejno: od indeksu *0* do *i-1*, od *j-1* do *i* (odwrócona kolejność) oraz od *j* do *n-1*. Kolejność punktów w ścieżce wynika z wprowadzonych oznaczeń - rys. 3.17. Nowe rozwiązanie zastępuje *BestSol*, a *NewTour* zwraca *true*. Jeśli nowe rozwiązanie zostanie ocenione negatywnie, nie następuje konstrukcja rozwiązania *TempSol*, a funkcja zwraca *false*.

Na kolejnych ilustracjach zostały zaprezentowane wyniki walidacji metody poszukiwań lokalnych *2-opt* dla różnych wariantów rozwiązań początkowych. Zadania wykorzystane do walidacji są takie same jak dla poprzednich metod.

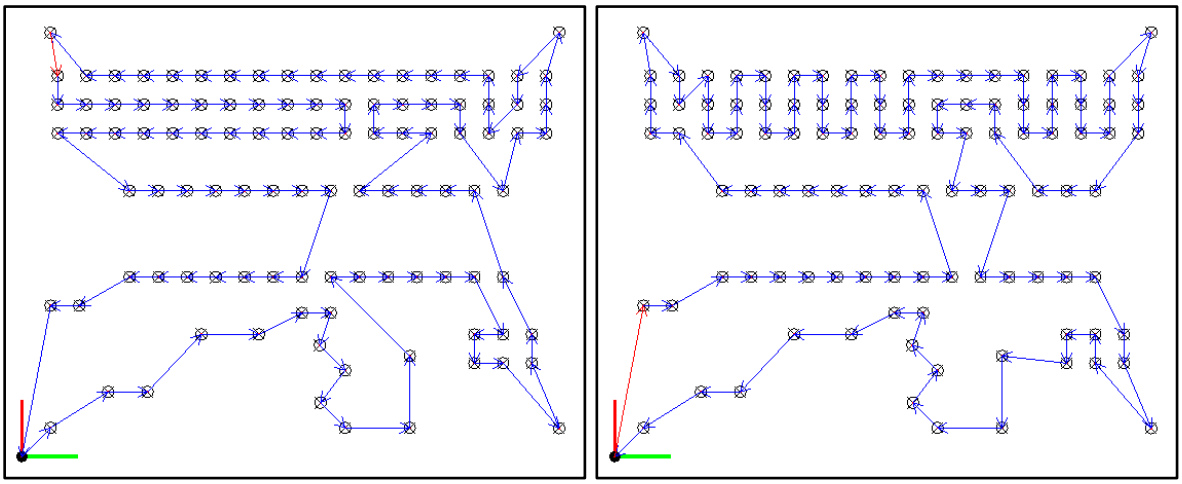


Rys. 3.18. Przykładowe rozwiązania uzyskane metodą *2-opt* z losowym rozwiązaniem początkowym

Pierwszą proponowaną postacią rozwiązania początkowego dla metody *2-opt* jest kolejność wygenerowana losowo. Zadanie płyty PCB zostało policzone 10 razy, następnie wyciągnięto średnią z uzyskanych wyników. Rysunek 3.18 przedstawia dwa przykładowe rozwiązania: z lewej ścieżka o długości 381.5 mm obliczona w 130 ms, z prawej ścieżka długości 357.0 mm uzyskana w 155 ms. Szczegółowe wyniki próby podano w zestawieniu wyników w tabeli 3.3.

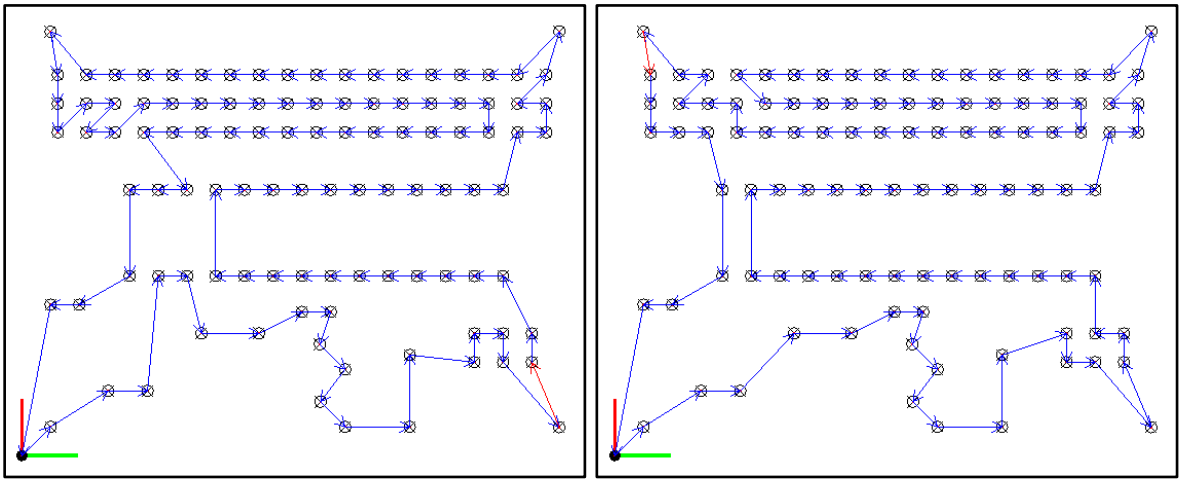
Kolejną grupą rozwiązań poddanych poprawie metodą *2-opt* były rozwiązania uzyskane metodą NN we wszystkich opisanych wcześniej wariantach. Rysunek 3.19 prezentuje wynik otrzymany z rozwiązania początkowego obliczonego metodą NN bez

wspomagania siatki. Z lewej pojedyncze wyzwolenie NN (długość ścieżki 369.3 mm w czasie 20 ms), z prawej wielokrotne wyzwolenie NN (długość ścieżki 358.5 mm w czasie 38 ms).



Rys. 3.19. Wynik metody *2-opt* z rozwiązaniem początkowym NN (bez wspomagania siatki)

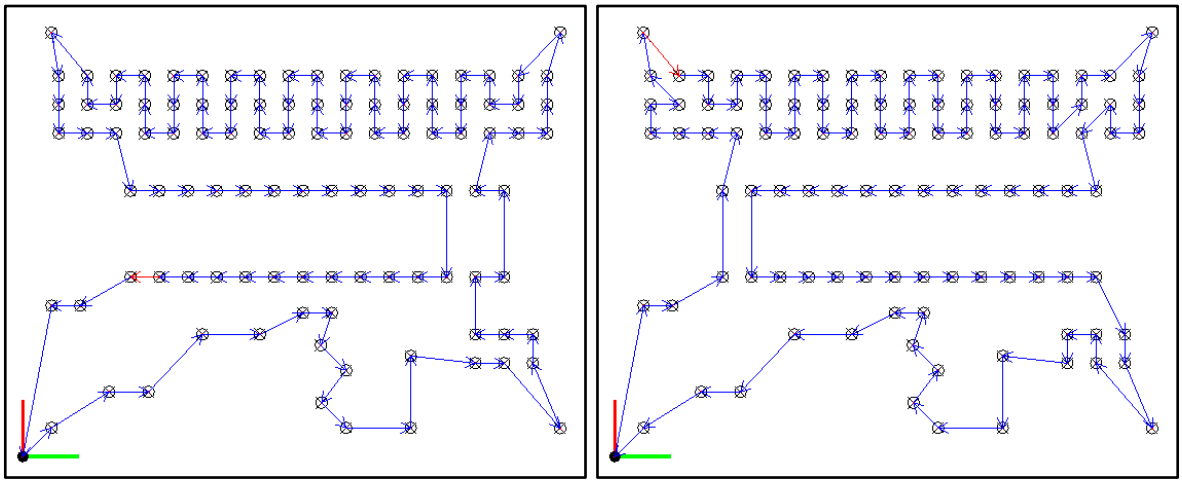
Rysunek 3.20 przedstawia wynik polepszenia *2-opt* rozwiązania początkowego uzyskanego metodą NN z priorytetem osi *X* dla siatek. Z lewej pojedyncza iteracja z wynikiem 359.1 mm uzyskanym w czasie 31 ms, z prawej wielokrotne wykonanie NN, długość: 350.7 mm, czas: 37 ms.



Rys. 3.20. Wynik metody *2-opt* z rozwiązaniem początkowym NN (priorytet osi *X*)

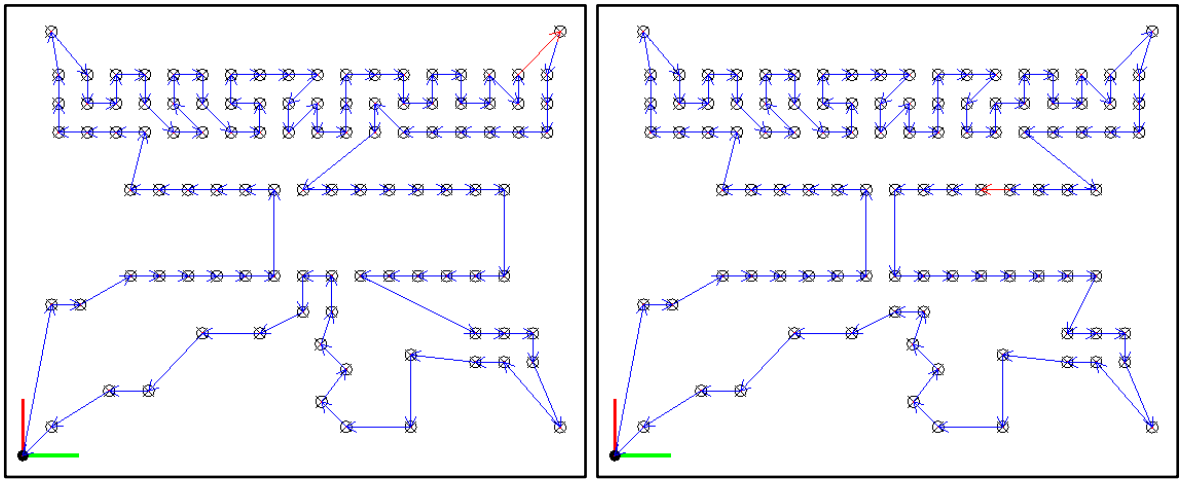
Ostatnie rozwiązania bazujące na metodzie NN zostały uzyskane poprzez zastosowanie priorytetu osi *Y* dla siatek i zaprezentowane na rysunku 3.21. Algorytm *2-opt* poprawił rozwiązanie uzyskane pojedynczą iteracją NN i osiągnął rezultat 377.1 mm w czasie 16 ms (lewa część ilustracji), a dla rozwiązania początkowego uzyskanego metodą NN wielokrotną uzyskał wynik 358.9 mm w czasie 39 ms (prawa część rysunku).

Rozwiązania uzyskane metodą *2-opt*, bazujące na wyniku metody FI, zaprezentowano na rysunku 3.22. Dla jednej iteracji FI wynik wynosi 365.1 mm w czasie 14 ms (lewa część rysunku 3.22), natomiast dla wielokrotnej wersji FI wynik *2-opt* wynosi 358.1 mm w czasie 702 ms (prawa część rysunku).



Rys. 3.21. Wynik metody *2-opt* z rozwiązaniem początkowym NN (priorytet osi *Y*)

Pomimo lepszych wyników osiąganych przez algorytm FI względem NN wyniki FI + *2-opt* nie są jednoznacznie lepsze niż NN + 2-opt.

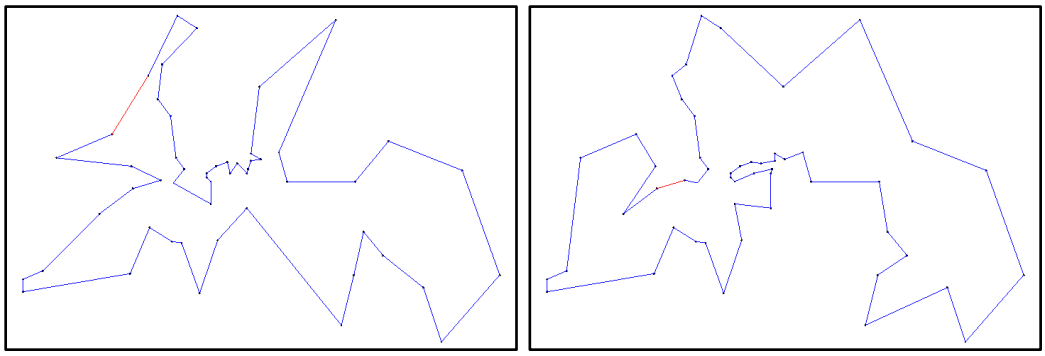


Rys. 3.22. Rozwiązania uzyskane metodą *2-opt* z rozwiązaniem początkowym FI

Kolejnym zadaniem wykorzystanym do analizy efektywności metody *2-opt* jest znane już zadanie *Berlin52*. Na kolejnych ilustracjach pokazano wyniki obliczeń tego zadania uzyskane dla różnych postaci rozwiązań początkowych.

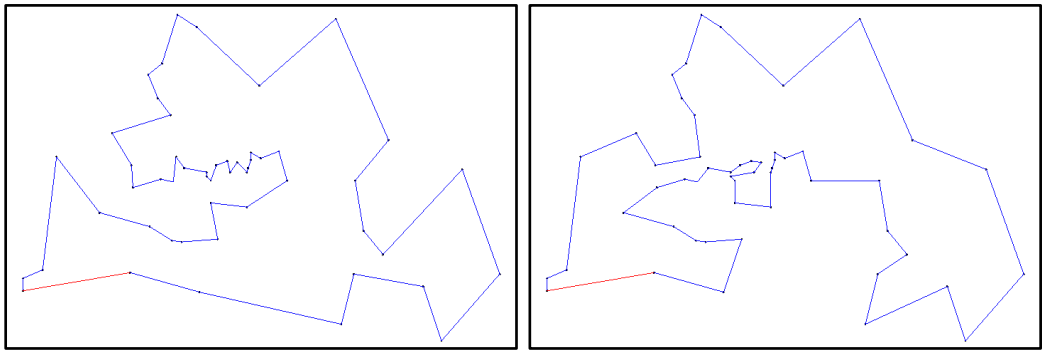
Rysunek 3.23 przedstawia przykładowe wyniki uzyskiwane metodą *2-opt* dla losowych rozwiązań początkowych. Z lewej ścieżka o długości 8401.0 m uzyskana w czasie 46 ms, z prawej ścieżka o długości 7770.1 m obliczona w 24 ms.

Podobnie jak dla zadania płyty PCB, wykonano 10 prób obliczenia ścieżki dla losowego rozwiązania początkowego. Szczegółowe wyniki podano w tabeli 3.3.



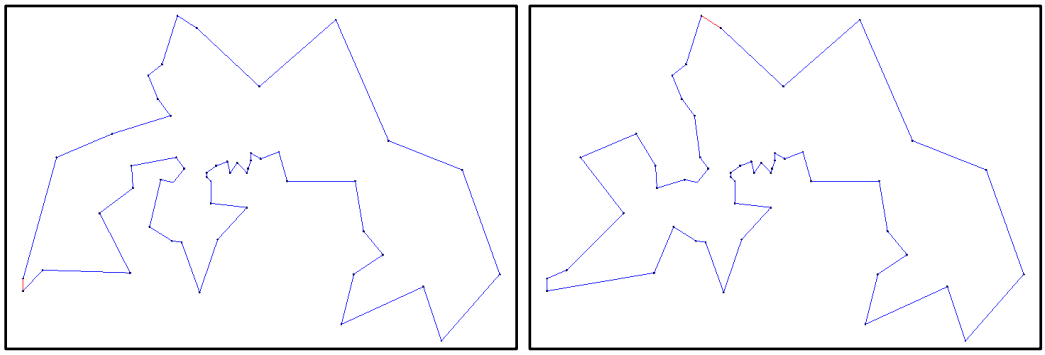
Rys. 3.23. Wynik metody *2-opt* z losowym rozwiązaniem początkowym

Na rysunku 3.24 pokazano wyniki metody *2-opt* dla rozwiązań początkowych uzyskanych metodą NN. Z lewej znajduje się wynik dla jednokrotnej metody NN (długość ścieżki: 8114.4 m, czas: 3 ms), z prawej dla wielokrotnej wersji NN (ścieżka o długości 7864.9 m obliczona w 6 ms)



Rys. 3.24. Wynik metody *2-opt* z rozwiązaniem początkowym NN

Na rysunku 3.25 Pokazano wyniki *2-opt* dla rozwiązania początkowego FI. Z lewej wynik dla jednokrotnej metody FI (długość ścieżki: 7783.0 m, czas: 2 ms). Z prawej wynik metody wielokrotnej FI (długość: 7544.4 m, czas: 45 ms).



Rys. 3.25. Wynik metody *2-opt* z rozwiązaniem początkowym FI

W tabeli 3.3 pokazano zestawienie wyników metody *2-opt* dla wszystkich dostępnych w programie *PCB CAM Processor* postaci rozwiązań początkowych. *Presolver* oznacza metodę użytą do wygenerowania rozwiązania początkowego. Metoda losowa (ang. Random - RND) została uruchomiana 10 krotnie. W tabeli podano wynik minimalny oraz maksymalny wraz z czasami ich uzyskania (czas podany obok wartości *min* nie jest minimalnym czasem metody, lecz odnosi się do rozwiązania minimalnego) oraz błędami. Wartości średnie zostały obliczone z 10 prób (podany czas stanowi średnią z wszystkich prób).

Tabela 3.3. Zestawienie wyników metody *2-opt*

Zadanie	*Presolver*	Licz. iteracji	Długość ścieżki	Czas	Błąd
Płyta PCB	RND	10	351.9 mm - min 381.5 mm - max 362.9 mm - średnia	112 ms 130 ms 129 ms	1.32% 9.85% 4.49%
	NN	1	369.3 mm - brak priorytetu	20 ms	6.33%
			359.1 mm - priorytet osi *X*	31 ms	3.40%
			377.1 mm - priorytet osi *Y*	16 ms	8.58%
		107	358.5 mm - brak priorytetu	38 ms	3.22%
			350.7 mm - priorytet osi *X*	37 ms	0.98%
			358.9 mm - priorytet osi *Y*	39 ms	3.34%
	FI	1	365.1 mm	14 ms	5.13%
		107	358.1 mm	702 ms	3.11%
Berlin52	RND	10	7629.9 m - min 8402.0 m - max 8058.3 m - średnia	37 ms 46 ms 30 ms	1.13% 11.37% 6.81%
	NN	1	8114.4 m	3 ms	7.56%
		52	7864.9 m	6 ms	4.25%
	FI	1	7783.0 m	2 ms	3.16%
		52	7544.4 m	45 ms	0.00%

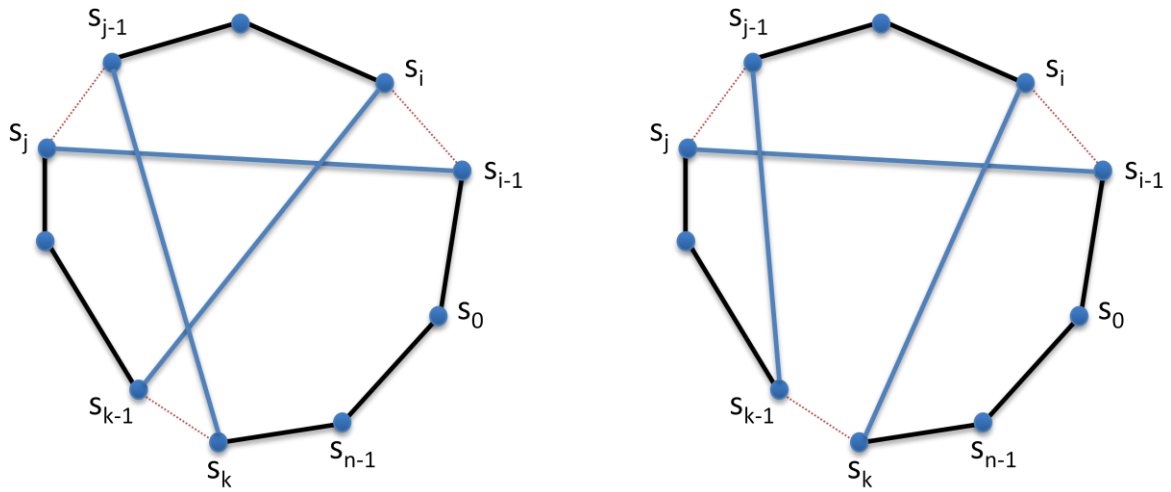
Wnioski:

- Metoda poszukiwań lokalnych *2-opt* osiąga dobre rezultaty w krótkim czasie, ale wymaga dostarczenia odpowiedniego rozwiązania początkowego.

- Analizując rezultaty można zaobserwować silny wpływ rozwiązania początkowego na wynik końcowy. Metoda *2-opt* jest metodą poszukiwań lokalnych, dlatego znalezione rozwiązanie jest jednym z lokalnych minimów znajdujących się w sąsiedztwie rozwiązania początkowego.
- Zwykle lepsza postać rozwiązania początkowego daje lepszy wynik końcowy, jednak dla rozwiązań losowych istnieje możliwość uzyskania porównywalnych wyników jak w przypadku innych metod.
- Dla zadania *Berlin52* udało się uzyskać optymalny wynik przy zastosowaniu rozwiązania początkowego w postaci wielokrotnie przeliczonego FI, natomiast jednokrotna wersja FI nie została polepszona przez *2-opt*.
- Najlepszym podejściem wydaje się wielokrotne przeliczenie zadania dla różnych warunków początkowych i wybór najlepszego z pośród otrzymanych wyników. Oczywiście, jeśli ograniczenia czasowe na to pozwalają.

3.4.2. Algorytm *3-opt*

Metoda *3-opt* jest bliźniaczo podobna do *2-opt*. Jedyną różnicą jest liczba zastępowanych krawędzi. Przy usuwaniu trzech krawędzi istnieje siedem sposobów na rekonfigurację ścieżki, ale można wykazać, że większość z nich jest tożsama i istnieją tylko dwie różne możliwości utworzenia nowego rozwiązania [29]. Rysunek 3.26 przedstawia możliwe sposoby rekonfiguracji cyklu Hamiltona dla metody *3-opt*.



Rys. 3.26. Możliwe rekonfiguracje trzech krawędzi cyklu Hamiltona

Listing 3.11 prezentuje pseudokod metody *3-opt*. Oprogramowana implementacja wyznacza wszystkie możliwe trójki krawędzi, które mogą zostać poddane zamianie. Pomocnicza procedura *NewTour1* została pokazana na listingu 3.12.

Listing 3.11. Algorytm poszukiwania *3-opt*

```
Input/Output: BestSol - rozwiązanie początkowe podlegające poprawie

n = BestSol.Count

do // Główna pętla, wykonywana dopóki udaje się poprawić wynik
{
    Improved = false
    for ( i = 0; i < n - 2 && !Improved; i++ )
    {
        PrevI = ( i == 0 ) ? n-1 : i-1;
        for ( j = i + 1; j < n -1 && !Improved; j++ )
        {
            for ( k = j + 1; k < n && !Improved; k++ )
            {
                if ( NewTour1( i, PrevI, j, k ) ) Improved = true
                if ( NewTour2( i, PrevI, j, k ) ) Improved = true
            }
        }
    }
}
while( Improved )
```

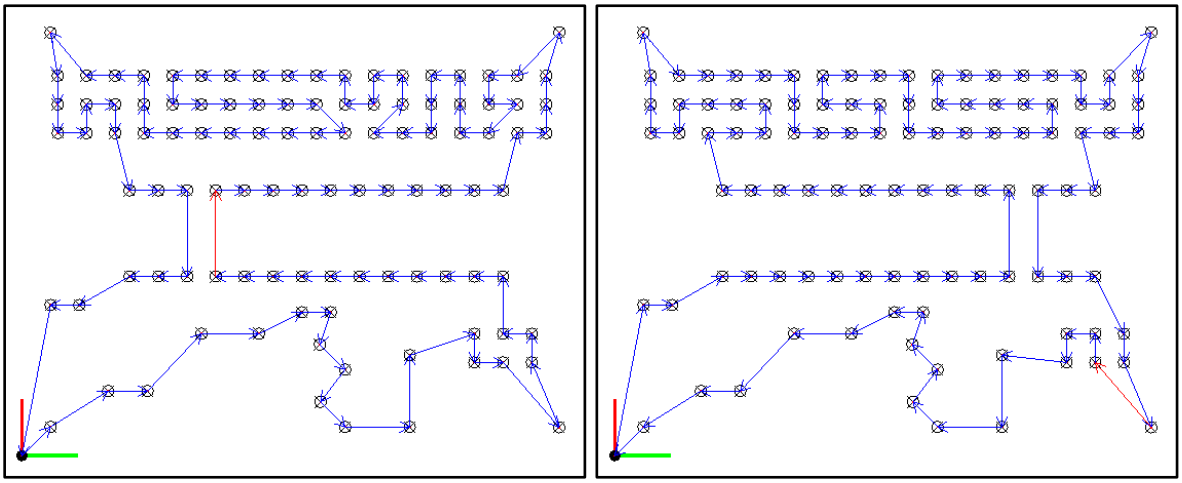
Listing 3.12. Procedura *NewTour1*

```
Input:  i, PrevI, j, k - indeksy podziału cyklu
Output: czy poprawiono rozwiązanie BestSol

n = BestSol.Count
CandLen = BestSol.Len

// Usunięte krawędzie
CandLen -= Dist( BestSol[PrevI], BestSol[i] )
CandLen -= Dist( BestSol[j-1],   BestSol[j] )
CandLen -= Dist( BestSol[k-1],   BestSol[k] )
// Dodane krawędzie
CandLen += Dist( BestSol[PrevI], BestSol[j] )
CandLen += Dist( BestSol[j-1],   BestSol[k] )
CandLen += Dist( BestSol[k-1],   BestSol[i] )

if ( CandLen < BestSol.Len ) // Ocena rozwiązania
{
    // Kompozycja rozwiązania
    for ( h = k; h < n; h++ ) TempSol.AddPoint( BestSol[h] )
    for ( h = 0; h < i; h++ ) TempSol.AddPoint( BestSol[h] )
    for ( h = j; h < k; h++ ) TempSol.AddPoint( BestSol[h] )
    for ( h = i; h < j; h++ ) TempSol.AddPoint( BestSol[h] )

    BestSol = TempSol
    return true
}
return false
```

Algorytm *3-opt* zaprezentowany na listingu 3.11 różni się od *2-opt* (listing 3.9) tym, że posiada dodatkową pętlę *for* dla indeksu *k*. Trójki indeksów *i, j, k* (oraz pomocnicza *PrevI* opisana wcześniej dla *2-opt*) są przekazywane pomocniczym procedurom *NewTour1* oraz *NewTour2*. Pierwsza z nich odpowiada na konstrukcję ścieżki zgodnie z rysunkiem 3.26 (lewa ścieżka), a *NewTour2* konstruuję drugi wariant drogi. Procedura *NewTour2* nie została tu pokazana, gdyż jest niemal identyczna jak *NewTour1*. Różni się indeksami wierzchołków użytych do obliczania *CandLen* (zgodnie z rys. 3.26) oraz jedną z pętli *for* użytych do konstrukcji rozwiązania *TempSol* (odwrócona kolejność dodawania punktów). Ponadto podział konstrukcji ścieżki na dwie procedury został tu wprowadzony dla ułatwienia. W rzeczywistej implementacji programu *PCB CAM Processor* występuje jedna procedura *NewTour*, która jest bardziej rozbudowana i pokrywa oba przypadki.

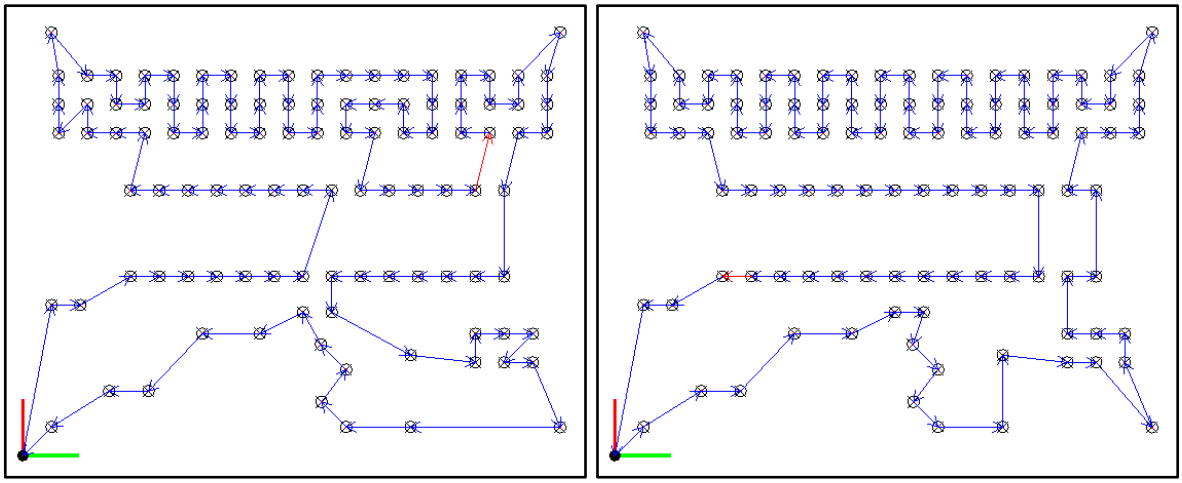
Walidację metody *3-opt* przeprowadzono analogicznie do poprzednich metod. Wyniki obliczeń w postaci zrzutów ekrany programu *PCBCAM Processor* zamieszczono na kolejnych rysunkach. Z uwagi na dużą liczbę wariantów rozwiązań początkowych, przedstawiono jedynie wybrane wyniki. Dla metody rozwiązań początkowych uzyskanych metodą NN pokazano jedynie najlepszy i najgorszy możliwy przypadek. Pełne zestawienie wszystkich wyników zostało zaprezentowana w tabeli 3.4.

Rysunek 3.27 przedstawia przykładowe rozwiązania uzyskane metodą *3-opt* dla losowych warunków początkowych. Po lewej ścieżka o długości 350.7 mm obliczona w czasie 377 ms, a po prawej ścieżka o długości 347.9 mm uzyskana w czasie 265 ms.



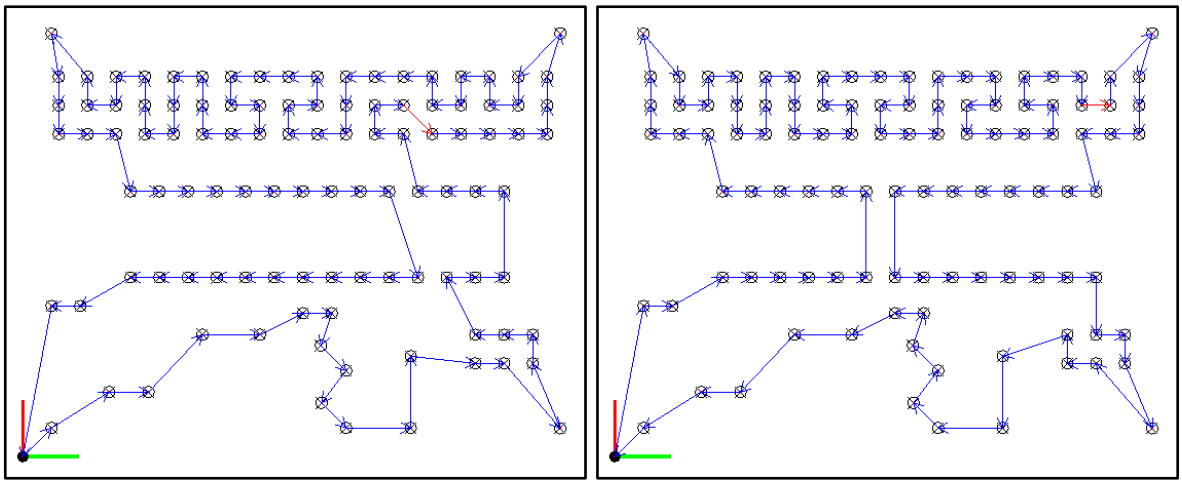
Rys. 3.27. Wynik metody *3-opt* z losowym rozwiązaniem początkowym

Na rysunku 3.28 pokazano wyniki metody *3-opt* dla rozwiązania początkowego NN. Po lewej stronie najgorszy z wyników uzyskany wielokrotną metodą NN bez wsparcia siatki (długość ścieżki: 360.8 mm, czas: 203 ms), po prawej najlepszy rezultat bazujący na jednokrotnej metodzie NN z priorytetem osi *Y* dla siatek (długość: 347.3 mm, czas 212 ms). Wyniki są dość zaskakujące. Dla rozwiązań początkowych uzyskanych wielokrotnymi metodami NN uzyskano gorsze wyniki niż dla jednokrotnych wariantów NN. Mimo lepszej jakości rozwiązania początkowego wynik końcowy może być gorszy.



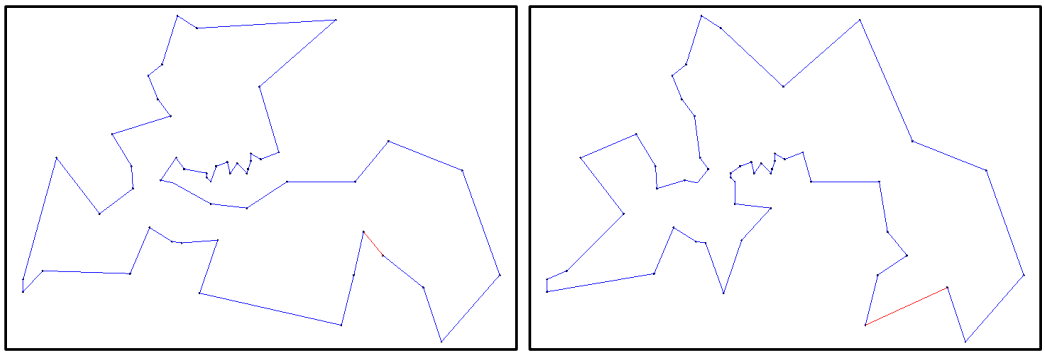
Rys. 3.28. Wyniki metody *3-opt* z rozwiązaniem początkowym NN

Wyniki metody *3-opt* dla rozwiązań początkowych uzyskanych metodą FI zaprezentowano na rysunku 3.29. W obu wariantach metody FI rozwiązanie końcowe było bardzo bliskie optymalnemu. Po lewej ścieżka o długości 349.3 mm obliczona w czasie 342 ms dla jednokrotnej metody FI. Po prawej rozwiązanie dla wielokrotnej metody FI o długości 347.6 uzyskane w czasie 879 ms.



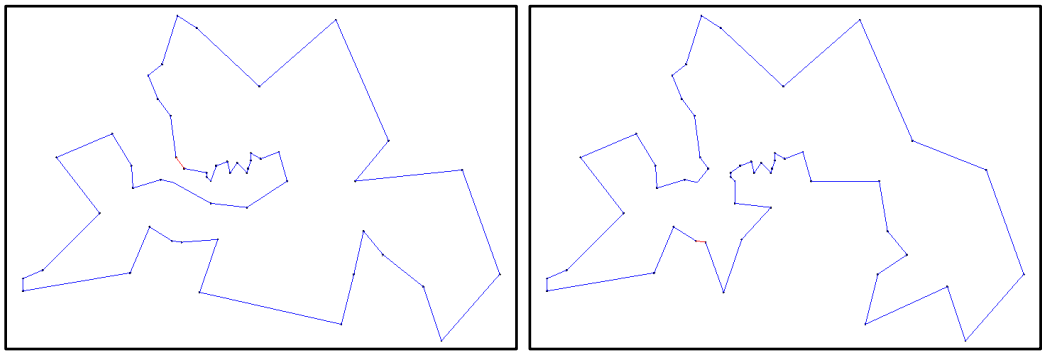
Rys. 3.29. Wyniki metody *3-opt* z rozwiązaniem początkowym FI

Kolejne ilustracje przedstawiają walidację metody *3-opt* dla znanego już zadania *Berlin52*. Ponownie za rozwiązania początkowe przyjęto metodę losową oraz NN i FI.



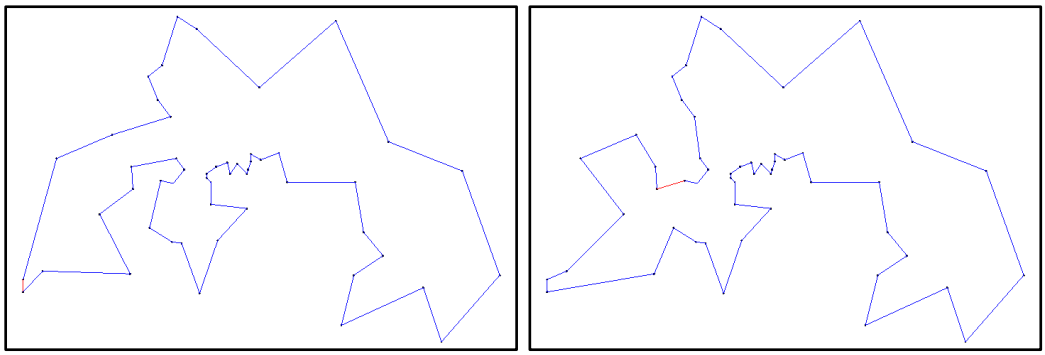
Rys. 3.30. Wyniki metody *3-opt* z losowym rozwiązaniem początkowym

Na rysunku 3.30 pokazano wyniki metody *3-opt* dla losowych rozwiązań początkowych. Z lewej znajduje się najgorszy z uzyskanych w 10 próbach wyników (długość ścieżki: 7991.0 m, czas: 50 ms), z prawej najlepszy (długość ścieżki: 7544.4 m, czas: 82 ms).



Rys. 3.31. Wyniki metody *3-opt* z rozwiązaniem początkowym NN

Rysunek 3.31 Prezentuje wyniki metody *3-opt* dla rozwiązań początkowych NN. Z lewej wynik dla jednokrotnej metody NN (długość ścieżki: 7887.2 m, czas: 37 ms). Dla wielokrotnej metody NN (z prawej) uzyskano optymalny wynik 7544.4 m w czasie 46 ms.



Rys. 3.32. Wyniki metody *3-opt* z rozwiązaniem początkowym FI

Wyniki metody *3-opt* dla rozwiązań początkowych FI prezentuje rysunek 3.32. Z lewej jednokrotna wersja FI z wynikiem 7783.0 m uzyskanym w 12 ms (podobnie jak dla *2-opt* brak poprawy rozwiązania początkowego). Z prawej wynik optymalny uzyskany w 56 ms bazując na wielokrotnej metodzie FI.

W tabeli 3.4 zestawiono wyniki metody *3-opt* dla wszystkich możliwych postaci rozwiązań początkowych. Organizacja tabeli oraz uwagi do niej są analogiczne jak dla metody *2-opt* (tabela 3.3). Rozwiązania losowe zostały również uzyskane w 10 próbach.

Tabela 3.4. Zestawienie wyników metody *3-opt*

Zadanie	Presolver	Licz. iteracji	Długość ścieżki	Czas	Błąd
Płyta PCB	RND	10	347.9 mm - min 356.6 mm - max 351.1 mm - średnia	265 ms 352 ms 363 ms	0.17% 2.68% 1.10%
	NN	1	355.5 mm - brak priorytetu	392 ms	2.36%
			349.7 mm - priorytet osi *X*	158 ms	0.69%
			347.3 mm - priorytet osi *Y*	212 ms	0.00%
		107	360.8 mm - brak priorytetu	203 ms	3.89%
			349.4 mm - priorytet osi *X*	264 ms	0.60%
			351.0 mm - priorytet osi *Y*	236 ms	1.07%
	FI	1	349.3 mm	342 ms	0.58%
		107	347.6 mm	879 ms	0.09%
Berlin52	RND	10	7544.4 m - min 7991.0 m - max 7819.2 m - średnia	82 ms 50 ms 67ms	0.00% 5.92% 3.64%
	NN	1	7887.2 m	37 ms	4.54%
		52	7544.4 m	46 ms	0.00%
	FI	1	7783.0 m	12 ms	3.16%
		52	7544.4 m	56 ms	0.00%

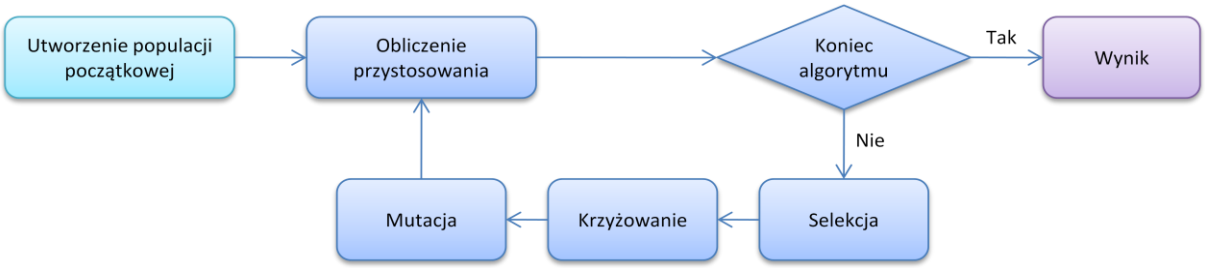
Wnioski:

- Metoda poszukiwań lokalnych *3-opt* w każdym przypadku osiąga zdecydowanie lepsze rezultaty niż *2-opt* bazując na tych samych rozwiązaniach początkowych.

- Czas obliczeń jest jednak zdecydowanie dłuższy względem metody *2-opt*, dlatego przy jej zastosowaniu dla większych zadań, należy brać pod rozwagę ograniczenia czasowe.
- Ponownie widoczny jest bardzo silny wpływ rozwiązania początkowego na wynik końcowy, a wręcz pewne zaskakujące obserwacje. Mimo lepszego rozwiązania początkowego, możliwe jest uzyskanie gorszego wyniku niż dla słabszego rozwiązania początkowego.
- Metoda ma wysoką efektywność. Dla pewnych warunków początkowych potrafi osiągać rozwiązania optymalne lub bardzo bliskie optymalnemu (błąd poniżej 1%).

3.5. Algorytmy genetyczne

Metody genetyczne należą do podejść ewolucyjnych, bazujących na obserwacji przyrody. Przewagą GA w stosunku do prezentowanych wcześniej metod jest odporność na znajdowanie lokalnych minimów. Dzięki losowości, którą kieruje się algorytm, istnieje możliwość badania znacznie szerszych przestrzeni rozwiązań dopuszczalnych oraz istnieje możliwość znalezienia rozwiązania optymalnego. Schemat blokowy, przedstawiający ideę GA przedstawiono na rysunku 3.33.



Rys. 3.33. Schemat blokowy optymalizacji metodą GA

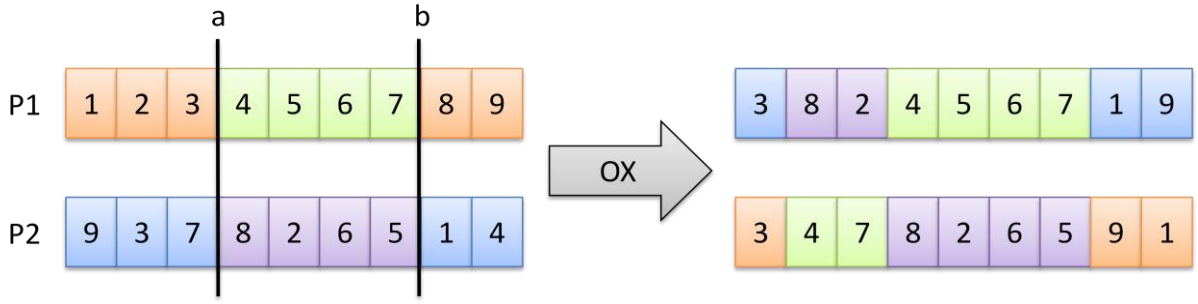
W metodzie genetycznej każde rozwiązanie jest traktowane jako osobnik pewnej populacji. Każdy osobnik jest w pewien sposób przystosowany do życia w danym środowisku. Miarą przystawania osobnika (rozwiązania) w algorytmie jest tzw. f*unkcja przystosowania*. Podobnie jak w naturze tylko najlepsze osobniki w danej populacji (te lepiej przystosowane) mają szansę przetrwać i wydać potomstwo. Proces rozmnażania modelowany jest w GA za pomocą mechanizmów krzyżowania i mutacji, które mogą być realizowane na wiele sposobów, zależnie od rodzaju zadania. Kolejne iteracje algorytmu symulują proces

życia populacji, gdzie starsze osobniki zastępowane są przez nowe pokolenia. Cykl przedstawiony na rysunku 3.33 kontynuowany jest do osiągnięcia pewne warunku zatrzymania algorytmu, którym może być np. wykonanie określonej liczby iteracji.

Istnieje wiele sposobów realizacji każdego z kroków algorytmu. Wyróżnia się wiele rodzajów tzw. *operatorów krzyżowania* oraz *operatorów mutacji*. Operatory stosowana w zadaniach TSP stanowią odrębną grupę. W literaturze można znaleźć porównanie najczęściej spotykanych operatorów dedykowanych zadaniom TSP [1].

3.5.1. Standardowe podejście genetyczne

Pierwsza z proponowanych w tej pracy metod bazujących na GA wykorzystuje jedynie mechanizmy: selekcji, krzyżowania i mutacji zgodnie z rysunkiem 3.33. Spośród wielu dostępnych realizacji selekcji wybrano metodę *listy rankingowej*, która jest prosta do zaimplementowania. Rozwiązania znajdują się na liście posortowanej względem rosnącej wartości funkcji przystosowania. Dla zadania TSP można określić tę funkcję jako długość ścieżki. Im krótsza ścieżka tym wyżej na liście znajduje się dane rozwiązanie. Jedynie pewna grupa najlepszych rozwiązań brana jest pod uwagę przy generowaniu nowego pokolenia. W procesie krzyżowania osobniki z tej grupy dobierane są losowo w pary i generują ustaloną wcześniej liczbę potomków. Krzyżowanie odbywa się przy pomocy operatora OX (ang. Ordered Crossover), który okazał się najlepszy w przytoczonych wcześniej badaniach [1]. Ideę operatora OX przedstawiono na rysunku 3.34.



Rys. 3.34. Zasada działania operatora krzyżowania OX

Zapis rozwiązania TSP (cylku Hamiltona) w postaci osobnika populacji algorytmu GA odbywa się za pomocą numerów określających kolejność odwiedzanych wierzchołków. Położenie punktów zapisane jest w tablicy, której indeksy odpowiadają numeracji punktów. Proste operatory krzyżowania stosowane do problemów numerycznych nie mają tu zastosowania, ponieważ nie można zwyczajnie wziąć pewnego fragmentu od jednego rodzica

i innego od drugiego. Generowałoby to nieprawidłowe rozwiązania tj. takie, które nie są cyklem Hamiltona. Aby zagwarantować, że wszystkie wierzchołki znajdą się w rozwiązaniu dokładnie raz potrzebne jest nieco bardziej złożone postępowanie. Na początku należy wybrać losowo pewną parę indeksów *a* i *b*, między którymi zachodzi relacja $a < b$. Indeksy wyznaczają tzw. punkty krzyżowania. Wierzchołki cyklu znajdujące się pomiędzy punktami krzyżowania przechodzą do potomków w całości w dokładnie to samo miejsce, jakie zajmowały u rodzica. Pozostała część ścieżki kopiowana jest od drugiego rodzica poczynając od drugiego punktu krzyżowania. Kolejno ustalane są indeksy od *b* do *n*, a następnie od *1* do *a*. Jeśli numer wierzchołka kopiowanego z rodzica do potomka występuje już w potomku to zostaje on pominięty i brany jest pod uwagę kolejny indeks. Na rysunku 3.34 zaznaczono kolorami, które fragmenty pochodzą od danego rodzica. Każde krzyżowanie generuje dwóch różnych potomków. Pseudokod implementacji operatora OX pokazano na listingu 3.13.

Listing 3.13. Implementacja operatora krzyżowania OX

```
Input:  Parent1, Parent2 - wybrane rozwiązania
Output: Child1, Child2 - nowe rozw. powstałe w procesie krzyżowania

a = RandomValue( 1,   n-3 ) // losowe p. krzyżowania
b = RandomValue( a+2, n-1 )

j1 = j2 = b // j1, j2 pomocnicze indeksy

for ( i = a; i < b; i++ ) // środkowy fragment od pierwszego z rodziców
{
    Child1[i] = Parent1[i]
    Child2[i] = Parent2[i]
}

for ( i = b; i < n; i++ ) // ustalenie numerów Child[i] od b do n
{
    j1 = OXSetAllel( Parent2, Child1, a, b, i, j1 )
    j2 = OXSetAllel( Parent1, Child2, a, b, i, j2 )
}

for ( i = 0; i < a; i++ ) // ustalenie numerów Child[i] od 0 do a
{
    j1 = OXSetAllel( Parent2, Child1, a, b, i, j1 )
    j2 = OXSetAllel( Parent1, Child2, a, b, i, j2 )
}
```

Wejściem zaprezentowanej na listingu 3.13 procedury jest para rodziców *Parent1* oraz *Parent2*, które mają wydać potomstwo *Child1* oraz *Child2* będące wynikiem pracy procedury. W pierwszym koku generowane są losowe wartości punktów krzyżowania *a* oraz *b*. Funkcja *RandomValue* zwraca losową liczbę całkowitą z przedziału domkniętego wyznaczonego przez argumenty. Zmienna *n* stanowi rozmiar zadania TSP. Indeks *b* musi spełniać zależność:

$b \geq a+2$, aby krzyżowanie miało sens. Pomocnicze indeksy j_1 oraz j_2 służą do zapamiętania ostatniego miejsca kopiowania z rodzica. Procedura zawiera trzy pętle *for*, które odpowiadają za kopiowanie kolejnych fragmentów z rodziców do potomków. Najpierw kopiowany jest środkowy fragment (znajdujący się pomiędzy punktami krzyżowania) *Parent1* do *Child1* oraz z *Parent2* do *Child2*. Następnie ustalane są numery wierzchołków zapisane pod indeksami od *b* do *n-1* oraz od *0* do *a-1*. Te dwie ostatnie pętle wykorzystują pomocniczą procedurę *OXSetAllel*, która odpowiada za kopiowanie niepowtarzających się fragmentów z rodzica do dziecka. Pseudokod tej procedury znajduje się na listingu 3.14.

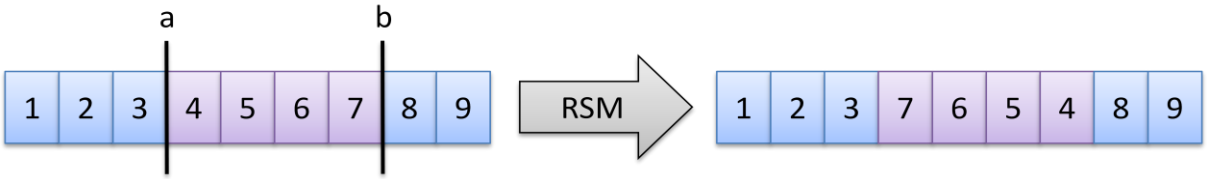
Listing 3.14. Procedura *OXSetAllel*

```
Input:  Parent, Child, a, b, i, j
Output: j

do
{
    Cand = Parent[j]
    CandOK = true

    for ( k = a; k < b; k++ )// czy dziecko ma już ten allel między a i b
    {
        if ( Cand == Child[k] )
        {
            CandOK = false
            break
        }
    }

    if ( ++j == n ) j = 0 // zwiększenie j z zapętleniem n->0

    if ( CandOK )
    {
        Child[i] = Cand
        return j
    }
}
while ( !CandOK )
```

Procedura *OXSetAllel* ustala *i-ty* element tablicy *Child* kopiując element z tablicy *Parent*, którego indeks zaczyna się od *j*. Kopiowany element musi być różny od skopiowanych wcześniej, znajdujących się pomiędzy indeksami *a* i *b*. Głowna pętla odpowiada za przegląd kolejnych kandydujących punktów, które mogą zostać dodane do ścieżki. Element *Parent[j]* zostaje ustalony jako kandydat (*Cand*) i wstępnie zatwierdzony jako odpowiedni (*CandOK* = *true*). Następnie pętla *for* sprawdza czy *Cand* nie znajduje się już w rozwiązaniu *Child* między punktami krzyżowania. Jeśli tak jest kończy sprawdzanie negując kandydata (*CandOK* = *false*). W dalszym kroku zwiększany jest indeks *j*, z którego kopiowane są kandydujące numery punktów. W ostatnim kroku, jeśli *CandOK* następuje

ustalenie pozycji *Child[i] = Cand* oraz zwrócenie indeksu *j*, w przeciwnym wypadku następuje powtórzenie próby ustalenia *Child[i]* kolejnym elementem z tablicy *Parent*.

Kolejnym ważnym elementem algorytmu genetycznego jest mutacja. Istnieje wiele możliwych *operatorów mutacji*. W opisywanej implementacji GA zastał zastosowany operator RSM (ang. Reverse Sequence Mutation - mutacja odwróconej kolejności). Jest on bardzo prosty, gdyż wymaga jedynie odwrócenia kolejności wierzchołków ścieżki między dwoma losowymi punktami *a* i *b*. Taka mutacja jest podobna do rekonfiguracji *2-opt* opisywanej wcześniej i dlatego została wybrana. Rysunek 3.35 przedstawia schematycznie działanie operatora RSM.



Rys. 3.35. Zasada działania operatora mutacji RSM

Ogólny schemat zaimplementowanego w oprogramowaniu *PCB CAM Processor* algorytmu GA przestawiono na listingu 3.15

Listing 3.15. Algorytm GA

```
Generowanie populacji początkowej
Obliczenie przystosowania
Sortowanie rozwiązań

for ( Liczba iteracji )
{
    Selekcja najlepszych
    Krzyżowanie wybranych osobników
    Mutacja RSM
    Zastąpienie starej populacji nowym pokoleniem

    if ( Elitaryzm ) Pozostawienie najlepszego osobnika starej populacji

    Obliczenie przystosowania
    Sortowanie rozwiązań
}

Zwrócenie wyniku - najlepszego osobnika
```

Generowanie populacji początkowej polega na utworzeniu określonej liczby osobników, które reprezentują postać rozwiązania uzyskanego jedną z opisanych wcześniej metod (RND, NN lub FI). Aby zróżnicować populację, każdy osobnik podlega następnie mutacji RSM. Obliczenie przystosowania to wyznaczenie długości ścieżki, jaką reprezentuje każdy osobnik. Aby wybrać pewną grupę najlepszych osobników należy utworzyć ranking,

czyli posortować rozwiązania od najlepszych do najgorszych. Osobniki z puli rodzicielskiej (najlepsze w populacji) są losowo dobierane w pary i poddane krzyżowaniu OX. Każde krzyżowanie generuje dwóch potomków. Istnieje możliwość ustalenia liczby krzyżowań pary rodziców. Algorytm jest tak ułożony, że liczba ta ma ścisły związek z procesem selekcji i rozmiarem populacji. Liczba krzyżowań stanowi coś w rodzaju współczynnika przyrostu naturalnego. Jeżeli grupa 100 rodziców zostanie skrzyżowana jeden raz, to osobników potomnych będzie również 100 (mnożnik populacji równy jeden), jeśli będą dwa krzyżowania, to potomków będzie 200. Aby utrzymać populację na stałym poziomie do kolejnego krzyżowania należy wybrać ponownie 100 z 200 osobników. Prezentowany algorytm posiada jeszcze możliwość pozostawienie w nowej populacji najlepszego osobnika z puli rodzicielskiej (pozostałe giną bez względu na swoje przystosowanie), takie postępowanie nosi nazwę *elitaryzmu*. Algorytm kończy się, kiedy zostanie wykonana określona liczba iteracji, a wynikiem końcowym jest postać rozwiązania prezentowana przez najlepszego osobnika populacji. *Solvera* GA dostępny w prezentowanym oprogramowaniu posiada następujące parametry:

- Wielkość puli rodzicielskiej
- Liczba krzyżowań pary
- Prawdopodobieństwo mutacji RSM
- Elitaryzm (włącz/wyłącz)
- Liczba iteracji algorytmu

Wielkość populacji wynika z iloczynu dwóch pierwszych parametrów. Prawdopodobieństwo mutacji jest liczbą z przedziału [0, 1]. Oznacza, że w każdej populacji potomnej, określona przez nie liczba osobników (wybranych losowo) zostanie zmutowana operatorem RSM.

Walidacje algorytmu genetycznego przeprowadzono dla tych samych zadań jak dla prezentowanych wcześniej metod. Aby znaleźć najlepszy zestaw parametrów, uruchomiono GA dla różnych nastaw:

- Wielkość puli rodzicielskiej: 100, 1000, 10000
- Liczba krzyżowań pary (selekcja): 2, 3, 5, 10
- Prawdopodobieństwo mutacji RSM: 0.01, 0.05, 0.1, 0.3, 0.5, 0.7

- Elitaryzm: tak
- Liczba iteracji algorytmu: wystarczająca do ustabilizowania populacji lub ograniczona czasowo.
- Losowe rozwiązanie początkowe

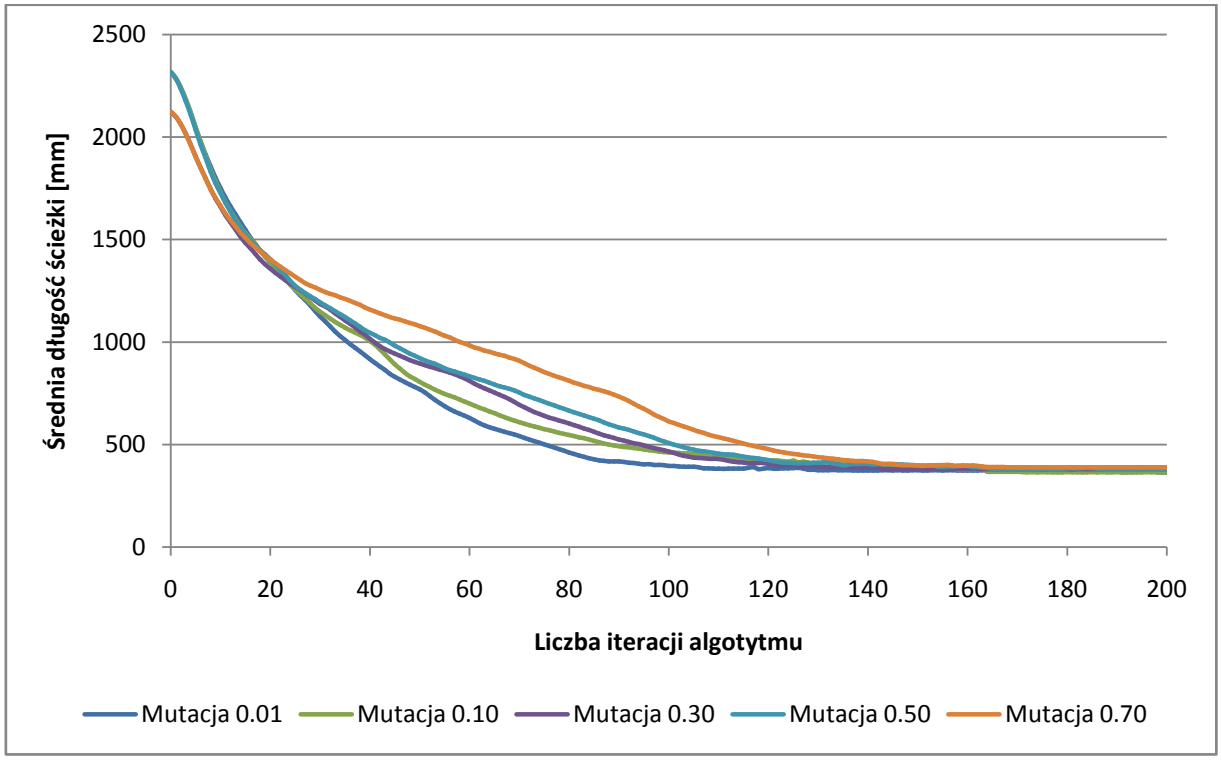
Dla wybranych kombinacji powyższych parametrów wykonano po 10 prób obliczenia długości ścieżki dla zadania płyty PCB. Tabela 3.5 przedstawia średnie wyniki oraz średni czas obliczeń każdej z prób. Poniżej wartości selekcji (liczby krzyżowań pary) podano w nawiasie liczbę wykonywanych iteracji algorytmu.

Tabela 3.5. Dobór parametrów algorytmu GA dla zadania płyty PCB

Wiel. puli rodz.	Selekcja	Współczynnik mutacji					
		0.01	0.05	0.10	0.30	0.50	0.70
100	2 (50000 it.)	379.7 mm 61 s	387.9 mm 66 s	458.6 mm 70 s	468.5 mm 71 s	495.0 mm 72 s	503.6 mm 72 s
	3 (20000 it.)	387.0 mm 36 s	385.1 mm 35 s	384.8 mm 36 s	375.8 mm 36 s	420.3 mm 41 s	477.9 mm 42 s
	5 (10000 it.)	386.3 mm 30 s	383.9 mm 30 s	387.9 mm 31 s	383.1 mm 30 s	372.4 mm 30 s	374.2 mm 31 s
	10 (5000 it.)	388.5 mm 30 s	385.4 mm 30 s	382.4 mm 31 s	377.6 mm 31 s	374.4 mm 31 s	371.9 mm 31 s
1000	2 (5000 it.)	888.6 mm 72 s	905.8 mm 72 s	885.9 mm 72 s	954.0 mm 73 s	-	-
	3 (2000 it.)	371.6 mm 40 s	370.8 mm 40 s	593.9 mm 43 s	779.3 mm 44 s	-	-
	5 (1000 it.)	369.3 mm 35 s	365.8 mm 35 s	375.4 mm 35 s	**369.1 mm 36 s**	455.7 mm 37 s	-
	10 (500 it.)	372.7 mm 39 s	367.9 mm 38 s	368.7 mm 39 s	365.9 mm 38 s	**361.8 mm 38 s**	365.4 mm 39 s
10000	2 (500 it.)	1142.8 mm 95 s	1197.8 mm 96 s	1216.4 mm 96 s	-	-	-
	3 (200 it.)	711.8 mm 64 s	769.0 mm 64 s	872.8 mm 64 s	-	-	-
	5 (100 it.)	502.3 mm 64 s	595.6 mm 64 s	596.9 mm 65 s	-	-	-
	10 (50 it.)	640.2 mm 93 s	637.1 mm 94 s	665.9 mm 95 s	-	-	-

Z przedstawionych w tabeli 3.5 rezultatów wynika, że wartość współczynnika mutacji ma wpływ na tempo stabilizacji populacji. Im wyższa mutacja tym wolniejsza stabilizacja. Selekcja również wpływa na tempo stabilizacji. Im więcej krzyżowań pary w każdej iteracji tym szybsze tempo stabilizacji populacji. Większa populacja ma większą szansę wydać wartościowych potomków, ale aby zachować zbliżony czas obliczeń jak w przypadku mniejszej populacji, należy (proporcjonalnie) zmniejszyć liczbę iteracji, która również ma wpływ na zbieżność algorytmu, ponieważ każda selekcja zawęża niejako obszar poszukiwań. Zbyt duża populacja powoduje bardzo widoczne zwiększenie czasu obliczeń, zbyt mała może nie mieć wystarczająco zróżnicowanej puli genetycznej, aby generować dobrej jakości rozwiązania. Zbieżność wydaje się być najważniejszą cechą algorytmu, jeśli bierzemy pod uwagę szybkość obliczeń. Niestety im wyższa zbieżność tym większe ryzyko utknięcia algorytmu w lokalnym minimum. Okazuje się, że najlepsze wyniki, w najlepszym czasie osiąga GA dla średniej wielkości populacji, selektywności na poziomie 5 oraz mutacji 0.3 lub selektywności 10 oraz mutacji 0.5. Te dwa parametry przeciwstawnie oddziałują na zbieżność, a prawidłowy dobór ich relacji pozwala osiągnąć kompromis pomiędzy czasem obliczeń, a jakością rozwiązania.

Za wykresie 3.1 porównano tempo stabilizacji populacji dla różnych wartości współczynnika mutacji dla puli rodzicielskiej równej 1000 oraz selektywności 10.



Wyk. 3.1. Wartość średniej długości ścieżki w funkcji liczby iteracji algorytmu

Na podstawie wykresu 3.1 można zauważyć, że istnieje możliwość zredukowania liczba iteracji algorytmu dla zadania płyty PCB, co zmniejszy czas obliczeń. Po około 200 iteracjach populacja jest już ustabilizowana.

Z uwagi na słabe wyniki oraz długi czas obliczeń dla losowej populacji początkowej, zdecydowano powtórzyć testy dla populacji początkowej wygenerowanej na bazie rozwiązania metody NN. Aby zróżnicować populację, każdy osobnik został jednokrotnie zmutowany RSM. Wybrano rozmiar puli rodzicielskiej równy 1000 oraz zmniejszono liczby iteracji. Pozostałe parametry ustalono takie same jak w poprzedniej próbie. Wykonano po 20 testów i wyciągnięto średnie. Wynik zestawiono w tabeli 3.6.

Tabela 3.6. Dobór parametrów algorytmu GA dla populacji początkowej uzyskanej metodą NN

Wiel. puli rodz.	Selekcja	Współczynnik mutacji				
		0.01	0.05	0.10	0.30	0.50
1000	2 (1000 it.)	353.3 mm 11.9 s	353.1 mm 11.9 s	353.2 mm 12.1 s	385.0 mm 13.8 s	399.9 mm 14.0 s
	3 (400 it.)	353.7 mm 7.4 s	353.5 mm 7.4 s	353.1 mm 7.3 s	**352.1 mm 7.4 s**	362.2 mm 7.9 s
	5 (200 it.)	355.6 mm 6.5 s	353.8 mm 6.7 s	353.6 mm 6.4 s	353.2 mm 6.5 s	**352.8 mm 6.4 s**
	10 (100 it.)	355.6 mm 7.3 s	352.7 mm 7.2 s	352.9 mm 7.3 s	**353.2 mm 7.1 s**	352.9 mm 7.2 s

Przedstawione w tabeli 3.6 wyniki pokazują bardzo niewielki wpływ współczynnika mutacji na wyniki. Można jednak zauważyć, że najlepsze wyniki osiągnięto dla współczynników mutacji na poziomie 0.3 oraz 0.5. Większość wyników wynosi około 353 mm, co stanowi 1.64% błędu względem optymalnej wartości (347.3 mm).

Tabela 3.7. Wyniki algorytmu GA dla populacji początkowej NN bez reguły elitaryzmu

Wiel. puli rodz.	Selekcja	Współczynnik mutacji				
		0.01	0.05	0.10	0.30	0.50
1000	5 (200 it.)	358.7 mm 6.5 s	354.9 mm 6.4 s	352.9 mm 6.5 s	**352.2 mm 6.4 s**	353.1 mm 6.5 s
	10 (100 it.)	356.7 mm 7.2 s	353.5 mm 7.2 s	352.9 mm 7.2 s	**352.5 mm 7.1 s**	352.6 mm 7.0 s

Kolejne badania przeprowadzono w celu sprawdzenia wpływu reguły *elitaryzmu* na osiągane przez algorytm wyniki. Powtórzono badania dla wybranych parametrów z tabeli 3.6.

Ponownie wykonując po 20 prób. Wyniki pokazano w tabeli 3.7. Analizując wyniki ciężko jednoznacznie potwierdzić lub odrzucić zasadność stosowania reguły elitaryzmu. Zachowanie w populacji najlepszego rozwiązania może doprowadzić do zbyt wczesnego ustabilizowania się populacji w okolicy lokalnego minimum i uniemożliwić dalszą eksplorację przestrzeni rozwiązań. Z kolei usuwanie najlepszego rozwiązania może prowadzić do *cofania* się algorytmu. Wynik końcowy może być wówczas gorszy od znalezionego najlepszego rozwiązania. Brak stosowania elitaryzmu wpływa z pewnością na odporność algorytmu pozwalając mu błądzić i czasowo pogarszać rozwiązanie, co zwiększa jego odporność na osiąganie lokalnych minimów.

Analizując wszystkie uzyskane dotąd wyniki postanowiono ustalić pewien zbiór parametrów algorytmu, który pozwoli w możliwie krótkim czasie osiągnąć zadowalające wyniki. Przeprowadzono po 10 prób dla obu zadań z następującymi parametrami:

- Wielkość puli rodzicielskiej: 1000
- Liczba krzyżowań pary (selekcja): 5
- Prawdopodobieństwo mutacji RSM: 0.3
- Elitaryzm: tak, nie
- Liczba iteracji algorytmu: 100
- Rozwiązanie początkowe: NN.

Wyniki pokazano w tabeli 3.8.

Tabela 3.8. Zestawienie wyników algorytmu GA

Zadanie	*Presolver*	Elitaryzm	Długość ścieżki	Czas	Błąd
Płyta PCB	NN	Tak	350.3 mm - min 356.0 mm - max 353.3 mm - średnia	3406 ms 3403 ms 3342 ms	0.86% 2.51% 1.72%
		Nie	351.1 mm - min 363.0 mm - max 353.9 mm - średnia	3387 ms 3399 ms 3316 ms	1.09% 4.52% 1.90%
Berlin52	NN	Tak	7974.6 m - min 8156.4 m - max 8070.1 m - średnia	1772 ms 1821 ms 1802 ms	5.70% 8.11% 6.97%
		Nie	7993.3 m - min 8178.8 m - max 8113.4 m - średnia	1765 ms 1870 ms 1811 ms	5.95% 8.41% 7.54%

Wnioski:

- Algorytm genetyczny posiada wiele parametrów, których prawidłowy dobór jest trudny i wymaga przeprowadzenia wielu prób.
- Można tak dobrać parametry dla konkretnego zadania, aby osiągać niemal optymalne wyniki (tabela 3.6 i tabela 3.7), przy czym czas obliczeń jest bardzo długi.
- Standardowy GA jest w stanie osiągnąć wyniki porównywalne z metodą *2-opt*, natomiast gorsze od *3-opt*, przy znacznie dłuższym czasie obliczeń.
- Stosując nielosowe rozwiązanie początkowe można skrócić czas obliczeń i poprawić wynik końcowy (tabela 3.5 i tabela 3.6).
- Potrzeba dalszych badań nad GA, aby zwiększyć jego skuteczność. Wprowadzenie dodatkowej *wiedzy* o zadaniu może okazać się obiecujące (rozdział 3.5.2)

3.5.2. Podejście hybrydowe

Standardowy algorytm genetyczny jest *ślepy*, to znaczy nie ma pojęcia o problemie, który rozwiązuje. Przetwarza jedynie ciągi kodowe odpowiadające warunkom zadania. Owe ciągi podlegają zewnętrznej ocenie celem policzenia przystosowania osobnika. Gdy nie wiemy jak poszukiwać rozwiązania, to takie postępowanie jest jak najbardziej uzasadnione, lecz w przypadku TSP dysponujemy szybkimi metodami poszukiwań lokalnych. Wyniki tych metod są oczywiście pewnymi rozwiązaniami lokalnymi, ale ich wymieszanie ze sobą oraz z innymi losowymi rozwiązaniami może dać znacznie lepsze rezultaty. Prezentowana w niniejszej pracy wersja hybrydowego algorytmu genetycznego zakłada wprowadzenie dodatkowego operatora mutacji, który polepsza losowe rozwiązanie metodą *2-opt*. Ogólna idea algorytmu pozostaje bez zmian. Sposób obliczanie *2-opt* jest analogiczny jak opisany wcześniej.

Biorąc pod uwagę wyniki uzyskane dla standardowego algorytmu, wytypowano pewien zbiór nastaw dla algorytmu hybrydowego:

- Wielkość puli rodzicielskiej: 100
- Liczba krzyżowań pary (selekcja): 2, 5, 10
- Prawdopodobieństwo mutacji RSM: 0.3
- Elitaryzm: tak
- Liczba iteracji algorytmu: 250, 100, 50

- Rozwiązanie początkowe: RND
- Prawdopodobieństwo mutacji *2-opt*: 0.001, 0.002, 0.005, 0.010.

Wielkość populacji została zredukowana, aby zmniejszyć nakład obliczeniowy. Liczba iteracji algorytmu również została zmniejszona z uwagi na znacznie lepszą zbieżność. Prawdopodobieństwo mutacji RSM przyjęto na poziomie 0.3. Przyjęto również regułę elitaryzmu, by zminimalizować czas obliczeń. Dla wybranych kombinacji parametrów przeprowadzono badanie skuteczności algorytmu dla zadania płyty PCB. Wykonano po 10 prób dla każdego zestawu parametrów. Wyniki zaprezentowano w tabeli 3.9.

Tabela 3.9 Dobór parametrów algorytmu hybrydowego GA+*2-opt*

Selekcja	Współczynnik mutacji *2-opt*			
	0.001	0.002	0.005	0.010
2 (250 it.)	-	-	348.0 mm 9.1 s	347.9 mm 16.2 s
5 (100 it.)	-	349.5 mm 1.1 s	348.1 mm 1.9 s	**347.6 mm** **4.1 s**
10 (50 it.)	350.8 mm 0.8 s	349.6 mm 1.3 s	347.8 mm 2.5 s	347.8 mm 4.4 s

Na podstawie uzyskanych wyników ustalono, że selekcja na poziomie 5 oraz prawdopodobieństwo mutacji *2-opt* wynoszące 0.1 dają najlepsze rezultaty. Ponadto istnieje możliwość skrócenia czasu obliczeń poprzez zredukowanie liczby iteracji. Do końcowej walidacji metody (dla obu prezentowanych zadań) wybrano następujące parametry:

- Wielkość puli rodzicielskiej: 100
- Liczba krzyżowań pary (selekcja): 5
- Prawdopodobieństwo mutacji RSM: 0.3
- Elitaryzm: tak
- Liczba iteracji algorytmu: 50
- Rozwiązanie początkowe: RND, NN, FI
- Prawdopodobieństwo mutacji *2-opt*: 0.010.

W tabeli 3.10 przedstawiono zestawienie wyników opisanej metody dla zadania płyty PCB oraz dla zadania *Berlin52*, dla trzech wariantów rozwiązań początkowych: RND, NN oraz FI. Każdy wariant obliczono 10-krotnie i wyciągnięto średnią.

Tabela 3.10. Zestawienie wyników algorytmu GA+*2-opt*

Zadanie	*Presolver*	Długość ścieżki	Czas	Błąd
Płyta PCB	RND	347.6 mm - min 347.9 mm - max 347.7 mm - średnia	3601 ms 3582 ms 3490 ms	0.09% 0.17% 0.12%
	NN	347.3 mm - min 348.9 mm - max 348.2 mm - średnia	1809 ms 2071 ms 2007 ms	0.00% 0.46% 0.26%
	FI	347.3 mm - min 355.2 mm - max 350.2 mm - średnia	935 ms 897 ms 1053 ms	0.00% 2.27% 0.84%
Berlin52	RND	7544.4 m - min 7598.4 m - max 7549.8 m - średnia	564 ms 703 ms 495 ms	0.00% 0.72% 0.07%
	NN	7598.4 m - min 7841.4 m - max 7715.7 m - średnia	268 ms 266 ms 274 ms	0.72% 3.94% 2.27%
	FI	7544.4 m - min 7544.4 m - max 7544.4 m - średnia	- - 207 ms	0.00% 0.00% 0.00%

Wnioski:

- Algorytm genetyczny hybrydowy osiąga optymalne lub niemal optymalne wyniki w bardzo krótkim czasie.
- Taka wersja GA może z powodzeniem konkurować z algorytmami poszukiwań lokalnych.
- Algorytm hybrydowy łączy zdolność GA do rozległej eksploracji przestrzeni rozwiązań dopuszczalnych z umiejętnością szybkiego znajdowania lokalnych minimów metodą *2-opt*.
- Dla losowej populacji początkowej algorytm jest zdolny osiągnąć rozwiązanie optymalne oraz ma wysoką powtarzalność, ale czas obliczeń jest najdłuższy. Wynika to z faktu, iż metoda *2-opt* musi wykonywać więcej iteracji. Dla rozwiązań początkowych w postaci NN lub FI algorytm jest szybszy, ale może być mniej odporny.

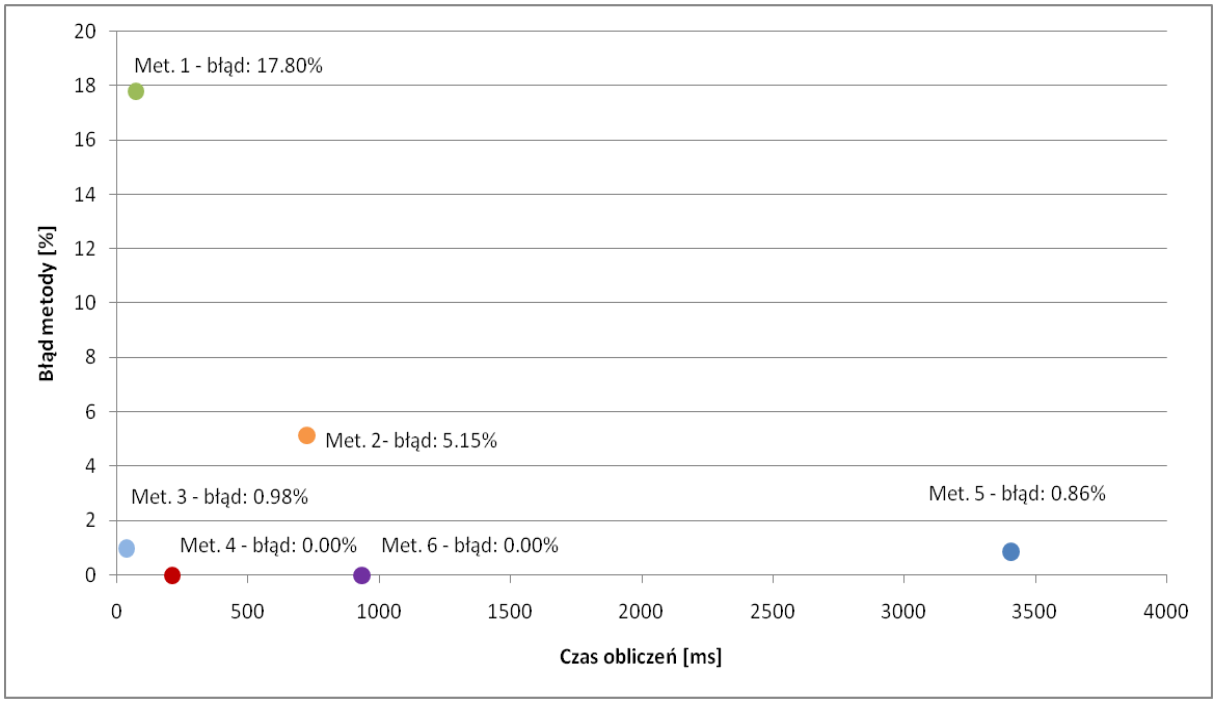
3.6. Porównanie zaimplementowanych metod - wyniki

W tym rozdziale przedstawiono krótkie podsumowanie wszystkich osiągniętych wyników. W tabeli 3.11 zestawiono najlepsze rezultaty osiągnięte każdą z omawianych metod dla obu prezentowanych zadań.

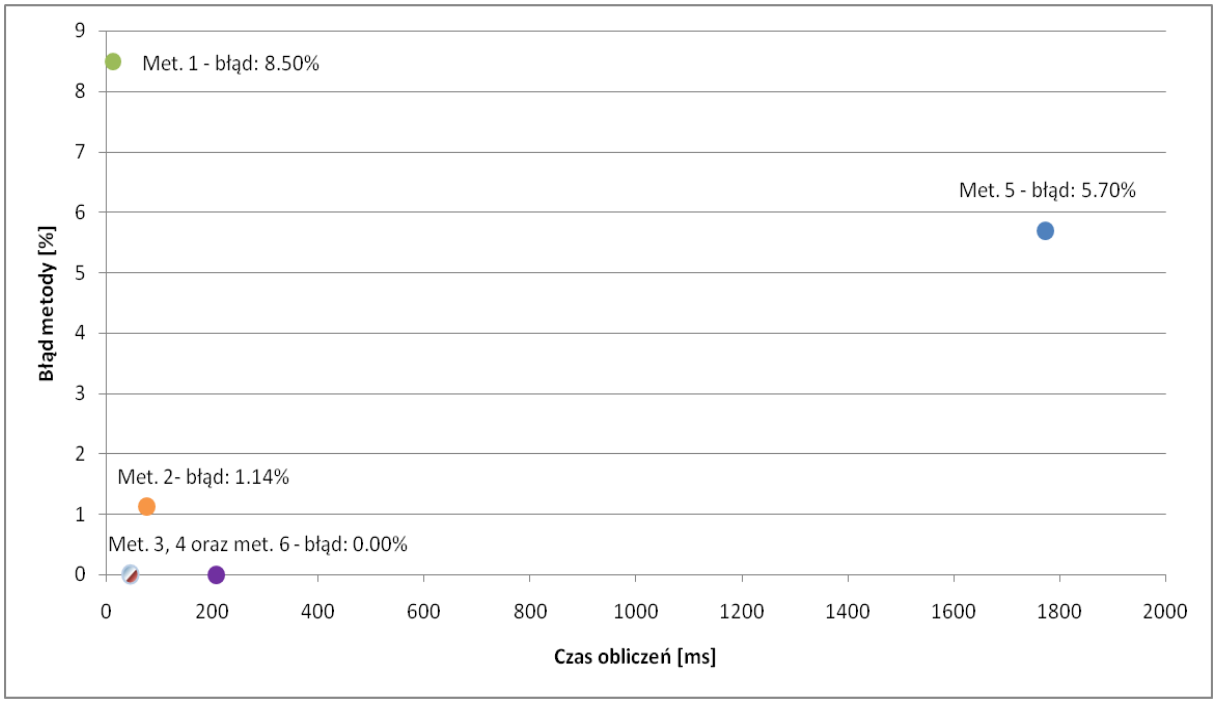
Tabela 3.11. Zestawienie najlepszych wyników wszystkich prezentowanych metod

Zadanie	Metoda		Długość ścieżki	Czas	Błąd
	Lp.	Nazwa			
Płyta PCB	1	Wielokrotna NN - priorytet osi *Y*	409.1 mm	73 ms	17.8%
	2	Wielokrotna FI	365.2 mm	724 ms	5.15%
	3	*2-opt - Presolver:* wielokrotna NN - priorytet osi *X*	350.7 mm	37 ms	0.98%
	4	*3-opt - Presolver:* jednokrotna NN - priorytet osi *Y*	347.3 mm	212 ms	0.00%
	5	GA (NN) z regułą elitaryzmu	350.3 mm	3406 ms	0.86%
	6	GA + *2-opt* (FI) z regułą elitaryzmu	347.3 mm	935 ms	0.00%
Berlin52	1	Wielokrotna NN	8182.2 m	13 ms	8.5%
	2	Wielokrotna FI	7630.6 m	77 ms	1.14%
	3	*2-opt - Presolver:* wielokrotna FI	7544.4 m	45 ms	0.00%
	4	*3-opt - Presolver:* wielokrotna NN	7544.4 m	46 ms	0.00%
	5	GA (NN) z regułą elitaryzmu	7974.6 m	1772 ms	5.70%
	6	GA + *2-opt* (FI) z regułą elitaryzmu	7544.4 m	207 ms	0.00%

Na wykresach 3.2 oraz 3.3 przedstawiono graficzne porównanie efektywności prezentowanych metod dla obu obliczanych zadań.



Wyk. 3.2. Porównanie wyników dla zadania płyty PCB



Wyk. 3.3. Porównanie wyników dla zadania *Berlin52*

3.7. Podsumowanie i wnioski

Spośród wszystkich prezentowanych metod rozwiązywania zadania TSP najszybsze i najskuteczniejsze są metody poszukiwań lokalnych. Wadą tych metod jest jednak bardzo silna wrażliwość na postać rozwiązania początkowego. Aby uzyskać najlepszy możliwy wynik należy obliczyć zadanie wielokrotnie, za każdym razem wybierając inne warunki początkowe, by następnie wybrać jeden z najlepszych wyników. Metoda *3-opt* posiada stosukowo wysoką złożoność obliczeniową. Czas obliczeń rośnie w przybliżeniu z sześcianem rozmiaru zadania, co dla dużych zadań może powodować trudności z jej zastosowaniem.

Z kolei algorytmy genetyczne posiadają w przybliżeniu liniową zależność czasu obliczeniowego od rozmiaru zadania. Jedyną różnicą jest tu rozmiar osobnika, a więc liniowo dłuższe operacje kopiowania pamięci oraz obliczania przystosowania. Dodatkowo algorytm hybrydowy dzięki zastosowaniu operatora mutacji *2-opt* jest w stanie szybko znaleźć dobre rozwiązanie. Dla GA istnieje możliwość sterowania jakością rozwiązania. Można tak dobrać parametry, aby nawet bardzo duże zadanie policzyć w akceptowalnym czasie, podczas gdy dla metod poszukiwań lokalnych czas obliczeniowy może być już nie akceptowalny.

Podsumowując, wybór odpowiedniej metody zależy od postaci zadania oraz od oczekiwań. Jeżeli wymagane jest bardzo dobre rozwiązanie można wybrać metodę *3-opt* lub wolno zbieżny GA lub GA + *2-opt*, natomiast jeśli należy obliczyć dobre rozwiązanie w możliwie jak najkrótszym czasie można zastosować metodę *2-opt* lub szybko zbieżny algorytm hybrydowy z małą liczbą iteracji.

4. OPTYMALIZACJA GRAWEROWANIA - METODY I WYNIKI

4.1. Spotykane podejścia do zadania TSPN

Zadania klasy TSPN stanowią pewne uogólnienie zadania komiwojażera. Najczęstszym podejściem do problemu jest pewna dekompozycja zadnia na dwa bardziej szczegółowe problemy: TSP oraz TPP, które polega na wyznaczeniu najkrótszej ścieżki prowadzącej przez regiony o ustalonej wcześniej kolejności. Szczegółowy opis problemu znajduje się w rozdziale 2.

Alatatsev et al. [3] zaproponowali podejście równoczesnego rozwiązywania zadania szukania sekwencji regionów metodą *3-opt* oraz rozwiązywania zadania TPP metodą RBA (ang. Rubber Band Algorithm) [26]. Rozwiązywany przez nich problem, był podobny do poruszanego w tej pracy optymalnego wykonywania konturów. Autorzy poczynili jednak pewne uproszczenie. Zamodelowali kontury jako elipsy, które są łatwe do obliczania, stanowią gładkie krzywe bez lokalnych minimów i nie mają żadnego odzwierciedlenia w realnych problemach. Zadanie obliczane przez *PCB CAM Processor* jest znacznie bardziej złożone. Kontury składają się z wielu odcinków i łuków tworzących zamknięte krzywe o dowolnym kształcie, których nie da się opisać prostymi funkcjami matematycznymi.

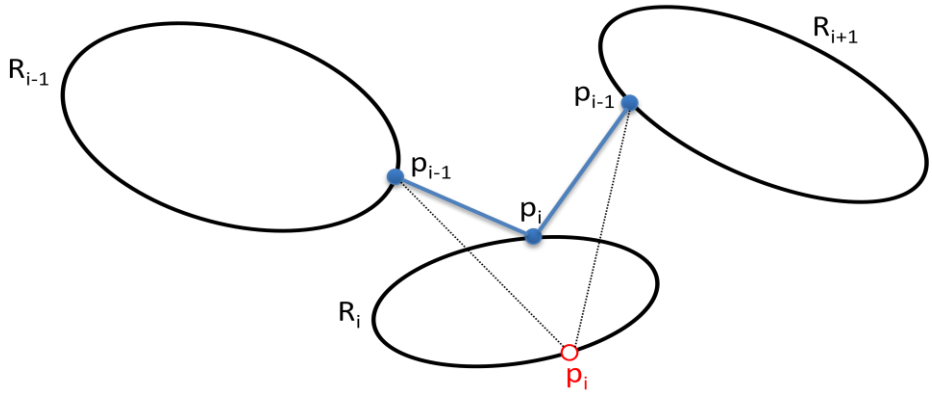
Inny problem podobnej klasy zaprezentował Kovacs [18]. W aplikacji zrobotyzowanego spawania w technologii RWL wykonywane są krótkie spoiny o długości poniżej 30 mm. Każda z nich traktowana jest jako niepodzielne zadanie, które należy wykonać w odpowiedniej sekwencji. Z uwagi na bardzo małe rozmiary spoin w stosunku do gabarytów detalu, można je traktować niemal jako punkty. Problem rozwiązano dzięki zastosowaniu zmodyfikowanej metody FI, do generowania rozwiązania początkowego oraz metody *2-opt* do polepszania wyniku.

Metody poszukiwań lokalnych są najczęściej stosowanym podejściem do rozwiązywania zadań optymalizacji bazujących na problemie TSP. Ponownie ważnym aspektem jest postać rozwiązania początkowego.

4.1.1. Metoda RBA

Podstawowym podejściem do rozwiązywania zadania TPP jest poszukiwanie takich punktów należących do regionów, które minimalizują dystans względem sąsiednich regionów, w których punkty zostały już wybrane. Przypomnijmy, że kolejność odwiedzanych

regionów jest już wcześniej ustalona. Postępowanie należy kontynuować do momentu, kiedy kolejne poszukiwania nie powodują już skrócenia dystansu. Na rysunku 4.1 przestawiono ideę metody RBA.



Rys. 4.1. Optymalizacja położenia punktu p_i w regionie R_i

Celem rozwiązywania zadania TPP jest uzyskanie wyniku, którego postać pokazano na rysunku 4.1. Należy znaleźć nowy punkt p_i w regionie R_i, taki że:

$$|p_{i-1}\, p_i| + |p_i\, p_{i+1}| \rightarrow \min \tag{4.1}$$

Poszukiwanie można prowadzić np. pewną zmodyfikowaną metodą połowienia, które zostanie dalej szczegółowo omówiona.

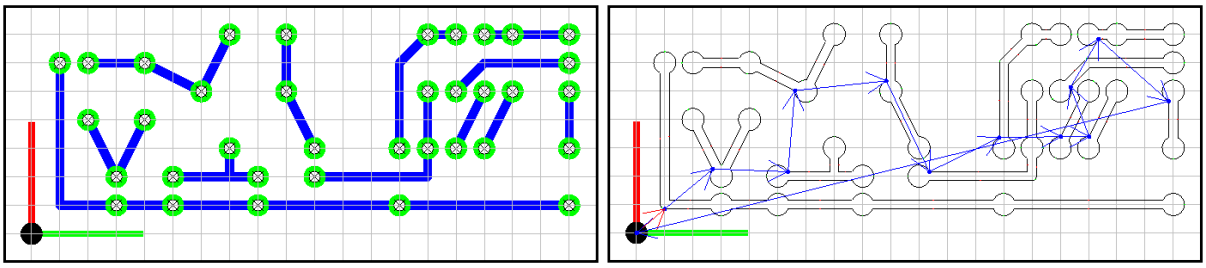
4.2. Poszukiwanie losowe

4.2.1. Podejście czysto losowe

Najprostszą metodą poszukiwania rozwiązanie jest losowe generowanie rozwiązań próbnych. W przypadku zadań kombinatorycznych, gdzie problem posiada skończoną liczbę kombinacji, a rozmiar zadania nie jest zbyt duży, istnieje pewne prawdopodobieństwo znalezienia rozwiązania. Im większa liczba prób tym większe szanse na uzyskanie dobrego wyniku. Dla zadania typu TSPN należy jednocześnie losować kolejność odwiedzanych regionów oraz losowo dobierać punkty należące do nich.

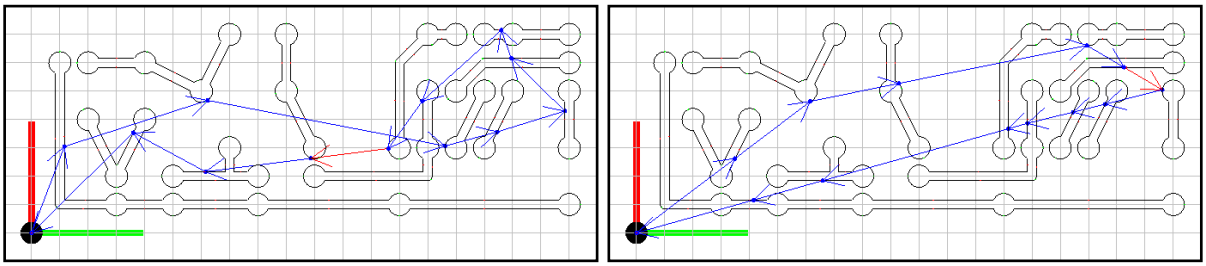
Do walidacji metod losowych użyto pewnego prostego modelu zawierającego jedynie 12 konturów. Rysunek 4.2 przedstawia model oraz postać rozwiązania uzyskiwaną bez optymalizacji w poprzedniej wersji programu *PCB CAM Processor*. Poszukiwanie losowe polega na wygenerowaniu określonej liczby rozwiązań próbnych i wyborze najlepszego

z nich. Wykonano po 100 powtórzeń algorytmu dla trzech różnych liczb iteracji: 10 tys., 100 tys., 1 mln. Wyniki zestawiono w tabeli 4.1.



Rys. 4.2. Prosty model PCB wraz z wynikiem uzyskanym metodą NN

Rysunek 4.3 przestawia najlepszy wynik uzyskany metodą losową zestawiony z rozwiązaniem niemal optymalnym obliczonym metodą *3-opt* + RBA, która zostanie dalej szczegółowo omówiona.



Rys. 4.3. Wynik metody losowej oraz *3-opt* + RBA

Tabela 4.1 zawiera zestawienie wyników metody losowej dla różnych liczb iteracji. Obok minimalnej, maksymalnej oraz średniej wartości ze 100 prób podano również błąd procentowy względem najlepszego uzyskanego wyniku (50.9 mm obliczonego w 42 ms) metodą *3-opt* + RBA - rys. 4.3.

Tabela 4.1. Zestawienie wyników metody losowej

Liczba iteracji	Długość ścieżki	Średni Czas	Błąd
10 tys.	61.4 mm - min 77.9 mm - max 71.2 mm - średnia	15 ms	20.63% 53.05% 39.88%
100 tys.	59.2 mm - min 71.0 mm - max 66.3 mm - średnia	138 ms	16.31% 39.49% 30.26%
1 mln.	57.8 mm - min 65.7 mm - max 62.1 mm - średnia	1375 ms	13.56% 29.08% 22.00%

4.2.2. Losowanie sekwencji oraz RBA

Lepszym podejściem, niż prezentowane wcześniej poszukiwanie całkowicie losowe, jest połączenie losowego poszukiwania sekwencji z obliczaniem zadania TPP metodą RBA. Takie podejście gwarantuje, że jeśli zostanie znaleziona prawidłowa sekwencja, to zadanie zostanie dalej rozwiązane znaną techniką obliczeniową. Losowaniu podlega jedynie sekwencja, co zwiększa prawdopodobieństwo znalezienia rozwiązania. Na kolejnych listingach zaprezentowano implementację metody RBA.

Listing 4.1. Procedura *OptRBA* - Metoda dwupodziału

```
Input:  P1, P2 - ustalone punkty w sąsiednich konturach
Output: optymalny punkt w bieżącym konturze

Index = n /2
Step  = n /4

While ( Step > 0 )
{
    Iplus  = Index + Step
    Iminus = Index - Step

    Dist1 = Dist( P1, Points[Iplus] )  + Dist( P2, Points[Iplus] )
    Dist2 = Dist( P1, Points[Iminus] ) + Dist( P2, Points[Iminus] )

    if ( Dist1 < Dist2 ) Index += Step
    else                 Index -= Step

    Step /= 2
}

return Points[Index]
```

Na potrzeby szybkich obliczeń TPP, kontury zostają wstępnie poddane dyskretyzacji. Wyznaczana jest pewna określona liczba punktów n ($n = 2^x$) rozłożonych równomiernie wzdłuż całego konturu. Zmienna *Index* przechowuje aktualny indeks tabeli punktów *Points*, natomiast *Step* stanowi krok poszukiwań wyrażony jako przyrost indeksu tablicy punktów. Algorytm działa wewnątrz pętli *while*, która kończy się, kiedy długość kroku poszukiwań spadnie do zera. W każdej kolejnej iteracji obliczane są odległości pomiędzy wybranym punktem konturu, a punktami *P1* oraz *P2*, znajdującymi się w sąsiednich konturach. Zmienne *Dist1* oraz *Dist2* wyrażają sumy odległości (rys. 4.1) dla obu sprawdzanych wariantów kierunku poszukiwań: dla indeksu powiększonego (*Iplus*) oraz pomniejszonego (*Iminus*) o długość kroku względem bieżącej pozycji w tablicy punktów. W kolejnym etapie wybierany jest korzystniejszy kierunek, a krok zmniejszany jest o połowę. Metoda bazuje na stosowanej dla funkcji, metodzie dwupodziału i podobnie znajduje jedno rozwiązanie, będące lokalnym

minimum. Jest jednak bardzo szybka. Dla podziału na 1024 punkty wymaga jedynie sprawdzenia wykonania 10 iteracji, by znaleźć rozwiązanie.

Metoda RBA zakłada obliczenie punktów dla wszystkich konturów znajdujących się w ustalonej wcześniej sekwencji. Schemat postępowania dla pojedynczego cyklu metody pokazano na listingu 4.2.

Listing 4.2. Procedura *OneRBA* - pojedynczy cykl metody RBA

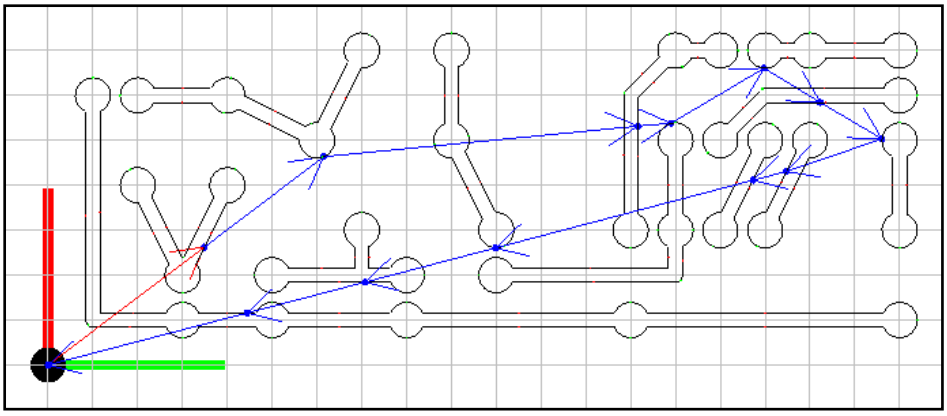
```
LastI = n-1

Contours[0].OptRBA( Contours[LastI].P, Contours[1].P )

for ( i = 1; i < LastI; i++ )
    Contours[i].OptRBA( Contours[i-1].P, Contours[i+1].P )

Contours[LastI].OptRBA( Contours[0].P, Contours[LastI-1].P )
```

Tablica *Contours* zawiera kolejne kontury, których wybrane punkty *P* tworzą ścieżkę stanowiącą rozwiązanie zadania TSPN. Dla każdego konturu z tablicy wywoływana jest opisana wcześniej procedura *OptRBA*, przyjmująca za argumenty punkty *P* sąsiednich konturów.

Listing 4.3. Algorytm metody RBA

```
do
{
    Len = Len( Contours )
    OneRBA( Contours )
    NewLen = Len( Contours )
}
while ( Len - NewLen > Eps )
```

Pełne obliczenie wymaga wielokrotnego uruchomienia procedury *OneRBA*, aż do uzyskania ścieżki, której długość jest mniejsza od obliczonej w poprzednim cyklu o mniej niż pewien akceptowalny błąd *Eps*.



Rys. 4.4. Wynik metody RND + RBA

Do walidacji metody RND + RBA użyto takiego samego modelu jak dla wcześniejszej metody. Wykonano po 100 prób dla trzech liczb iteracji: 1 tys., 10 tys. oraz 100 tys. W tabeli 4.2 zestawiono wyniki poszczególnych prób. Rysunek 4.4 przedstawia najlepszy z uzyskanych wyników.

Tabela 4.2. Zestawienie wyników metody RND +RBA

Liczba iteracji	Długość ścieżki	Średni Czas	Błąd
1 tys.	51.8 mm - min 64.0 mm - max 57.3 mm - średnia	45 ms	1.77% 25.74% 12.57%
10 tys.	51.0 mm - min 56.9 mm - max 54.0 mm - średnia	442 ms	0.20% 11.79% 6.09%
100 tys.	51.1 mm - min 52.8 mm - max 52.1 mm - średnia	4451 ms	0.39% 3.73% 2.36%

Wnioski:

- Metody losowe nadają się do rozwiązywania bardzo prostych zadań. Przy braku znajomości innych, lepszych metod, mogą być brane pod uwagę.
- Im wyższa liczba iteracji algorytmu, tym wyższe prawdopodobieństwo osiągnięcia lepszego wyniku oraz naturalnie dłuższy czas obliczeń.
- Czysto losowe poszukiwanie pozwala uzyskać wynik na poziomie 57.8 mm, czyli 13.56% błędu względem najlepszego możliwego rozwiązania programu. Wynik jest lepszy, niż rozwiązanie uzyskiwane przez poprzednią wersję oprogramowania, które obliczane jest metodą NN i wynosi 61.4 mm (błąd: 20.63%).
- Metoda RND + RBA jest lepsza od podejścia czysto losowego. Pozwala uzyskać niemal optymalne wyniki.

4.3. Heurystyki lokalnych poszukiwań

Przedstawione w tym rozdziale metody bazują na opisanych w rozdziale 3 metodach poszukiwań *k-optymalnych*, które okazują się być bardzo dobrym podejściem do rozwiązywania zadań kombinatorycznych.

4.3.1. Poszukiwanie *2-opt*

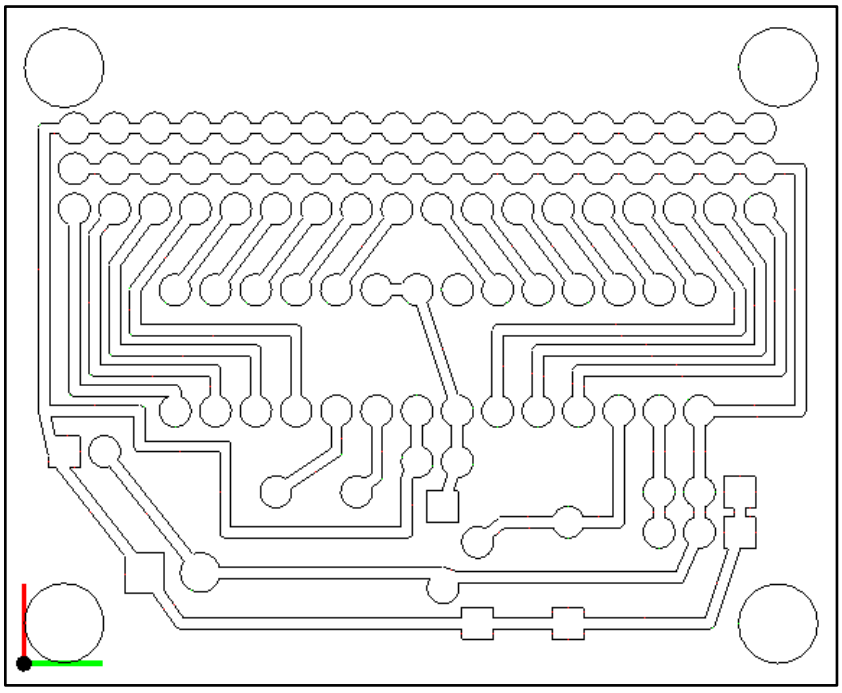
W przypadku zadania TSPN metoda *2-opt* służy do poszukiwanie sekwencji konturów, która po znalezieniu ścieżki metodą RBA jest poddawana ocenie. Dla zadania TSP istniała możliwość obliczenia różnicy między bieżącym rozwiązaniem, a proponowanym, wymagająca jedynie odjęcia długości zastępowanych krawędzi i dodania długości nowych. Dla zadania TSPN ocena rozwiązania jest możliwa dopiero po pełnej konstrukcji rozwiązania, obejmującej wyznaczenie nowej sekwencji oraz obliczenie RBA z zadanym błędem dopuszczalnym. Takie postępowanie znacznie obniża efektywność metody, dlatego postanowiono wprowadzić pewne ulepszenie. Podczas konstrukcji rozwiązania, tj. dodawania kolejnych konturów do listy, wywoływana jest procedura *OptRBA*. Po dodaniu nowego konturu można już obliczyć punkt dla konturu dodanego w poprzednim kroku. Równocześnie obliczana jest długość powstającej ścieżki, która obejmuje jedynie odległości między ustalonymi już punktami. Jeżeli obliczana długość, przekroczy wartość najlepszego rozwiązania, to konstrukcja rozwiązania jest przerywana. W przeciwnym wypadku, rozwiązanie zostaje przeliczone dokładniej poprzez wykonanie RBA dla kompletnego, zamkniętego cyklu. Jeśli okaże się lepsze, to zostaje zapamiętane.

Listing 4.4. Procedura *NewTour*

```
if ( !AddContoursAndRBA() )          // Konstrukcja nowego rozwiązania
{
    return false
}
else
{
    RBA( CandSol, Eps )              // Obliczenie RBA dla nowego rozw.

    if ( CandSol.Len < BestSol.Len ) // Jeśli nowe rozw. jest lepsze
    {
        BestSol = CandSol            // Przyjęcie nowego rozwiązania
        return true
    }
    else return false
}
```

Ogólna zasada metody *2-opt* pozostaje bez zmian względem prezentowanej w rozdziale 3. Jedyną różnicą w implementacji jest procedura konstrukcji rozwiązania *NewTour*, której pseudokod zaprezentowano na listingu 4.4. Za konstrukcje rozwiązania próbnego odpowiada procedura *AddContoursAndRBA*. Na podstawie rozwiązania *BestSol* oraz indeksów podziału cyklu, wytypowanych przez poszukiwanie *2-opt* (rys. 3.17), tworzy ona nowe rozwiązanie. Jak opisano wcześniej procedura kończy prace z wynikiem *false*, kiedy długość konstruowanej ścieżki przekroczy wartość najlepszego rozwiązania, natomiast zwraca *true*, kiedy uda się uzyskać obiecujące rozwiązanie. Procedura *NewTour* zwraca *true*, kiedy uda się skonstruować pełne rozwiązanie, które po obliczeniu RBA będzie lepsze od poprzedniego. Odpowiedź procedury steruje przebiegiem dalszych poszukiwań metody *2-opt*. Bardziej szczegółowy opis metody *2-opt* znajduje się w rozdziale 3.4.

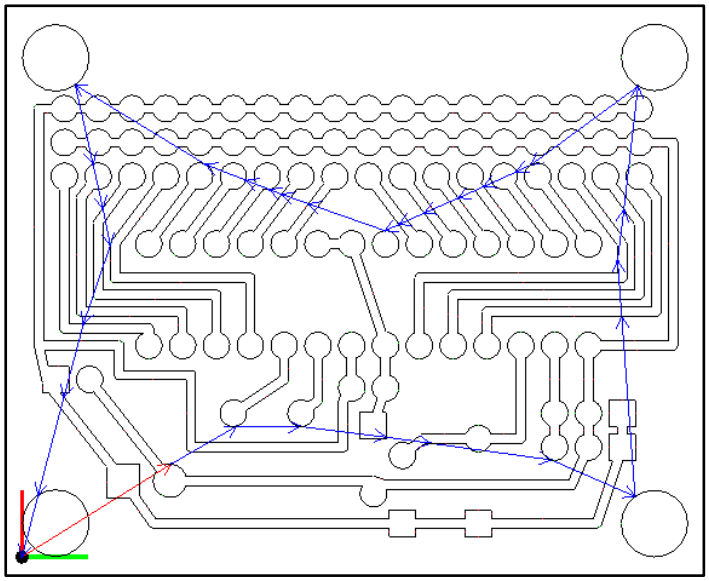
Walidację metody *2-opt* + RBA przeprowadzono dla zadania grawerowania układu ścieżek modelu płyty PCB, który był już prezentowany w rozdziale 3. Model zawiera 30 konturów. Rysunek 4.5 przedstawia postać zadnia grawerowania.



Rys. 4.5. Zadanie grawerowania konturów płyty PCB

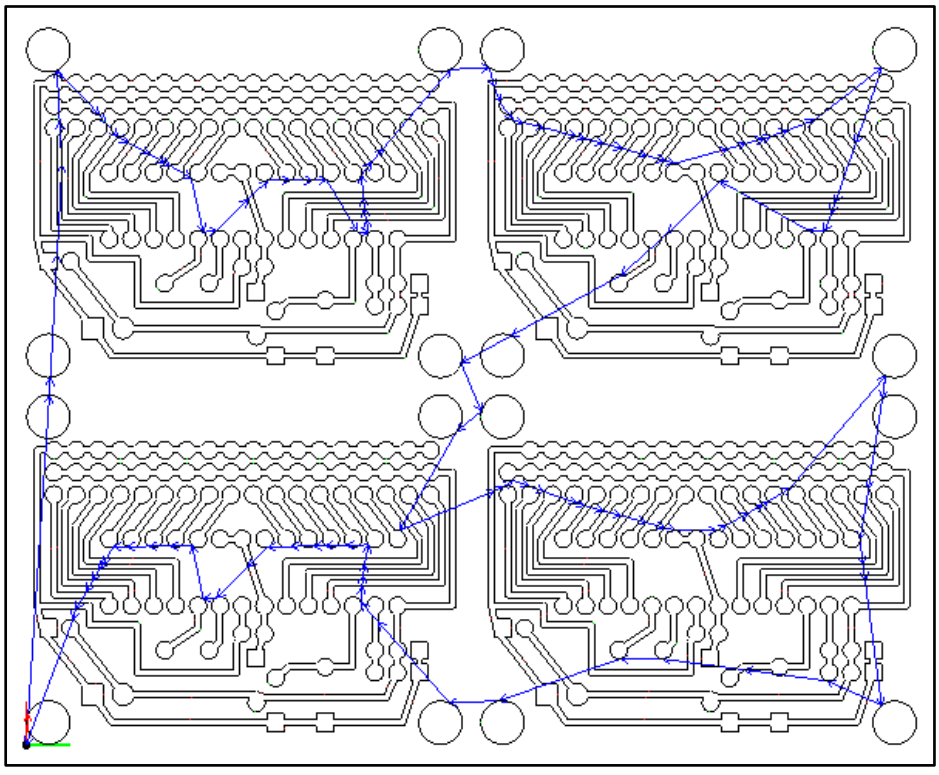
Podobnie jak dla zadania TSP, istnieją różne możliwości uzyskania rozwiązania początkowego dla metody *2-opt*. Dla zadania grawerowania konturów dostępne są rozwiązania wygenerowane losowo lub metodą najbliższego sąsiedztwa - NN. Dla metody losowej wykonano 20 powtórzeń i wyciągnięto średnią. Wyniki zaprezentowano w tabeli 4.3.

Na rysunku 4.6 zaprezentowano wynik obliczania ścieżki dla metody *2-opt* + RBA z rozwiązaniem początkowym uzyskanym metodą NN. Długość uzyskanej ścieżki wynosi 165.0 mm i została obliczona w czasie 249 ms.



Rys. 4.6. Wynik metody *2-opt* + RBA

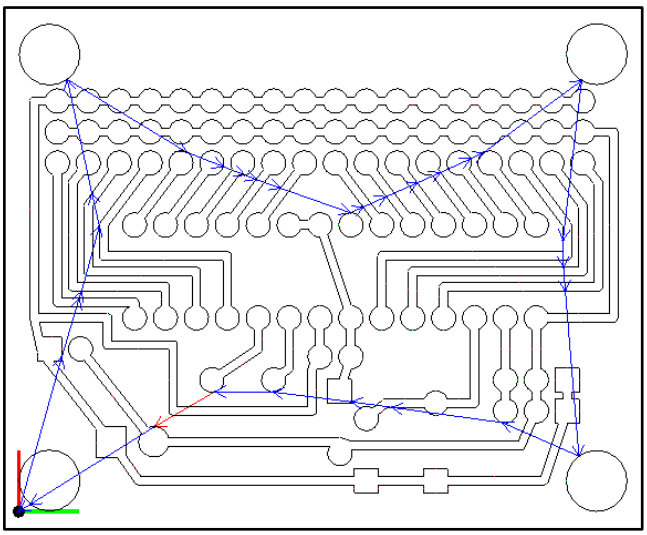
Kolejnym zadaniem wykorzystanym w obliczeniach był model zawierający 4 płyty PCB ułożone obok siebie, czyli łącznie 120 konturów. Długość ścieżki wyniosła 538.0 mm i została obliczona w 46.6 s. Wynik obliczeń przedstawiono na rysunku 4.7.



Rys. 4.7. Wynik metody *2-opt* + RBA

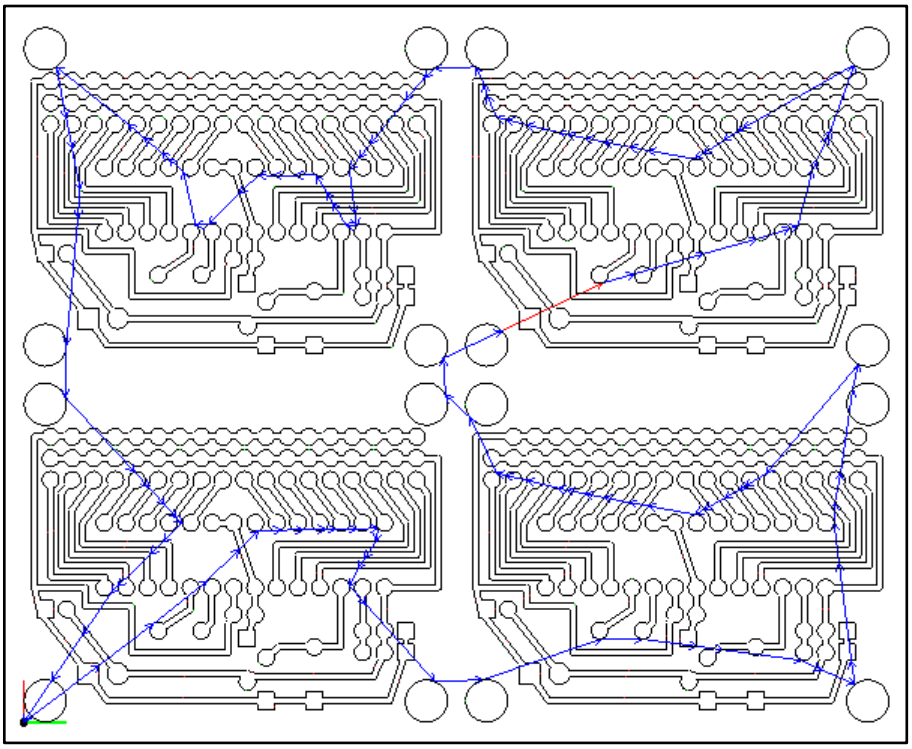
4.3.2. Poszukiwanie *3-opt*

Walidację metody *3-opt* przeprowadzono dla tych samych problemów, co dla metody *2-opt*. Metoda jest bliźniaczo podobna do *2-opt*, a ogólny opis został poruszony w rozdziale 3. Rysunek 4.8 przedstawia wynik obliczenia zadania grawerowania płyty PCB metodą *3-opt*. Uzyskana w czasie 2.5 s ścieżka ma długość 165.0 mm.



Rys. 4.8. Wynik metody *3-opt* + RBA

Dla zadania czterech płyt PCB uzyskano wynik 515.5 mm w czasie 523.4 s. Rysunek 4.9 przedstawia przebieg wyznaczonej ścieżki.



Rys. 4.9. Wynik metody *3-opt* + RBA

W tabeli 4.3 zestawiono wyniki obliczeń dla metod *2-opt* oraz *3-opt* dla obu prezentowanych zadań. Niestety nie są znane rozwiązania optymalne, dlatego nie udało się wyznaczyć błędów metod, a jedynie porównać ich wyniki. Dla zadania czterech płyt PCB nie wykorzystywano losowego rozwiązania początkowego z uwagi na bardzo długi czas obliczeń.

Tabela 4.3. Zestawienie wyników metod *2-opt* oraz *3-opt*

Zadanie	Metoda	Liczba iteracji	Długość ścieżki	Czas
Płyta PCB	RND *2-opt*	20	165.0 mm - min 179.6 mm - max 170.2 mm - średnia	992 ms 730 ms 903 ms
	NN *2-opt*	1	165.0 mm	249 ms
	RND *3-opt*	20	165.0 mm - min 171.3 mm - max 165.9 mm - średnia	2437 ms 2513 ms 3387 ms
	NN *3-opt*	1	165.0 mm	2495 ms
4 x Płyta PCB	NN *2-opt*	1	538.0 mm	46.6 s
	NN *3-opt*	1	512.5 mm	523.4 s

Wnioski:

- Analiza wyników wykazuje, że podobnie jak dla zadania TSP, postać rozwiązania początkowego ma wpływ na wynik końcowy. Widać też wyraźne wydłużenie czasu obliczeń w przypadku losowych rozwiązań początkowych. Ma to związek z większą liczbą poprawek, jaką należy wykonać w porównaniu z rozwiązaniem NN, które ma znacznie lepszą jakość.
- Metoda *2-opt* jest znacznie szybsza niż *3-opt*, ale dla większych zadań uzyskuje gorsze wyniki. W prezentowanym przykładzie (4 x Płyta PCB), wzrost nakładu obliczeniowego dla metody *3-opt* wydaje się być nieadekwatny w stosunku do polepszenia rozwiązania.
- Zastosowanie optymalizacji zadania grawerowania przyniosło znaczne skrócenie ścieżki w obu prezentowanych zadaniach. Poprzednia wersja *PCB CAM Processor*, stosując metodę NN, uzyskiwała odpowiednio 244.0 mm dla płyty PCB oraz 802.5 mm dla modelu czterech płyt PCB.

4.4. Podejście genetyczne

Podobnie jak dla metod *k-optymalnych*, zadanie TSPN zostaje zdekomponowane na: TSP oraz TPP. Algorytm GA służy do rozwiązywania części TSP, natomiast algorytm RBA rozwiązuje zadanie TPP.

4.4.1. Standardowy algorytm genetyczny

Implementacja GA jest identyczna jak dla zadania TSP, dlatego badania nastaw algorytmu ograniczono. Na podstawie wyników GA dla TSP wytypowano pewne obiecujące zakresy zmian parametrów algorytmu. Z uwagi na znacznie większy nakład obliczeniowy związany z obliczeniem funkcji przystosowania (patrz: metoda RBA), zmniejszono pulę rodzicielską do 100. Algorytm przebadano dla zadania płyty PCB przy następujących parametrach:

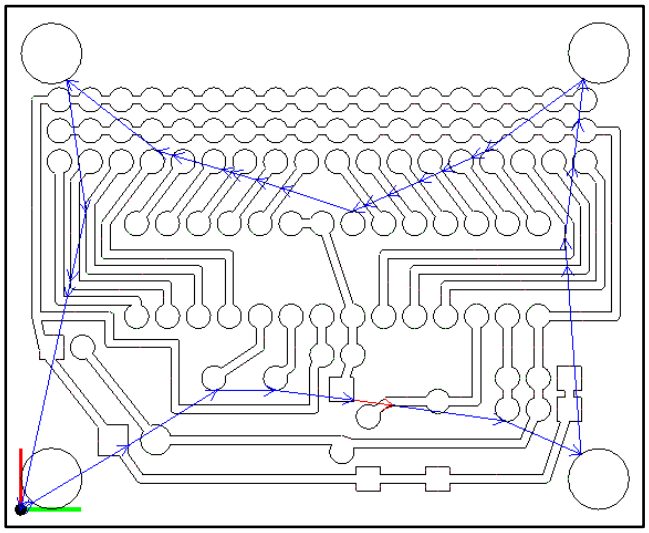
- Wielkość puli rodzicielskiej: 100
- Liczba krzyżowań pary (selekcja): 3, 5, 10
- Prawdopodobieństwo mutacji RSM: 0.1, 0.3, 0.5, 0.7
- Elitaryzm: tak
- Liczba iteracji algorytmu: wystarczająca do ustabilizowania populacji lub ograniczona czasowo.
- Rozwiązanie początkowe uzyskane metodą NN.

Wykonano po 10 prób dla każdego z możliwych zestawów parametrów. Średnie z uzyskanych wyników zaprezentowano w tabeli 4.4.

Tabela 4.4. Dobór parametrów algorytmu GA dla zadania grawerowania płyty PCB

Wielkość puli rodzicielskiej	Selekcja	Współczynnik mutacji			
		0.1	0.3	0.5	0.7
100	3 (400 it.)	166.2 mm 7.2 s	165.7 mm 10.0 s	165.1 mm 12.9 s	175.3 mm 18.9 s
	5 (200 it.)	165.7 mm 6.2 s	165.4 mm 8.4 s	165.1 mm 10.6 s	165.1 mm 12.4 s
	10 (100 it.)	165.0 mm 6.7 s	**164.9 mm 8.9 s**	165.0 mm 10.1 s	165.0 mm 12.5 s

Analiza wyników przedstawionych w tabeli 4.4 wykazuje, że najlepsze rezultaty osiąga się dla selekcji na poziomie 10. Wyższy współczynnik mutacji w tej grupie wyników, wpływa na wydłużenie czasu obliczeń, natomiast nie poprawia rozwiązania. Można przyjąć, że selekcja równa 10 oraz prawdopodobieństwo mutacji równe 0.3 są najlepszymi nastawami GA dla zadania płyty PCB. Ponadto czas obliczeń można zredukować nawet dwukrotnie, ponieważ w większości prób osiągnięto stabilizację populacji między 30, a 50 iteracją. Przy proponowanych nastawach, w zaledwie 50 iteracjach, udało się uzyskać wynik 164.7 mm w czasie 5.0 s. Rysunek 4.10 przedstawia najlepszy uzyskane dotąd rozwiązanie zadania grawerowanie płyty PCB.



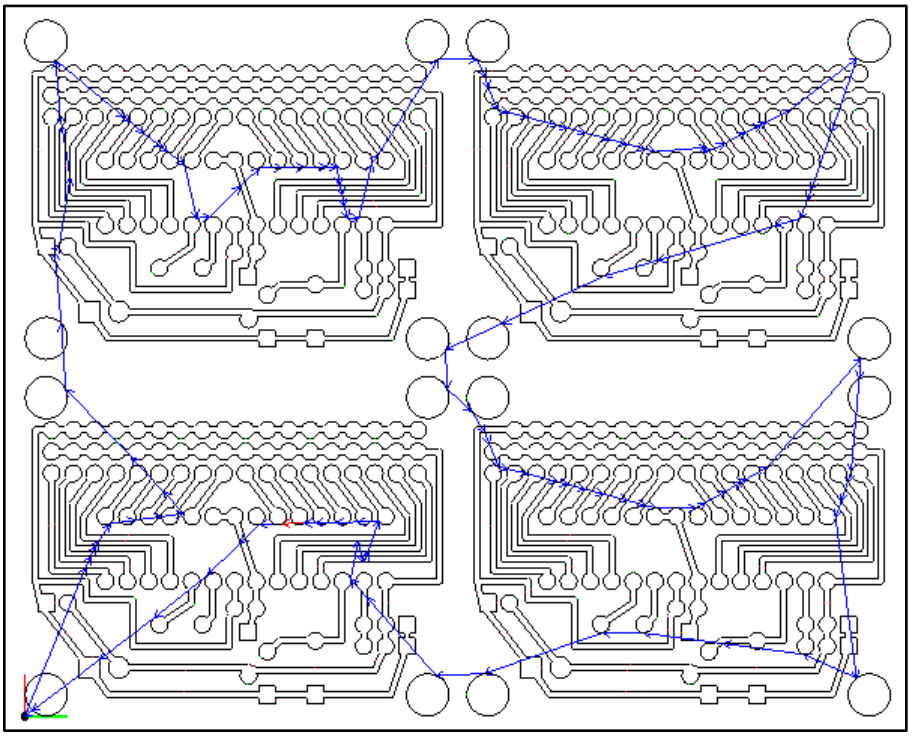
Rys. 4.10. Najlepszy wynik metody GA + RBA (164.7 mm)

Analizując wyniki uzyskane dla zadania płyty PCB ustalono pewien zbiór nastaw algorytmu dla zadania czterech płyt PCB. Dla każdego z wybranych zestawów parametrów wykonano po 10 prób i wyciągnięto średnie. W tabeli 4.5 zestawiono wyniki wszystkich prób.

Tabela 4.5. Dobór parametrów algorytmu GA dla zadania czterech płyt PCB

Wielkość puli rodzicielskiej	Selekcja	Współczynnik mutacji	
		0.3	0.5
100	10 (200 it.)	533.7 mm (73.0 s) - min 584.3 mm (69.7 s) - max 558.9 mm (68.5 s) - średnia	539.3 mm (80.2 s) - min 570.5 mm (99.4 s) - max 556.6 mm (81.9 s) - średnia
	20 (100 it.)	527.7 mm (71.0 s) - min 585.3 mm (68.4 s) - max **551.7 mm (71.0 s) - średnia**	**522.4 mm (89.9 s) - min** 594.5 mm (81.1 s) - max 555.0 mm (84.6 s) - średnia

Z uwagi na czterokrotnie większy rozmiar zadania zdecydowano zwiększyć liczbę iteracji oraz wprowadzić wyższą selekcję, aby poprawić zbieżność algorytmu. Ponownie widoczne jest wydłużenie czasu obliczeń spowodowane większym współczynnikiem mutacji. Wynika to z faktu, iż metoda RBA musi wykonać więcej iteracji dla rozwiązań reprezentowanych przez osobniki po mutacji. Na rysunku 4.11 przedstawiono najlepszy wynik dla zadania czterech płyt PCB uzyskany metodą GA. Rozwiązanie o długości ścieżki 522.4 mm uzyskano w 89.9 s. W odniesieniu do wyniku metody *3-opt* (512.5 mm uzyskanego w 523.4 s), jest to bardzo dobry rezultat.



Rys. 4.11. Najlepszy wynik metody GA + RBA (522.4 mm)

4.4.2. Algorytm hybrydowy

Podobnie jak dla zadania TSP, wprowadzono metodę *2-opt* jako dodatkowy operator mutacji w algorytmie. Z uwagi na konieczność obliczania ścieżki metodą RBA dla każdego przestawienia *2-opt*, zdecydowano ograniczyć liczbę sprawdzeń metody do 1000. Oznacza to, że mutacja może poprawić dane rozwiązanie pewną skończoną liczbę razy. W najgorszym wypadku nie poprawi go wcale. Dla zadania TSP mutacja *2-opt* obliczana była do końca, to znaczy do maksymalnego polepszenia wyniku.

Walidację algorytmu hybrydowego przeprowadzono dla tych samych zadań. Na podstawie wcześniejszych obserwacji wytypowano pewne wartości nastaw algorytmu i dla

każdego ich zestawu wykonano po 20 prób oraz wyciągnięto średnie. Dla zadania płyty PCB zastosowano następujące parametry:

- Wielkość puli rodzicielskiej: 50
- Liczba krzyżowań pary (selekcja): 5, 10
- Prawdopodobieństwo mutacji RSM: 0.3
- Elitaryzm: tak
- Prawdopodobieństwo mutacji *2-opt*: 0.1, 0.3, 0.5.

Większa złożoność obliczeniowa metody hybrydowej wymusza redukcję populacji. Celem stosowania hybrydyzacji jest uzyskanie bardziej efektywnej metody. W tabeli 4.6 zestawiono wyniki wszystkich prób.

Tabela 4.6. Dobór parametrów algorytmu GA + *2-opt* dla zadania grawerowania płyty PCB

Wielkość puli rodzicielskiej	Selekcja	Współczynnik mutacji *2-opt*		
		0.1	0.3	0.5
50	5 (100 it.)	166.8 mm 2.6 s	166.5 mm 3.0 s	165.6 mm 3.5 s
	10 (50 it.)	165.9 mm 2.8 s	**165.6 mm** **3.1 s**	165.9 mm 3.6 s

Walidację zadania czterech płyt PCB przeprowadzono dla większej populacji oraz wykonano większą liczbę iteracji algorytmu niż dla mniejszego zadania. Wykonano po 10 prób dla każdego z typowanych zestawów parametrów. Wyniki zestawiono w tabeli 4.7.

Tabela 4.7. Dobór parametrów algorytmu GA + *2-opt* dla zadania czterech płyt PCB

Wielkość puli rodzicielskiej	Selekcja	Współczynnik mutacji *2-opt*		
		0.1	0.3	0.5
50	10 (200 it.)	544.2 mm (37.0 s) - min 592.9 mm (36.0 s) - max 569.7 mm (37.0 s) - śr.	547.8 mm (43.9 s) - min 578.7 mm (41.5 s) - max 564.5 mm (43.4 s) - śr.	542.4 mm (58.1 s) - min 597.6 mm (48.8 s) - max 565.7 mm (51.7 s) - śr.
	20 (100 it.)	537.5 mm (37.6 s) - min 598.8 mm (38.0 s) - max **558.8 mm (37.6 s) - śr.**	535.3 mm (47.3 s) - min 579.3 mm (45.3 s) - max 564.2 mm (46.5 s) - śr.	534.6 mm (66.8 s) - min 576.4 mm (62.5 s) - max 562.7 mm (56.6 s) - śr.
100	20 (100 it.)	530.3 mm (77.0 s) - min 571.1 mm (75.8 s) - max **547.7 mm (81.3 s) - śr.**	**522.0 mm (88.2 s) - min** 561.5 mm (90.0 s) - max 547.7 mm (90.1 s) - śr.	527.9 mm (102.1 s) - min 589.2 mm (105.7 s) - max 547.9 mm (104.7 s) - śr.

Wnioski:

- Analiza wyników wykazuje, że algorytmu genetyczne potrafią znaleźć dobre rozwiązanie w zadowalającym czasie. Dla mniejszego zadanie udało się znaleźć lepsze rozwiązanie niż dla metody *3-opt* (164.7 mm, zamiast 165.0 mm).
- Zastosowanie hybrydyzacji poprawia zbieżność algorytmu, ale wydłuża czas obliczeń. Dla większych zadań istnieje możliwość uzyskania nieco lepszych wyników w podobnym czasie.
- Dla zadania czterech płyt PCB, stosując hybrydyzację, uzyskano lepsze średnie rozwiązania oraz mniejszy rozrzut wyników, przy tych samych ustawieniach algorytmu, co dla standardowego GA.
- Stosowanie współczynnika mutacji *2-opt* większego niż 0.1 nie poprawia już wyniku, jedynie wydłuża czas obliczeń.

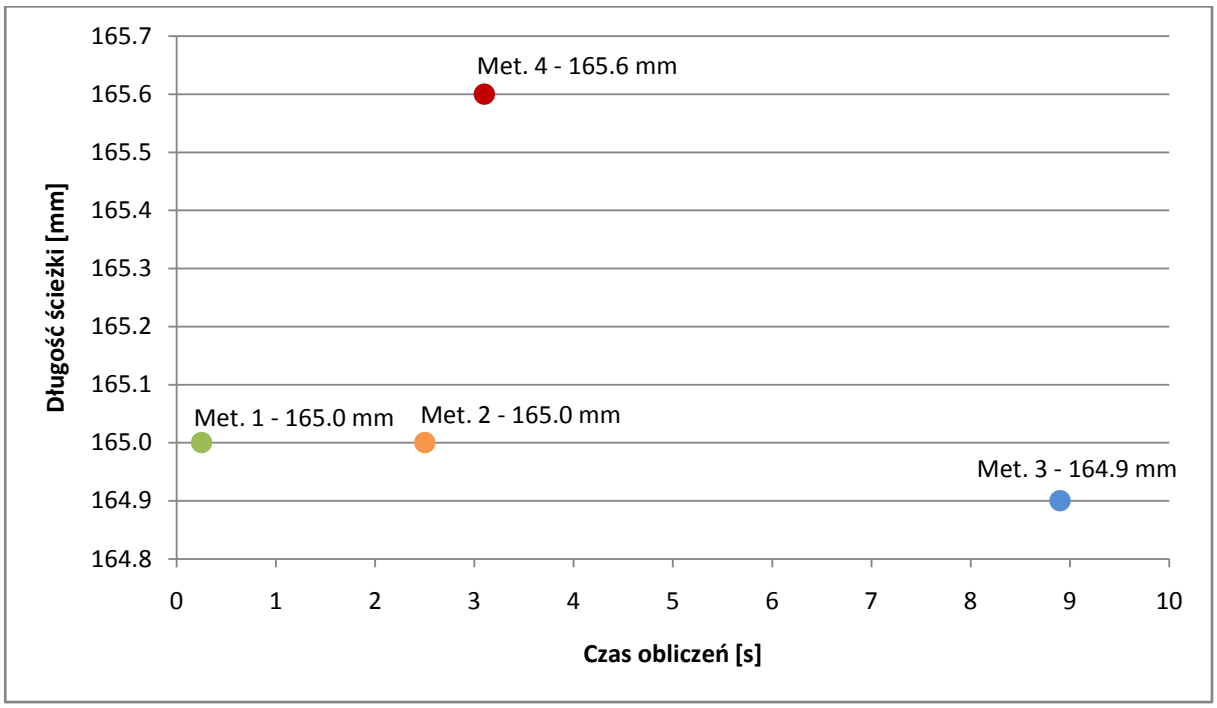
4.5. Porównanie zaimplementowanych metod - wyniki

W tabeli 4.8 zestawiono najlepsze rezultaty metod poszukiwań lokalnych oraz metod genetycznych dla obu prezentowanych zadań.

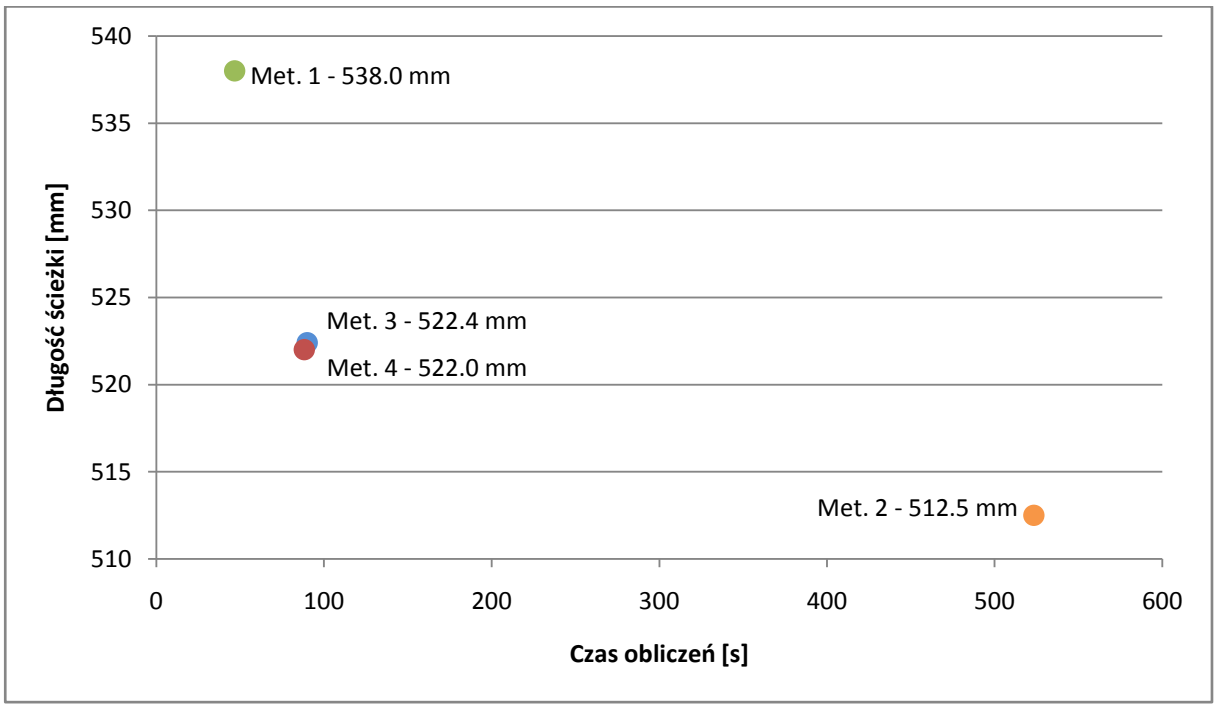
Tabela 4.8. Zestawienie najlepszych wyników wszystkich prezentowanych metod

Zadanie	Metoda		Długość ścieżki	Czas
	Lp.	Nazwa		
Płyta PCB	1	NN *2-opt* + RBA	165.0 mm	0.25 s
	2	NN *3-opt* + RBA	165.0 mm	2.5 s
	3	GA + RBA	164.9 mm	8.9 s
	4	GA + *2-opt* + RBA	165.6 mm	3.1 s
4 x Płyta PCB	1	NN *2-opt* + RBA	538.0 mm	46.6 s
	2	NN *3-opt* + RBA	512.5 mm	523.4 s
	3	GA + RBA	522.4 mm	89.9 s
	4	GA + *2-opt* + RBA	522.0 mm	88.2 s

Dla zadania płyty PCB przedstawiono najlepsze średnie wyniki metod genetycznych, natomiast dla czterech płyt PCB, najlepsze przypadki uzyskane podczas wszystkich prób. Pełne wyniki znajdują się w tabelach od 4.4 do 4.7. Wykresy 4.1 oraz 4.2 stanowią graficzne porównanie efektywności prezentowanych metod.



Wyk. 4.1. Porównanie wyników dla zadania płyty PCB



Wyk. 4.2. Porównanie wyników dla zadania czterech płyt PCB

4.6. Podsumowanie i wnioski

Optymalizacja zadania grawerowania przynosi wyraźne skrócenie ścieżki narzędzia. Poprzednia wersja oprogramowania *PCB CAM Processor* uzyskiwała odpowiednio 244.0 mm dla płyty PCB oraz 802.5 mm dla modelu czterech płyt PCB. Zarówno metody poszukiwań lokalnych jak i metody genetyczne, z powodzeniem uzyskują znacznie lepsze rezultaty.

Metody losowe nie znajdują się w podsumowaniu, gdyż nie są w stanie osiągać zadowalających wyników dla tak dużych zadań.

Dla mniejszych zadań zdecydowanie najlepsze okazują się metody poszukiwań lokalnych. Algorytmom genetycznym trudno z nimi konkurować. Natomiast wraz ze wzrostem skomplikowania zadania, GA wychodzą na prowadzenie. Istnieje możliwość takiego doboru parametrów, żeby uzyskać lepsze wyniki niż metoda *2-opt*. Przy zadaniu czterech płyt PCB widać przepaść między metodami *2-opt* oraz *3-opt*. Ta ostatnia staje się tu wręcz niemożliwa do zastosowania. GA są wstanie wypełnić lukę pomiędzy *2-opt* i *3-opt*, to znaczy uzyskiwać lepsze wyniki niż *2-opt* w czasie znacznie krótszym niż *3-opt*.

Hybrydyzacja nie przynosi tak dobrych rezultatów jak w przypadku zadania wiercenia, ponieważ stosowanie metody *2-opt* musi być ograniczone ze względu na złożoność obliczeniową dla zadania TSPN. Mimo to zastosowanie algorytmu hybrydowego podnosi efektywność GA.

Olbrzymią zaletą GA jest możliwość dowolnego dobierania parametrów algorytmu. Można w ten sposób wpływać na jakość rozwiązania. Jeśli dysponuje się dużą ilością czasu można wydłużyć obliczenia i uzyskać lepsze wyniki. Dla bardzo trudnych zadań można zadowolić się gorszym wynikiem, uzyskanym znacznie szybciej niż metodami poszukiwań lokalnych.

PODSUMOWANIE

Minimalizacja czasu obróbki jest kluczowym kryterium optymalizacji programów generowanych przez środowiska klasy CAM. Zastosowane w oprogramowaniu *PCB CAM Processor* metody optymalizacji spowodowały znacznie skrócenie ścieżki narzędzia dla prezentowanych zadań wiercenia i grawerowania w porównaniu z proponowanym wcześniej podejściem [31].

Jak wykazują przeprowadzone badania, najskuteczniejszym podejściem do rozwiązywania zadań kombinatorycznym są metody poszukiwań lokalnych. Wadą tych metod jest jednak wysokie ryzyko utknięcia w minimum lokalnym. Aby poprawić odporność takich metod można uruchamiać je dla różnych warunków początkowych. Postać rozwiązania początkowego jest bardzo istotna dla metod *2-opt* oraz *3-opt*.

Metody ewolucyjne są znacznie bardziej odporne, niestety czas potrzebny na obliczenie rozwiązań konkurujących z metodami poszukiwań lokalnych zajmuje znacznie więcej czasu. Podejścia hybrydowe są wstanie wprowadzić do algorytmów genetycznych pewną dodatkową wiedzę, która pozwala skierować poszukiwania rozwiązania w bardziej właściwym kierunku. Poprawia to znacząco zbieżność tych metod, jednakże szybko zbieżne metody są z natury mniej odporne.

Wybrane metody rozwiązywania zadań klasy TSP, zostały dobrane tak, aby wykazywać pewien uniwersalizm w kontekście zadań kombinatorycznych i mogły nadawać się również do rozwiązywania bardziej ogólnych zadań klasy TSPN. Wyznaczanie ścieżki prowadzącej poprzez kontury jest znacznie trudniejszym zadaniem niż klasyczne zadanie komiwojażera. Pojawiający się pewien dodatkowy stopień swobody, który związany jest z możliwością dowolnego doboru punktów początkowo-końcowych dla każdego z konturów, zwiększa złożoność obliczeniową problemu.

Wszystkie założenia pracy zostały spełnione, a zaimplementowane w oprogramowaniu metody, z powodzeniem nadają się do rozwiązywania pojawiających się problemów generowania ścieżek narzędzia. Dzięki możliwości wyboru z pośród wielu dostępnych metod, użytkownik może dopasować rozwiązanie do własnych oczekiwań. Istnieje możliwość uzyskiwania przybliżonych rozwiązań w bardzo krótkim czasie lub niemal optymalnych, które wymagają znacznie większego nakładu obliczeniowego.

BIBLIOGRAFIA

1. Abdoun O., Abouchabaka J.: *A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem*, International Journal of Computer Applications Vol. 31– No. 11, Listopad 2011.
2. Abu-Dakka F., Assad I., Alkhdour R., Abderahim M.: *Statistical evaluation of an evolutionary algorithm for minimum time trajectory planning problem for industrial robots*, International Journal of Advanced Manufacturing Technology, 2016.
3. Alatartsev S., Mersheeva V., Augustine M., Ortmeier F.: *On Optimizing a Sequence of Robotic Tasks*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Listopad 3-7, 2013, Tokio.
4. Alatartsev S., Stellmacher S., Ortmeier F.: *Robotic Task Sequencing Problem: A Survey*, Journal of Intelligent and Robotic System, 80, 2015, s. 279–298.
5. Anton F., Anton S., Răileanu S., Borangiu T.: *Optimizing trajectory points for high speed robot assembly operations*, Advances in Robot Design and Intelligent Control, 371, 2016, s. 127-135.
6. Applegate D. L., Bixby R. E., Chvátal V., Cook W. J.: *The Traveling Salesman Problem: A Computational Study*, Princeton University Press, USA 2006.
7. Arkin E. M., Hassin R.: *Approximation algorithms for the geometric covering salesman problem*, Discrete Applied Mathematics, vol. 55, 1995, s. 197–218.
8. Chiddarwar S. S., Babu N. R.: *Optimal trajectory planning for industrial robot along a specified path with payload constraint using trigonometric splines*, International Journal of Automation and Control, 6(1), 2012.
9. Choong S. S., Wong L., Lim C. P.: *An artificial bee colony algorithm with a Modified Choice Function for the traveling salesman problem*, Elsevier, Swarm and Evolutionary Computation 44 (2019), s. 622–635.
10. Constantinescu D., Croft E.: *Smooth and time-optimal trajectory planning for industrial manipulators along specified paths*, Journal of Robotic Systems, 17 (5), 2000, s. 233–249.
11. Dror M., Efrat A., Lubiw A., Mitchell J. S. B.: *Touring a sequence of polygons*, Artykuł na "35th ACM symposium on Theory of Computing", ACM Press, 2003, s. 473–482.

12. Erdős G., Kovács A., Váncza J.: *Optimized joint motion planning for redundant industrial robots,* CIRP Annals - Manufacturing Technology, 65, 2016, s. 451–454.

13. Fung R. F., Cheng Y. H.: *Trajectory planning based on minimum absolute input energy for an LCD glass-handling robot*, Applied Mathematical Modelling, 38, 2014, s. 2837–2847.

14. Gao X., Mu Y., Gao Y.: *Optimal trajectory planning for robotic manipulators using improved teaching-learning-based optimization algorithm*, Industrial Robot: An International Journal, 43(3), 2016, s. 308-316.

15. Gleeson D., Bjorkenstam S., Bohlin R., Carlson J. S., Lennartson B.: *Towards energy optimization using trajectory smoothing and automatic code generation for robotic assembly*, 6th Conference on Assembly Technologies and Systems (CATS), Procedia CIRP, vol. 44, 2016, s. 341-346.

16. Gregory J., Olivares A., Staffetti E.: *Energy-optimal trajectory planning for robot manipulators with holonomic constraints*, Systems & Control Letters, 61(2), Elsevier, 2012, s. 279–291.

17. Kim J., Croft E. A.: *Trajectory planning for robots: the challenges of industrial considerations, Motion Planning for Industrial Robots*, IEEE International Conference on Robotics and Automation (ICRA), 2014.

18. Kovács A.: *Integrated task sequencing and path planning for robotic remote laser welding*, International Journal of Production Research, 54(4), 2016, s. 1210–1224.

19. Lin H. I.: *A fast and unified method to find a minimum-jerk robot joint trajectory using particle swarm optimization*, Journal of Intelligent and Robotic System, 75, 2014, s. 379–392.

20. Liu H., Lai X., Wu W.: *Time-optimal and jerk-continuous trajectory planning for robot manipulators with kinematic constraints*, Robotics and Computer-Integrated Manufacturing, 29(2), 2013, s. 309–317.

21. Luo L. P., Yuan C., Yan R. J., Yuan Q., Wu J., Shin K. S., Han C. S.: *Trajectory planning for energy minimization of industry robotic manipulators using the Lagrange interpolation method*, International Journal of Precision Engineering and Manufacturing, 16(5), 2015, s. 911–917.

22. Mashimo T., Urakubo T., Kanade T.: *Singularity-based four-bar linkage mechanism for impulsive torque with high energy efficiency*, Journal of Mechanisms and Robotics, 7(3), 2015.

23. Matai R., Singh S., Mittal M.L.: *Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches*, Traveling Salesman Problem, Theory and Applications, Listopad 2010.

24. Meike D., Pellicciari M., Berselli G.: *Energy efficient use of multirobot production lines in the automotive industry: detailed system modeling and optimization*, IEEE Transactions on Automation Science and Engineering, 11(3), 2014, s. 798–809.

25. Mennell W.: *Heuristics for solving three routing problems: close-enough traveling salesman problem, close-enough vehicle routing problem, sequence-dependent team orienteering problem*. Rozprawa doktorska na Uniwersytecie Maryland, College Park 2009.

26. Pan X., Li F., Klette R.: *Approximate shortest path algorithms for sequences of pairwise disjoint simple polygons*, Artykuł konferencyjny, "Canadian Conference on Computational Geometry", 2010, s. 175–178.

27. Perumaal S., Jawahar N.: *Synchronized trigonometric curve trajectory for jerk-bounded time-optimal pick and place operation*, International Journal of Robotics and Automation, 27(4), 2012, s. 385-395.

28. Ratiu M., Prichici M. A.: *Industrial robot trajectory optimization - a review*, Artykuł konferencyjny, "MATEC Web of Conferences", Styczeń 2017.

29. Rego C., Glover F.: *Local Search and Metaheuristics for the Traveling Salesman Problem*, The Traveling Salesman Problem And Its Variations, Kluwer Academic Publishers, 2002, s. 309–368.

30. Reiter A., Springer K., Gattringer H., Müller A.: *An explicit approach for time-optimal trajectory planning for kinematically redundant serial robots*, PAMM Proceedings in Applied Mathematics and Mechanics, 15, 2015, s. 67–68.

31. Siwiec Radosław: *Opracowanie postprocesora dla generowania ścieżki narzędzia robota przemysłowego*. Praca inżynierska na Akademii Techniczno-Humanistycznej, Bielsko-Biała 2016.

32. Spensieri D., Carlson J. S., Bohlin R., Kressin J., Shi J.: *Optimal robot placement for tasks execution*, Artykuł konferencyjny "6th CIRP Conference on Assembly Technologies and Systems (CATS)", Procedia CIRP 44, 2016, s. 395 – 400.

33. Spong M. W., Hutchinson S., Vidyasagar M.: *Robot modeling and control*, Wyd. Wiley, Hoboken 2006.

34. Srivastava S., Kumar S., Garg R., Sen P.: *Generalized traveling salesman problem through n sets of nodes*, CORSE Journal, vol. 7, 1969, s. 97–101.

35. Sysło M., Deo N., Kowalik J.: *Algorytmy optymalizacji dyskretnej z programami w języku Pascal*, Wydawnictwo Naukowe PWN, Warszawa 1999.

36. Wang L.: *Redefining the shop floor - make it as cool as a computer game*, Atrykuł opublikowany na stronie: http://www.kth.se/en/forskning/artiklar/redefining-the-shop-floor-1.590274, 10.09.2015.

37. Wigstrom O., Lennartson B., Vergnano A., Breitholtz C.: *High level scheduling of energy optimal trajectories*, IEEE Transactions on Automation Science and Engineering, 10(1), 2013, s. 57–64.

38. https://www.kuka.com/pl

39. http://www.math.uwaterloo.ca/tsp/concorde.html

40. https://wwwproxy.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/