**Zastosowanie metaheurystyk do rozwiązywania problemów planowania projektów z wieloma wymaganymi umiejętnościami i ograniczonymi zasobami**

Spis treści

[Wstęp 4](#_Toc103439152)

[1. Problemy obliczeniowe w realizacji projektów 5](#_Toc103439153)

[1.1. Wprowadzenie do problemów obliczeniowych 5](#_Toc103439154)

[1.2. Problem spełnialności (Boolean satisfiability problem, SAT) 6](#_Toc103439155)

[1.3. Problem komiwojażera (travelling salesman problem, TSP) 6](#_Toc103439156)

[1.4. Programowanie nieliniowe (Nonlinear programming, NLP) 8](#_Toc103439157)

[1.5. Problem planowania projektu z wieloma wymaganymi umiejętnościami i ograniczonymi zasobami (Multi-Skill Resource-Constrained Project Scheduling Problem, MSRCPSP) 9](#_Toc103439158)

[1.6. Elementu problemu 11](#_Toc103439159)

[1.6.1. Model i wielkość przestrzeni poszukiwań 11](#_Toc103439160)

[1.6.2. Sąsiedztwo 12](#_Toc103439161)

[1.6.3. Funkcja oceny rozwiązania, cel i ograniczenia rozwiązania problemu 14](#_Toc103439162)

[2. Metody heurystyczne do rozwiązywania problemów 16](#_Toc103439163)

[2.1. Metaheurystyki w rozwiązywaniu problemów 16](#_Toc103439164)

[2.2. Algorytm zachłanny 16](#_Toc103439165)

[2.3. Przeszukiwanie lokalne 18](#_Toc103439166)

[2.4. Symulowane wyżarzanie 19](#_Toc103439167)

[2.5. Algorytm genetyczny 20](#_Toc103439168)

[3. Założenia realizacji badania 23](#_Toc103439169)

[3.1. Cel badania i zastosowana procedura badania 23](#_Toc103439170)

[3.2. Przyjęte założenia dla eksperymentu 23](#_Toc103439171)

[3.3. Opis sposobu realizacji projektu i badania 25](#_Toc103439172)

[4. Wyniki badań 26](#_Toc103439173)

[4.1. Otrzymane wyniki z przeprowadzonego badania 26](#_Toc103439174)

[4.2. Wizualizacja wyników badań 31](#_Toc103439175)

[4.3. Wnioski z przeprowadzonej analizy uzyskanych wyników 33](#_Toc103439176)

[Zakończenie 35](#_Toc103439177)

[Literatura 36](#_Toc103439178)

# Wstęp

# Problemy obliczeniowe w realizacji projektów

## Wprowadzenie do problemów obliczeniowych

Różne definicje projektów (np. Trockiego), z jakich części się te projekty składają.

W życiu napotyka się na wiele problemów, które są trudne do rozwiązania, zarówno dla człowieka, jak i dla komputerów. Powodów takiego stanu rzeczy może być kilka (Garey i Johnson, 1979), między innymi:

* Liczba potencjalnych rozwiązań dla danego problemu jest tak duża, że nie jest możliwe przeszukanie wszystkich rozwiązań w zadowalającym nas czasie. Niekiedy mogłoby to zająć miliony lat dla niektórych problemów, ale nawet jak zajmuję tylko parę minut, to może być zbyt długim oczekiwaniem – przykładowo przy wyznaczaniu nowej trasy dla nawigacji samochodowej.
* Problem jest tak skomplikowany, że modele, które używamy, są zbyt uproszczone, aby dać sensowny rezultat – przykładowo trudno jest przewidzieć pogodę na rok do przodu, gdyż jest tak wiele zmiennych, że jest to praktycznie niemożliwe do zrobienia przy zachowaniu jakiejkolwiek dokładności i idącej za tym użyteczności dla takiej prognozy.
* Ograniczenia, które są nałożone na rozwiązania są tak skomplikowane, że problematyczne jest w ogóle stworzenie takiego, które było by prawidłowe i nie łamało żadnych zasad – przykładowo tylko jeden klucz może odszyfrować poprawnie zaszyfrowane dane.

W kolejnych punktach tego rozdziału zostaną przedstawione trzy problemy: problem spełnialności, problem komiwojażera i programowanie nieliniowe. Natomiast w ostatnim, problem planowania projektu z wieloma wymaganymi umiejętnościami i ograniczonymi zasobami.

## Problem spełnialności (Boolean satisfiability problem, SAT)

Jednym z podstawowych zagadnień rachunku zdań w logice jest problem spełnialności. Polega on na znalezieniu takich wartości zmiennych, które mogą przyjmować tylko wartości prawda lub fałsz, dla których dana formuła logiczna będzie spełniona (Vizel, Weissenbacher i Malik, 2015). Przykładowy fragment takiej formuły może wyglądać następująco:

Problem ten, tak samo jak kolejne przedstawione w tej pracy, był pierwszym który udowodniono, że należy do grupy problemów NP-zupełnych (Cook, 1971). Oznacza to, że:

* Najlepsze rozwiązanie nie może zostać znalezione w czasie wielomianowym.
* Sprawdzenie, czy dane rozwiązanie jest poprawne jest możliwe w czasie wielomianowym.

Tak długo, jak formuła nie jest skomplikowana, to możemy sprawdzić po prostu każdą możliwą opcję. Jednak, jak łatwo zauważyć, liczba kombinacji rośnie wykładniczo – dla każdej zmiennej są możliwe dwa stany, więc potencjalnych rozwiązań jest tyle ile dwa do potęgi. Rośnie to na tyle szybko, że już dla 50 takich zmiennych, liczba możliwych stanów jest większa niż biliard. Już taka ilość jest bardzo trudna do sprawdzenia nawet w przypadku pomocy komputerów, a dla ludzi bez nich dosłownie niemożliwa.

Jednym z najczęściej spotykanych obecnie zastosowań dla silników rozwiązujących ten problem są zagadnienia związane z projektowaniem układów cyfrowych. Dzięki temu można chociażby znaleźć bardziej optymalne rozłożenia komponentów w układach scalonych (Nam, Sakallah i Rutenbar, 2002), oraz przeprowadzić formalną weryfikację mikroprocesorów potokowych (Bryant, German, Velev i Murray, 1999).

Kilka zdań (2-4) kończących/podsumowujących punkt, aby nie kończyć powołaniem na literaturę.

## Problem komiwojażera (travelling salesman problem, TSP)

W problemie komiwojażera (travelling salesman problem, skrót TSP) mamy zdefiniowaną listę miast i odległości pomiędzy każda parą z nich (Beardwood, Halton i Hammersley, 1959). Polega on na znalezieniu najkrótszej możliwej takiej trasy, która odwiedza każde miasto dokładnie raz, a na sam koniec wraca do miejsca początkowego. Jest to szczególny przypadek problemu marszrutyzacji, który pozwala na odwiedzenie każdego miasta więcej niż raz. Przykładowe rozwiązanie przedstawiono na poniższym rysunku.

Obraz zawierający tekst, zewnętrzne, ruch uliczny, jasne

Opis wygenerowany automatycznie

Rysunek 1.1: …..

Źródło: ….

Optymalnych rozwiązań dla problemu jest zawsze więcej niż jeden. Nawet jeżeli istnieje tylko jeden optymalny cykl dla takiego grafu, to zawsze możemy wygenerować nowe rozwiązanie, zaczynając podróż z innego miasta, a także można je odwrócić. Przykładowo, jeżeli przedstawimy rozwiązanie jako listę odwiedzanych miast, to jeżeli pierwsza była by optymalnym rozwiązaniem, to wszystkie następujące po niej także takie będą:

* 1-3-5-2-4
* 3-5-2-4-1
* 5-2-4-1-3, itd.

Występują w tym problemie pewnie uproszczenia w stosunku do prawdziwego życia. Jednym z nich jest fakt, że w rzeczywistości nie każde miasto musi mieć pomiędzy sobą bezpośrednią drogę. Dodatkowo nie zawsze koszt pokonania takiej ścieżki jest taki sam w obie strony. Może on także zależeć od godziny, w jakiej dana podróż się odbywa.

Problem ten został pierwszy raz zdefiniowany w latach 30 i jest jednym z najczęściej używanym punktów odniesienia dla nowo powstających metod optymalizacji. Ze względu na jego popularność, mimo tego, że jest on NP-kompletny, istnieje wiele algorytmów i heurystyk które pozwoliły wygenerować najlepsze możliwe rozwiązania nawet dla przypadków składających się z dziesiątek tysięcy miast. Często używanym zbiorem instancji problemu o rożnym poziomie trudności jest TSPLIB, a największym całkowicie rozwiązanym przypadkiem jest problem składający się z 85 900 miast (Rego, Gamboa, Glover i Osterman, 2011).

Kilka zdań (2-4) kończących/podsumowujących punkt, aby nie kończyć powołaniem na literaturę.

## Programowanie nieliniowe (Nonlinear programming, NLP)

Kolejna klasą problemów, są te związane z programowaniem nieliniowym. Polegają one na zalezieniu minimum, maksimum, lub punktów zerowych dla danej funkcji (Bazaraa i Shetty, 1979). Dodatkowo na przestrzeń rozwiązań mogą zostać nałożone ograniczenia, w postaci równości, lub nierówności, które muszą zostać spełnione, aby dane rozwiązanie było poprawne.

Zmienne mogą przyjmować dowolne wartości z zakresu liczb rzeczywistych. W związku z tym istnieje nieskończoność potencjalnych rozwiązań – ze względu na to, że zawsze możemy podać taką wartość, której jeszcze nie rozwiązywaliśmy. Nie każde z tych rozwiązań za to musi być optymalne lub w ogóle poprawne. W szczególności możemy dla danego problemu nie wiedzieć, czy w ogóle istnieje takie rozwiązanie które spełnia wszystkie podane ograniczenia i dana metoda może próbować znaleźć jakiekolwiek które będzie poprawne.

Przykładowym problemem może być znalezienie maksimum dla następującej funkcji (Keane, 1996), z następującymi ograniczeniami:

Możemy zwizualizować tą funkcję przykładowo dla (Michalewicz i Fogel, 2004). Rozwiązaniom niespełniającym ograniczeń nadano zerową wartość (Rysunek 1.2).



Jak widać powierzchnia tego wykresu jest bardzo nieregularna, zwłaszcza im bliżej początków osi. Dodatkowo możemy zaobserwować tutaj nagły spadek do zera, co oznacza niepoprawne rozwiązania. Jest on bardzo blisko fragmentu gdzie znajdują się najlepsze rozwiązania według tego wykresu, co pokazuje, że granica pomiędzy globalnym maksimum, a nieprawidłowym rozwiązaniem może być bardzo cienka. W związku z tym niektóre metody rozwiązań koncentrują się właśnie na tych krawędziach, uważając je za rejony z największym potencjałem. Oczywiście wszystko to zależy od konkretnego przypadku dla danego równania.

## Problem planowania projektu z wieloma wymaganymi umiejętnościami i ograniczonymi zasobami (Multi-Skill Resource-Constrained Project Scheduling Problem, MSRCPSP)

Zbiór danych dla problemu planowania projektów z wieloma wymaganymi umiejętnościami i ograniczonymi zasobami, który został nazwany ….. (MSRCPSP)został zdefiniowany przez naukowców związanych z Politechniką Wrocławską. Ich celem było stworzenie takiego problemu, która miał by jednoczesny balans prostoty implementacji i wierności rzeczywistej sytuacji (P.B., M. i K., 2015).

Celem klasycznego problemu planowania projektów było przydzielenie zasobów do zadań w taki sposób, aby zminimalizować czas i/lub koszt wykonania projektu. Jednakże pojawiają się w nim także ograniczenia:

* Niektóre zadania, mogą do rozpoczęcia pracy nad nimi wymagać zakończenia innych zadań.
* Dany zasób może zostać przydzielony tylko do jednego zadania naraz i musi je skończyć w całości, bez możliwości dzielenia pracy pomiędzy dwoma rozpoczętymi zadaniami.

Problem ten został rozszerzony przez dodanie umiejętności. Każdy zasób posiada pewien zbiór umiejętności na określonym poziomie, a każde zadanie wymaga jedną umiejętność na określonym poziomie. W związku z tym zadanie może mieć przydzielony dany zasób, tylko jeżeli posiada on daną umiejętność na równym lub wyższym poziomie niż to potrzebne.

Został on opracowany razem ze współpracy z inżynierami z firmy Volvo. Dzięki temu jego twórcy dostarczają oparte na rzeczywistości, ale zanonimizowane, zestawy danych odpowiadające rzeczywistym projektom. Jest ich łącznie 42: tzn. 6 łatwiejszych problemów i 36 pełnowymiarowych, które posiadają do:

* 200 zadań.
* 150 relacji pomiędzy zadaniami.
* 15 umiejętności na różnych poziomach.
* 40 zasobów.

Kilka zdań (1-4) kończących/podsumowujących punkt/rozdział. Nie kończy się punktu wypunktowaniem (jeśli nie jest to podsumowanie).

## Elementu problemu

### Model i wielkość przestrzeni poszukiwań

Aby użyć jakikolwiek algorytm lub heurystykę do rozwiązania danego problemu musimy najpierw zdefiniować pewne podstawowe koncepty dla danego problemu:

* Model dla problemu.
* Sąsiedztwo rozwiązań.
* Funkcję oceny rozwiązania.
* Cel, który chcemy osiągnąć.
* Potencjalne ograniczenia nałożone na rozwiązania.

Pierwsza rzeczą, którą trzeba zdefiniować w celu rozwiązania danego problemu obliczeniowego, jest zdefiniowanie jego modelu. Jest to określenie sposobu, w jaki możemy przedstawić alternatywne rozwiązania i umożliwić nam ich modyfikację. Nie istnieje jeden najlepszy model dla danego problemu. Jak udowodnili (Fogel i Ghozeil, 1997) wśród takich przedstawień problemu, które są dla siebie bijekcjami, żadna z nich nie daje przewagi w rezultatach nad innymi. W związku z tym najczęściej wybierane są takie, które są najbardziej intuicyjne dla danego problemu, co pozwala na ich łatwiejsze zrozumienie.

Przypadek SAT jest najprostszy do zamodelowania. Jego rozwiązaniem jest ciąg binarnych wartości, reprezentujących po kolei stany jakie przypisujemy poszczególnym zmiennym. W związku z tym możliwych rozwiązań jest dokładnie . Dla 100 zmiennych jest to wartość rzędu .

W przypadku TSP najczęściej spotykanym modelem jest permutacja liczb naturalnych od do . Każda z liczb jest przypisana do konkretnego miasta i ich kolejność w danym rozwiązaniu jest także kolejnością w której zostaną odwiedzone. W związku z tym, że w podstawowym problemie komiwojażera odległości pomiędzy miastami są symetryczne, to nie ma znaczenia dla danego rozwiązania czy lista miast zostanie przeprocesowana od lewej do prawej, czy od prawej do lewej. Dodatkowo także nie ma znaczenia od którego miasta zaczniemy taką podróż. W związku z tymi dwoma obserwacjami wielkość przestrzeni poszukiwań wynosi . Dla porównania z poprzednim problemem, dla 100 miast wartość ta jest rzędu

Dla NLP teoretyczna przestrzeń przeszukiwań jest nieograniczona, ponieważ każda zmienna może przyjąć dowolną wartość ze zbioru liczb rzeczywistych. W związku z tym potrzeba każdą ze zmiennych odpowiednio poddać dyskretyzacji, aby było możliwe zastosowanie komputerów w celach obliczeniowych. W związku z tym możemy albo dokonać tego procesu samemu, na przykład dzieląc daną przestrzeń na określoną liczbę punktów w stałej odległości, lub skorzystać z precyzji jakie dają nam liczby zmiennoprzecinkowe na danych platformach obliczeniowych. W przypadku, gdy użylibyśmy standardowych liczb zmiennoprzecinkowych o podwójnej precyzji (IEEE Standard for Floating-Point Arithmetic, 2008) takich rozwiązań było by , co dla równań z 100 zmiennymi dało by rząd wielkości .

W przypadku MSRCPSP można zastosować różne potencjalne modele rozwiązania. Można przykładowo skupić się wyłącznie na priorytecie zadań, a do przydzielenia jakie zadanie ma być wykonane przez jaki zasób, zastosować podejście z użyciem prostego algorytmu zachłannego. Jest też możliwy sposób dokładnie odwrotny, czyli skupić się na przypisaniu zasobów do zadań i nie przejmować się tak dokładnie kolejnością zadań. Możliwe są także rozwiązania ustalające zarówno priorytet zadań jak i przypisania konkretnych zadań do nich. Dla takiego sposobu potencjalnych rozwiązań jest , gdzie to liczba zasobów, a to liczba zadań. Dla 100 zasobów i 100 zadań daje to rząd wielkości potencjalnych rozwiązań równy

### Sąsiedztwo

Integralna częścią niektórych algorytmów oraz heurystyk jest pojęcie sąsiedztwa. Dwa rozwiązania są swoimi sąsiadami jeżeli są w pewien mierzalny sposób odpowiednio blisko siebie. Idąc dalej takie sąsiedztwem dla danego punktu w przestrzeni poszukiwań możemy nazwać wszystkie inne rozwiązania które spełniają taki warunek. Często takie punkty są uzyskiwane przez dokonanie jednej jak najmniejszej zmiany, ale konkretna definicja takiego warunku zależy od modelu problemu.

Dla SAT możemy zdefiniować je, poprzez odwrócenie wartości jednej ze zmiennych. Przykładowo jeżeli będziemy mieli rozwiązanie zapisane w postaci ciągu liczb binarnych: 01110, to sąsiedztwo dla niego będzie wyglądało następująco:

* 11110 (zamiana pierwszego bitu).
* 00110 (zamiana drugiego bitu).
* 01010 (zamiana trzeciego bitu).
* 01100 (zamiana czwartego bitu).
* 01111 (zamiana piątego bitu).

Dla TSP nie możemy zamienić tylko pojedynczego miasta na inne, ponieważ wtedy występowało by ono w rozwiązaniu więcej niż jeden raz, więc było by ono nieprawidłowe. W związku z tym najprościej jest zamienić dwa miasta miejscami. W zależności od tego ile sąsiadów chcemy wygenerować, możemy zamieniać miejscami tylko dwa miasta leżące obok siebie w danej ścieżce, lub dwa losowe miasta niezależnie od ich pozycji. W pierwszym wypadku, dla przykładowej ścieżki wyglądającej następująco: 1 – 3 – 4 – 5 – 2, zostały by wygenerowane następujące rozwiązania:

* 3 – 1 – 4 – 5 – 2 (zamiana pierwszego miasta z drugim).
* 1 – 4 – 3 – 5 – 2 (zamiana drugiego miasta z trzecim).
* 1 – 3 – 5 – 4 – 2 (zamiana trzeciego miasta z czwartym).
* 1 – 3 – 4 – 2 – 5 (zamiana czwartego miasta z piątym).
* 2 – 4 – 3 – 5 – 1 (zamiana piątego miasta z pierwszym).

Dla NLP jednym z podejść jest określenie maksymalnej odległości dla danych zmiennych dla której dwa punkty mogą dalej zostać zdefiniowane jako swoi sąsiedzi. Gdy mamy taką definicję, to możemy zmienić jedną z wartości punktów o losową wartość z przedziału nieprzekraczającej jej. Przykładowo dla rozwiązania składających się z trzech zmiennych o następujących wartościach: i maksymalnej odległości równej , każdy z poniższych punktów byłby sąsiadem:

* (zmiana wartości pierwszej zmiennej).
* (zmiana wartości drugiej zmiennej).
* (zmiana wartości trzeciej zmiennej).
* (zmiana wartości pierwszej zmiennej).
* (zmiana wartości drugiej zmiennej).

Dla MSRCPSP definicja sąsiedztwa zależy od modelu jaki zostanie wykorzystany. Dla listy priorytetów zadań, sąsiedztwo można zdefiniować przez zamianę priorytetów dwóch dowolnych zadań, podobnie jak w TSP. Za to dla listy przypisań zasobów można podejść do tego jako zmianę danego przypisanego zasobu, na dowolny inny zgodny z ograniczeniami. W przypadku gdy obecne są te dwie listy naraz, to sąsiedzi mogą być generowani na dwa wcześniej wymienione sposoby.

Mając zdefiniowane pojęcie sąsiedztwa możemy zdefiniować także pojęcie lokalnego optimum. Dane rozwiązanie jest w nim wtedy, gdy jest ono lepsze, lub co najmniej równe, niż wszystkie inne z jego sąsiedztwa. Najprostsze algorytmy poszukiwań rozwiązań, skupiają się tylko i wyłącznie na lokalnym optimum. Niestety w większości wypadków takie optimum nie jest jednocześnie globalnym. Wielkość zdefiniowanego sąsiedztwa pokazuje dla nich zależność pomiędzy skutecznością w poszukiwaniu rozwiązań, a czasem wykonania. Gdy takie sąsiedztwo jest niewielkie, to wtedy możemy szybko przeszukać wszystkie możliwości, jednakże taki algorytm może nie zauważyć jeszcze lepszego rozwiązania które jest tuż obok. W przeciwnym za to wypadku, taki algorytm może dojść do lepszych wyników w tej samej liczbie iteracji, jednakże czas jego wykonania może wzrosnąć do takiego stopnia, że będzie zupełnie bezużyteczny. W każdym wypadku taki rozmiar musi zostać dostosowany do konkretnego problemu dla którego ma zostać znalezione rozwiązanie.

### Funkcja oceny rozwiązania, cel i ograniczenia rozwiązania problemu

Aby osiągnąć jak najlepsze rozwiązanie, należy zdefiniować funkcje oceny. W zależności od tego co chcemy osiągnąć, sposoby rozwiązania problemów dążą do jej minimalizacji, lub maksymalizacji. Odpowiednia funkcja powinna być jak najszybsza do obliczenia, nawet kosztem pewnych przybliżeń, ponieważ jest to element algorytmów i heurystyk który zwykle zajmuje najwięcej czasu w ich poszukiwaniach (He, Chen i Yao, 2015).

Można je podzielić na dwie typy. Pierwszym są porządkowe – pozwalają one na porównanie ze sobą dwóch rozwiązań. Drugim zaś typem są numeryczne – pozwalają one dodatkowo na określenie na ile jedno rozwiązanie jest lepsze od drugiego. Numeryczne pozwalają na większą elastyczność w projektowaniu odpowiedniego sposobu rozwiązania problemu, jednakże nie zawsze jest możliwe ich zastosowanie, a także mogą być droższe

Innym podziałem jest podział na funkcje statyczne i dynamiczne – w tych pierwszych wartość oceny danego rozwiązania nie zmienia się, a w drugim przeciwnie. Przykładem jest próba opracowania najlepszego algorytmu do gry w szachy, gdzie jako funkcje oceny zwykle stosuje się wyniki w grze przeciwko innym, w związku z czym może ona się zmieniać w zależności od tego, z jakimi algorytmami zostanie on porównany.

Kolejną trudnością w projektowaniu takiej funkcji, jest fakt, że może nam zależeć na kilku rożnych kryteriach optymalizacji. Przykładowo przy wyborze samochodu może nam zależeć zarówno na jego cenie, jak i odpowiedniej mocy, wyposażeniu czy zużyciu paliwa. Aby połączyć te kryteria razem, możemy zastosować sumę ważoną, przydzielając odpowiednią wagę dla każdego z celów, jaki chcemy osiągnąć i w ten sposób łącząc je w pojedynczą wartość.

W przypadku większości problemów trzeba także uwzględniać ograniczenia w możliwych rozwiązaniach. Aby sobie z nimi poradzić jest kilka możliwych rozwiązań, a w zależności od tego z jakim problemem mamy do czynienia, mogą zostać zastosowane inne sposoby:

* Zaprojektowanie takiego sposobu rozwiązania problemu, aby tworzyć tylko poprawne rozwiązania.
* Opracowanie sposobu naprawy nieprawidłowych rozwiązań tak, aby były z powrotem poprawne.
* Dodawanie kary do funkcji oceny gdy dane rozwiązanie nie spełnia ograniczeń.
* Stosowanie dwóch różnych funkcji oceny, w zależności od tego czy rozwiązanie jest poprawne czy nie.

Dla SAT liczba poprawnych rozwiązań jest bardzo mała, a w szczególności może nawet wynosić jeden. W związku z tym, sposoby rozwiązywania tego problemu muszą operować na nieprawidłowych rozwiązaniach. Najczęstszą spotykaną funkcja oceny jest ilość spełnionych części całego wyrażenia i w tym wypadku dążymy do jej maksymalizacji.

W przypadku TSP jako naturalna funkcję oceny możemy potraktować odległość danej drogi. W tym przypadku dążymy do jej minimalizacji. Ze względu na to, że wszystkie prawidłowe rozwiązania są swoimi permutacjami, to często spotyka się tak zaprojektowane algorytmy, aby uwzględniały to i produkowały tylko i wyłącznie poprawne rozwiązania.

Zaś w przypadku NLP funkcją oceny jest wartość samej funkcji która chcemy zoptymalizować. W zależności od konkretnego przypadku chcemy ją minimalizować lub maksymalizować. Także sposób obchodzenia się z ograniczeniami bardzo często zależy od charakteru funkcji nad którą pracujemy.

MSRCPSP jest problemem, który ma dwa różne kryteria optymalizacji. Można dążyć zarówno do minimalizacji czasu wykonania danego harmonogramu, jak i do minimalizacji jego kosztu. Jako że minimalizacja kosztów jest łatwiejsza – można wybrać tylko najtańsze zasoby – postanowiłem się w mojej pracy skupić wyłącznie na minimalizacji czasu. Dodatkowo zaprojektowałem wszystkie zastosowane przeze mnie operator i reprezentację tak aby generowały wyłącznie poprawne rozwiązania, omijając w ten sposób problemy związane z naprawą lub eliminacja nieprawidłowych rozwiązań.

# Metody heurystyczne do rozwiązywania problemów

## Metaheurystyki w rozwiązywaniu problemów

Gdy zdefiniowaliśmy już elementy, z których składają się problemy obliczeniowe, to następnie możemy przejść do opisu potencjalnych metod, które pozwolą nam na uzyskaniu jak najlepszych rozwiązań. Ważny w tym wypadku jest odpowiedni balans pomiędzy skutecznością metody w poszukiwaniu co raz to lepszych rozwiązań, a także jej czasem działania. W skrajnym przypadku dla wielu problemów przegląd zupełny rozwiązań znalazł by dla nas to najbardziej optymalne rozwiązanie, jednak zajęło by to zbyt dużo czasu żeby było to w ogóle możliwe.

Dla przedstawionych wcześniej w tej pracy problemów nie są znane takie algorytmy, które pozwalały by na uzyskanie najlepszego rozwiązania w akceptowalnym czasie. W związku z tym w ich wypadku stosowane są metaheurystyki. Są to uniwersalne wysokopoziomowe podejścia, które dostarczają zbiór zasad, czy strategii w celu stworzenia heurystycznego algorytmu optymalizacyjnego (Sörensen i Glover, 2013). Otrzymane w ten sposób metody są nazywane metodami heurystycznymi i w przeciwieństwie do klasycznych algorytmów, nie gwarantują one znalezienia optymalnego rozwiązania.

Metody heurystyczne można podzielić na dwie kategorie: takie, które operują tylko na kompletnych rozwiązaniach i takie, które operują także na niekompletnych. W tym pierwszym wypadku jeżeli zatrzymamy działanie metody przed czasem, to zawsze otrzymamy prawidłowe rozwiązanie, w drugim wypadku może to być niemożliwe.

Opisać/wymienić typologię metod

## Algorytm zachłanny

Algorytmy zachłanne opierają się na tworzeniu pełnego rozwiązania krok po kroku. Powodem ich popularności jest ich prostota. Główna idea stojąca za nimi jest następująca: za każdym razem gdy trzeba podjąć decyzję o dołożeniu nowej części do rozwiązania, wybierz taką, która dla danego częściowego rozwiązania da najlepszy rezultat w danym momencie. To podejście zakłada heurystycznie, że każde gdy będziemy podążać najlepszymi krokami w danym momencie, to osiągniemy finalnie najlepsze rozwiązanie - lecz oczywiście jest to dość krótkowzroczna metoda, ponieważ nie zawsze tak będzie. Stąd bierze się nazwa algorytmu zachłannego.

Dla SAT można opracować następujący algorytm zachłanny: dla każdej zmiennej, w dowolnej kolejności, dopasuj taką jej wartość prawda lub fałsz, która w danym momencie sprawi, że jak największa liczba podrównań będzie spełniona. W przypadku remisu, można wybrać tą wartość na przykład losowo, lub na przemian. Niestety takie podejście napotka na problem na takim prostym przykładowym równaniu:

Jak widać, w tym wypadku, takie podejście przydzieliło by najpierw wartość dla zmiennej , ponieważ spełnione wtedy zostały by drugie i trzecie podrównania. Niestety, wtedy pierwsze z nich nie będzie spełnione i cały przypadek pozostanie bez rozwiązania, nie zależnie od tego jakie wartości zostaną przydzielone dla innych zmiennych. Można by było ulepszać ten algorytm, dodając co raz to kolejne reguły, jednakże nigdy nie będzie możliwe stworzenie takiego algorytmu zachłannego, który by pozwolił na znalezienie rozwiązania w każdy wypadku. Wynika to z przedstawionego wcześniej faktu, iż problem SAT jest NP-zupełny i jego rozwiązanie w czasie wielomianowym jest niemożliwe.

Podobnie jest w przypadku NLP. Można było by opracować algorytm, który starał by się wiele razy dla każdej ze zmiennych dobrać najlepsza wartość w danym momencie, tworząc pewnego rodzaju wielowymiarowe przeszukiwanie liniowe. Jednakże problem z takim podejściem jest taki, jak z innymi algorytmami genetycznymi – patrzą one krótkowzrocznie i pomijają interakcje jakie występują pomiędzy różnymi zmiennymi (Himmelblau, 1972).

Za to w przypadku TSP wygląda to trochę inaczej – o wiele łatwiejsze tutaj jest opracowanie takiego algorytmu zachłannego, który generuje zawsze poprawne rozwiązania. W związku z tym istnieją takie metody rozwiązywania problemów, które przykładowo zaczynają swoje działanie na zbiorze początkowym który został wygenerowany przez algorytm genetyczny, zamiast na losowych rozwiązaniach. Jedną z najprostszych procedur, jaka przychodzi tutaj do głowy, jest zaczęcie w dowolnym losowym mieście i następnie odwiedzanie zawsze tego miasta, do którego jest najbliżej i w które jeszcze nie jest na wygenerowanej ścieżce. Oczywiście tak jak poprzednio, sposób ten jest chciwy i zawsze wybierze najkrótszą w danej chwili ścieżkę, nie ważne czy byłaby ona w końcowym rozrachunku optymalna.

Dla MSRCPSP zdecydowałem się na zastosowanie proste algorytmu zachłannego, podobnego dla tego dla TSP. Sposób jego działania jest następujący: wybiera po kolei takie zadanie i przypisuje taki do niego zasób, który w danym momencie powiększy czas wykonania całego harmonogramu o jak najmniejszą wartość. W wypadku kilku takich możliwości, jest ono losowane spośród z nich. Jest to jak widać bardzo prosty algorytm, jednak zdecydowałem się na jego implementacje, aby można było go porównać z innymi, także bardziej skomplikowanymi, metodami.

## Przeszukiwanie lokalne

Przeszukiwanie lokalne skupia się na wyjaśnionym już wcześniej koncepcie sąsiedztwa. Zamiast robić przegląd zupełny rozwiązań, polega ono na modyfikacji tych już istniejących, w celu ich poprawy. Początkowe rozwiązania mogą zostać wygenerowane losowo, lub przykładowo wybrane z określonych regularnie punktów w ich przestrzeni poszukiwań. Sposób działania tej metody jest następujący: wygeneruj początkowe rozwiązanie, a następnie wybierz jednego, lub więcej, z jego sąsiadów i jeżeli będzie któryś z nich lepszy, to skup swoje poszukiwania na nim.

Metoda ta wymaga dobrania odpowiedniej definicji sąsiedztwa. Gdy potencjalnych sąsiadów będzie mało, to ich przejrzenie będzie szybkie, jednak może on łatwiej utknąć w lokalnym minimum. A gdy będzie ich zbyt dużo, to jego wykonanie może trwać po prostu zbyt długo. W szczególnym wypadku, gdyby w takim sąsiedztwie znalazło by się każde inne rozwiązanie, metoda ta zamieniła by się w przegląd zupełny. Nie ma jednej uniwersalnej wartości jaka pasuje do każdego przypadku i tak jak w przypadku innych, musi ona być zawsze strojona pod konkretny określony problem, do którego chcemy znaleźć jak najlepsze rozwiązanie.

Problemem tego podejścia jest fakt, że bardzo łatwo może ono utknąć w lokalnym minimum i nie ma żadnego sposobu na wyjście z niego, gdy już tam wpadnie. Mimo tych wad jest to jedna z najszybszych i najłatwiejszych do implementacji metod. W związku z tym jest czasami łączona z innymi, bardziej skomplikowanymi, aby lepiej przygotować początkową pulę rozwiązań, zamiast przykładowo je losować. Analogicznie także może zostać ona wykorzystana na sam koniec innych działania algorytmów czy metaheurystyk, aby zobaczyć czy nie przegapiły one prostych zmian, które mogły by jeszcze wprowadzić finalne poprawki.

## Symulowane wyżarzanie

Aby poprawić działanie poprzedniej metody zostało opracowane symulowane wyżarzanie. Jak wiele innych technik rozwiązywania takich problemów zostało one zainspirowane prawdziwymi zjawiskami występującymi w przyrodzie – w tym wypadku procesami zachodzącym podczas obróbki cieplnej metali (Kirkpatrick, Gelatt i Vecchi, 1983).

Metoda ta bazuje na sposobie działania przeszukiwania lokalnego z jedną, ale istotną zmian. Jeżeli funkcja oceny dla nowego wygenerowanego rozwiązania zwraca wartość gorszą niż dla obecnego , to może zostać ono i tak zaakceptowane z prawdopodobieństwo równym: , gdzie to malejąca z czasem temperatura. Równanie to ma zastosowanie w przypadku problemów minimalizacji, w przypadku maksymalizacji różnica ta musi zostać odwrócona. Zmiana ta pozwala na wyjście z lokalnego minimum i potencjalne trafienie w przyszłości na rejony z lepszymi rozwiązaniami, niż te na które metoda trafiła na początku.

Im ocena nowego, ale gorszego rozwiązania, jest bliższa ocenie tego poprzedniego, tym większa jest szansa na jego zaakceptowanie. Przykładowe prawdopodobieństwa przedstawia pierwsza z poniższych tabel.

Tabela …..

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 100 | 10 | 100 | 100% |
| 100 | 10 | 101 | 90,48% |
| 100 | 10 | 102 | 81,87% |
| 100 | 10 | 105 | 60,65% |
| 100 | 10 | 110 | 36,79% |
| 100 | 10 | 120 | 13,53% |
| 100 | 10 | 150 | 0,67% |

Źródło: ….

Im większa tym różnica, tym szansa na akceptacje nowego rozwiązania staje się coraz mniejsza. W przypadku gdy jest tylko nieznacznie gorsze, jest ona dość duża, a w wypadku przeciwnym, jest ona minimalna i dąży do zera. Ma na to także wpływ wysokość temperatury w danym momencie, co jest przedstawione w drugiej z tabel.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 100 | 100 | 110 | 90,48% |
| 100 | 50 | 110 | 81,87% |
| 100 | 20 | 110 | 60,65% |
| 100 | 10 | 110 | 36,79% |
| 100 | 5 | 110 | 13,53% |
| 100 | 2 | 110 | 0,67% |

Funkcja temperatury jest funkcja malejąca i jej wzór, tak samo jak wysokość temperatury początkowej, muszą zostać dopasowane indywidualne do danego problemu dla jakiego mają zostać znalezione rozwiązania Wraz z postępem prac tej metody, temperatura ta maleje i związku z tym, prawdopodobieństwo także dąży do zera. W przypadku swojej implementacji tego algorytmu do problemu harmonogramowania zdecydowałem na określenie temperatury bieżącej jako iloraz temperatury maksymalnej, przez numer bieżącej iteracji metody.

## Algorytm genetyczny

Ostatnia z przedstawionych w tej pracy metod jest najbardziej skomplikowana. Podobnie jak poprzednia, jest tak samo inspirowana procesami zachodzącymi w przyrodzie, a konkretnie działaniem naturalnej ewolucji wśród żywych organizmów. Schemat jej działania można opisać ją w następujący sposób:

1. Wygeneruj określoną liczbę losowych osobników.
2. Powtarzaj przez określoną liczbę iteracji:
   1. Przeprowadź proces selekcji.
   2. Przeprowadź proces krzyżowania.
   3. Przeprowadź proces mutacji.
3. Zwróć najlepsze rozwiązanie.

Początkowe osobniki zwykle generowane są w sposób losowy. Pierwszym krokiem działania tej metaheurystyki jest selekcja. Metod na przeprowadzenie jej jest wiele i zwykle mogą zostać zastosowane do wielu problemów. Przykładowe sposoby to:

* Selekcja turniejowa – polega ona na wybraniu określonej liczby losowych osobników z populacji, a następnie wybraniu najlepszego z nich. Proces ten powtarza się tak długo aż to potrzebne. W tym wypadku im większy rozmiar turnieju, tym mniejsza szansa na wybranie mniej optymalnych osobników
* Selekcja rankingowa – polega ona na uszeregowaniu wszystkich osobników z populacji i wybraniu tylko tych najlepszych
* Selekcja ruletkowa – polega ona na losowaniu osobników z populacji, gdzie im lepsza ocena danego osobnika, tym większa jego szansa na wylosowanie go
* Selekcja elitystyczna – polega ona na wybraniu najlepszego osobnika z populacji i zachowanie go w niezmienionej postaci do końca iteracji metody

Selekcja taka może wybrać liczbę osobników w nowej populacji równej poprzedniej i wtedy niektóre osobniki będą dla siebie identyczne, lub wybrać mniejszą i braki uzupełnić losowo wygenerowanymi nowymi rozwiązaniami W swojej implementacji algorytmu genetycznego zdecydowałem się na wykorzystanie połączonej selekcji elitystycznej, z turniejową.

Następnym krokiem jest proces krzyżowania, którego celem jest stworzenie nowych rozwiązań które będą łączyły cechy innych dwóch, lub większej ilości, rozwiązań i dzięki temu uzyskanie potencjalnie lepszych osobników. W przeciwieństwie do poprzedniego, jest on wykonywany z pewnym prawdopodobieństwem i nie wszystkie osobniki będą brały udział w nim. W przypadku problemu SAT proces ten jest najprostszy – część wartości jest brana z pierwszego rozwiązania, a pozostałe z drugiego. W przypadku NLP oprócz takiego podejścia, można zamiast tego chociażby uśrednić wartości dla krzyżowanych osobników – efektem tego będzie powstanie dwóch identycznych osobników, o ile nie zostaną złamane żadne dodatkowe ograniczenia. W przypadku TSP krzyżowanie jest bardziej skomplikowane – w związku z tym, że dane miasto nie może zostać odwiedzone dwa razy, należy zastosować takie operatory których efektem działania będzie zachowanie prawidłowej kolejności. Zaś dla przypisania priorytetów zdecydowałem się na implementacje operatora krzyżowania pozycji (Syswerda, 1991): wybiera on część priorytetów z pierwszego osobnika, a brakujące uzupełnia zgodnie z kolejnością z drugiego.

Ostatnim krokiem jest proces mutacji, którego celem jest taka zmiana cech dotychczasowych osobników, aby być może udało się wyjść z lokalnego minimum lub maksimum. Podobnie jak poprzedni, jest on wykonywany z prawdopodobieństwem, jednak zwykle mniejszym niż to, które występuje w przypadku krzyżowania. Schemat działania jest bardzo podobny do procesu generowania nowych sąsiadów, który został opisany w jednym z poprzednich rozdziałów. Gdy dany osobnik zostanie wybrany do poddania się mutacji, to jest generowany jego sąsiad który go zastąpi. Należy odpowiednio dobrać szansę na zadziałanie tego operatora. Zbyt niska wartość prawdopodobieństwa mutacji spowoduje, że populacje szybko utkną w lokalnych minimach, a zbyt wysoka sprawi, że wręcz przeciwnie, nie zdążą one osiągnąć żadnych dobrych rezultatów przed kolejną taką modyfikacją.

Jak widać metoda ta pozwala na wiele różnych modyfikacji, ale i wymaga odpowiedniego ustawienia tak samo wielu różnych parametrów. W związku z tym, zaletą jej jest fakt, że można ją zastosować do rozwiązywania różnych typów problemów. Jest to jednocześnie też jej wadą, ponieważ trudne jest określenie najbardziej optymalnego takiego zestawu i jest to różne w zależności od problemu który chcemy rozwiązać, a nawet od konkretnych danych dla danego problemu.

# Założenia realizacji badania

## Cel badania i zastosowana procedura badania

W poniższym rozdziale zostanie przedstawiona metodyka badań, podane wyniki uzyskane przy użyciu wytworzonej aplikacji oraz zostanie przedstawiona ich analiza. Zostały one przeprowadzone dla zdefiniowanego wcześniej problemu MSRCPSP, przy użyciu przedstawionych w poprzedniej części pracy metod obliczeniowych.

Tutaj opisać najpierw – zdefiniowany problem badawczy, potem jaki jest cel badania, następnie, że w badaniu zastosowano MSRCPSP.

Baranie przeprowadzono według następującej procedury:

1. Identyfikacja problemu.
2. Zdefiniowanie elementów problemu.
3. Wybór metod heurystycznych.
4. Implementacja metod heurystycznych.
5. Dobranie parametrów dla metod heurystycznych.
6. Przeprowadzanie eksperymentu.
7. Analiza wyników eksperymentu.
8. Wizualizacja wyników.
9. Sformułowanie wniosków.

## Przyjęte założenia dla eksperymentu

oraz przyjęte założenia co do MSRCPSP (pojawiły się wcześniej w opisie tej pracy – wyjaśnię)

Na problemie MSRCPSP skupiłem swoje badania, których założenia i wyniki zostaną przedstawione w nadchodzących rozdziałach.

W przypadku MSRCPSP zdecydowałem się na zastosowanie modelu składającego się z dwóch list. Pierwsza z nich to lista przypisań danych zasobów do danych zadań, a druga z nich to lista priorytetów z jakimi mają zostać wykonane dane zadania. Sposób ten pozwala na pełną kontrolę nad rozwiązaniem - w przeciwieństwie do modeli opartych na np. samej liście priorytetów zadań. Nie wymaga on zastosowywania żadnych dodatkowych heurystyk przy obliczaniu momentu rozpoczęcia zadań. Wadą tego podejścia jest konieczność zaprojektowania dwóch różnych zachowań dla każdej z list, w przypadku każdej operacji jaką chcemy wykonać na takim rozwiązaniu.

Dla MSRCPSP zdecydowałem się na definicje sąsiedztwa inspirowaną wyżej przedstawianymi przykładami. Dla listy priorytetów zadań, sąsiedztwo zdefiniowałem przez zamianę priorytetów dwóch dowolnych zadań, podobnie jak w TSP. Za to dla listy przypisań zasobów, zdecydowałem się na zmianę danego przypisanego zasobu, na dowolny inny zgodny z ograniczeniami. Każdy sąsiad może być oddalony od drugiego tylko o jedno naraz z tych dwóch transformacji. Generuje to dwa różniące się od siebie typy sąsiedztwa, jednak było to wymagane przez podwójny charakter przyjętego przeze mnie modelu.

Jako że minimalizacja kosztów jest łatwiejsza – można wybrać tylko najtańsze zasoby – postanowiłem się w mojej pracy skupić wyłącznie na minimalizacji czasu. Dodatkowo zaprojektowałem wszystkie zastosowane przeze mnie operator i reprezentację tak aby generowały wyłącznie poprawne rozwiązania, omijając w ten sposób problemy związane z naprawą lub eliminacja nieprawidłowych rozwiązań.

Dla MSRCPSP zdecydowałem się na zastosowanie proste algorytmu zachłannego, podobnego dla tego dla TSP. Sposób jego działania jest następujący: wybiera po kolei takie zadanie i przypisuje taki do niego zasób, który w danym momencie powiększy czas wykonania całego harmonogramu o jak najmniejszą wartość. W wypadku kilku takich możliwości, jest ono losowane spośród z nich. Jest to jak widać bardzo prosty algorytm, jednak zdecydowałem się na jego implementacje, aby można było go porównać z innymi, także bardziej skomplikowanymi, metodami.

Selekcja

W przypadku MSRCPSP zdecydowałem się na zastosowanie dwóch sposobów. Dla przypisania zasobów wybrałem następującą metodę: podziel rozwiązania na losowej wielkości dwie części, a następnie zamień je nawzajem częściami, tak aby powstały dwa osobniki z elementami z obu początkowych rozwiązań.

Mutacja

## Opis sposobu realizacji projektu i badania

Każda z podanych wcześniej metod uruchomiłem dziesięć razy, na każdym z dostarczonych przez naukowców z Politechniki Wrocławskiej zbiorze danych. Następnie dla każdego uruchomienia została zapisana ocena najlepszego rozwiązania i dla uzyskanych w ten sposób rozwiązań dla danego zbioru została policzona wartość minimalna, średnia i odchylenie standardowe. Na większą ilość iteracji nie pozwoliła dostępna mi moc obliczeniowa mojego komputera.

Badania zajęły około 9 godzin, przy wykorzystaniu wielowątkowości, na komputerze wyposażonym w procesor Intel Core i9-9900K podkręconym w celu osiągniecia stałego taktowania 4,7 GHz, oraz posiadającym 32 GB pamięci operacyjnej DDR4 o taktowaniu 3600 MHz. Kod programu został napisany przy użyciu języka Java, a także został sprofilowany, aby usunąć potencjalne nieoptymalne implementacje algorytmów.

Parametry uruchomieniowe dla metod prezentują się następująco:

* Algorytm genetyczny:
  + Rozmiar populacji: 100.
  + Liczba iteracji: 10000.
  + Rozmiar selekcji turniejowej: 5.
  + Szansa na krzyżowanie: 100%.
  + Szansa na mutację: 50%.
* Przeszukiwane lokalne:
  + Ilość powtórzeń: 100000.
* Symulowane wyżarzanie:
  + Ilość iteracji: 1000000.
  + Temperatura maksymalna: 250000.
* Algorytm zachłanny:
  + Ilość powtórzeń: 100.

Parametry te zostały dobrane empirycznie, na bazie wcześniejszych doświadczeń na mniejszych zbiorach danych. Zostały one dobrane tak, aby czas działania każdej z metod był zbliżony do siebie, a uzyskane wyniki jak najlepsze. Średni taki czas jednej iteracji metody wyniósł około 77 sekund.

# Wyniki badań

## Otrzymane wyniki z przeprowadzonego badania

Wprowadzenie do punktu – wyjaśnię

Wprowadzenie do pierwszej tabeli i ogólne wnioski – potem tabela 1; następnie według tego samego schematu – kolejne – wyjaśnię

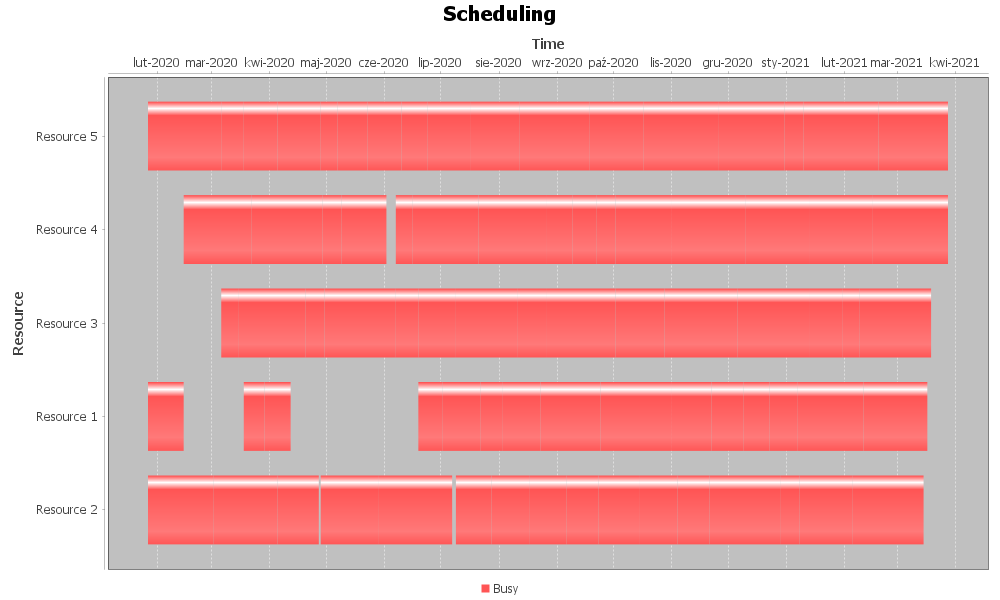
Otrzymane wyniki prezentują się następująco:

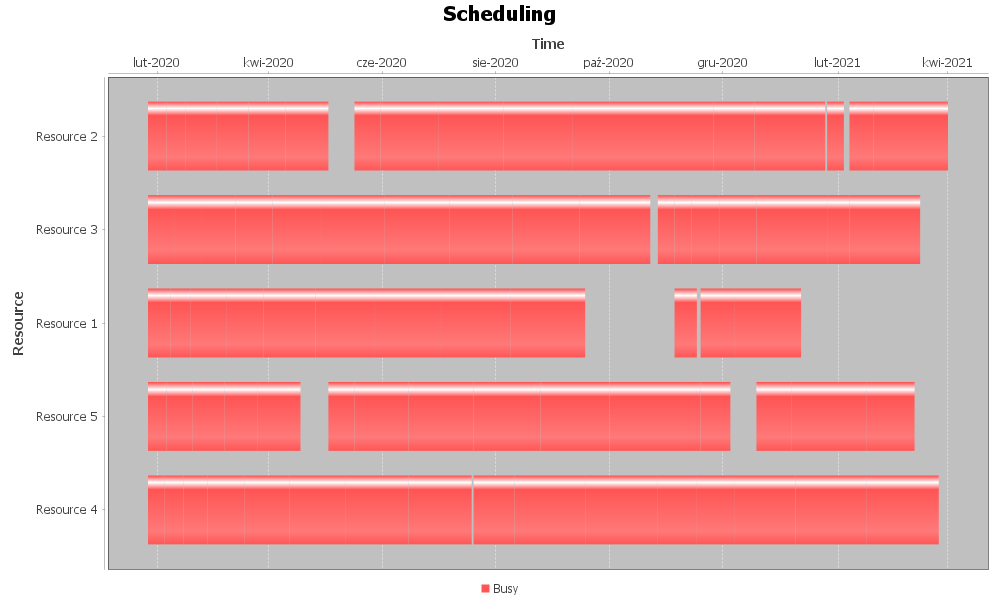
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Metoda obliczeniowa** | **Zbiór danych** | **Min.** | **Śred.** | **Odch. Std.** |
| Algorytm genetyczny | Średnia | 296,56 | 297,86 | 0,98 |
| Algorytm zachłanny | Średnia | 323,98 | 327,78 | 2,42 |
| Przeszukiwanie lokalne | Średnia | 325,91 | 363,72 | 30,38 |
| Symulowane wyżarzanie | Średnia | 296,35 | 297,07 | 0,56 |
| Algorytm genetyczny | 10\_3\_5\_3 | 93,00 | 94,10 | 1,37 |
| Algorytm zachłanny | 10\_3\_5\_3 | 99,00 | 99,00 | 0,00 |
| Przeszukiwanie lokalne | 10\_3\_5\_3 | 93,00 | 105,60 | 7,96 |
| Symulowane wyżarzanie | 10\_3\_5\_3 | 93,00 | 93,00 | 0,00 |
| Algorytm genetyczny | 10\_5\_8\_5 | 80,00 | 80,00 | 0,00 |
| Algorytm zachłanny | 10\_5\_8\_5 | 80,00 | 80,00 | 0,00 |
| Przeszukiwanie lokalne | 10\_5\_8\_5 | 80,00 | 82,30 | 4,06 |
| Symulowane wyżarzanie | 10\_5\_8\_5 | 80,00 | 80,00 | 0,00 |
| Algorytm genetyczny | 10\_7\_10\_7 | 104,00 | 104,00 | 0,00 |
| Algorytm zachłanny | 10\_7\_10\_7 | 104,00 | 104,00 | 0,00 |
| Przeszukiwanie lokalne | 10\_7\_10\_7 | 104,00 | 104,00 | 0,00 |
| Symulowane wyżarzanie | 10\_7\_10\_7 | 104,00 | 104,00 | 0,00 |
| Algorytm genetyczny | 100\_10\_26\_15 | 235,00 | 236,10 | 0,99 |
| Algorytm zachłanny | 100\_10\_26\_15 | 270,00 | 280,30 | 7,56 |
| Przeszukiwanie lokalne | 100\_10\_26\_15 | 272,00 | 308,80 | 37,37 |
| Symulowane wyżarzanie | 100\_10\_26\_15 | 234,00 | 234,10 | 0,32 |
| Algorytm genetyczny | 100\_10\_27\_9\_D2 | 208,00 | 210,50 | 2,17 |
| Algorytm zachłanny | 100\_10\_27\_9\_D2 | 237,00 | 241,30 | 3,06 |
| Przeszukiwanie lokalne | 100\_10\_27\_9\_D2 | 226,00 | 281,10 | 26,57 |
| Symulowane wyżarzanie | 100\_10\_27\_9\_D2 | 207,00 | 208,40 | 0,84 |
| Algorytm genetyczny | 100\_10\_47\_9 | 253,00 | 254,90 | 1,20 |
| Algorytm zachłanny | 100\_10\_47\_9 | 274,00 | 277,70 | 2,31 |
| Przeszukiwanie lokalne | 100\_10\_47\_9 | 272,00 | 299,30 | 29,39 |
| Symulowane wyżarzanie | 100\_10\_47\_9 | 253,00 | 253,70 | 0,48 |
| Algorytm genetyczny | 100\_10\_48\_15 | 244,00 | 246,10 | 1,52 |
| Algorytm zachłanny | 100\_10\_48\_15 | 280,00 | 283,00 | 4,32 |
| Przeszukiwanie lokalne | 100\_10\_48\_15 | 262,00 | 296,60 | 35,46 |
| Symulowane wyżarzanie | 100\_10\_48\_15 | 244,00 | 244,20 | 0,42 |
| Algorytm genetyczny | 100\_10\_64\_9 | 242,00 | 243,90 | 1,10 |
| Algorytm zachłanny | 100\_10\_64\_9 | 292,00 | 298,20 | 3,33 |
| Przeszukiwanie lokalne | 100\_10\_64\_9 | 258,00 | 292,30 | 17,47 |
| Symulowane wyżarzanie | 100\_10\_64\_9 | 242,00 | 242,80 | 0,63 |
| Algorytm genetyczny | 100\_10\_65\_15 | 245,00 | 246,00 | 0,82 |
| Algorytm zachłanny | 100\_10\_65\_15 | 295,00 | 299,70 | 3,47 |
| Przeszukiwanie lokalne | 100\_10\_65\_15 | 254,00 | 310,40 | 36,85 |
| Symulowane wyżarzanie | 100\_10\_65\_15 | 243,00 | 244,20 | 0,63 |
| Algorytm genetyczny | 100\_20\_22\_15 | 127,00 | 128,90 | 1,45 |
| Algorytm zachłanny | 100\_20\_22\_15 | 157,00 | 159,40 | 1,43 |
| Przeszukiwanie lokalne | 100\_20\_22\_15 | 162,00 | 185,20 | 22,75 |
| Symulowane wyżarzanie | 100\_20\_22\_15 | 126,00 | 127,30 | 0,95 |
| Algorytm genetyczny | 100\_20\_23\_9\_D1 | 172,00 | 172,00 | 0,00 |
| Algorytm zachłanny | 100\_20\_23\_9\_D1 | 172,00 | 172,00 | 0,00 |
| Przeszukiwanie lokalne | 100\_20\_23\_9\_D1 | 172,00 | 224,40 | 45,34 |
| Symulowane wyżarzanie | 100\_20\_23\_9\_D1 | 172,00 | 172,00 | 0,00 |
| Algorytm genetyczny | 100\_20\_46\_15 | 165,00 | 167,00 | 1,76 |
| Algorytm zachłanny | 100\_20\_46\_15 | 191,00 | 196,90 | 3,14 |
| Przeszukiwanie lokalne | 100\_20\_46\_15 | 193,00 | 214,90 | 20,05 |
| Symulowane wyżarzanie | 100\_20\_46\_15 | 161,00 | 163,70 | 2,36 |
| Algorytm genetyczny | 100\_20\_47\_9 | 125,00 | 125,60 | 0,84 |
| Algorytm zachłanny | 100\_20\_47\_9 | 164,00 | 168,40 | 2,07 |
| Przeszukiwanie lokalne | 100\_20\_47\_9 | 164,00 | 200,10 | 29,76 |
| Symulowane wyżarzanie | 100\_20\_47\_9 | 124,00 | 125,10 | 0,88 |
| Algorytm genetyczny | 100\_20\_65\_15 | 205,00 | 205,00 | 0,00 |
| Algorytm zachłanny | 100\_20\_65\_15 | 205,00 | 205,00 | 0,00 |
| Przeszukiwanie lokalne | 100\_20\_65\_15 | 205,00 | 229,30 | 23,51 |
| Symulowane wyżarzanie | 100\_20\_65\_15 | 205,00 | 205,00 | 0,00 |
| Algorytm genetyczny | 100\_20\_65\_9 | 126,00 | 128,30 | 1,77 |
| Algorytm zachłanny | 100\_20\_65\_9 | 149,00 | 153,20 | 1,93 |
| Przeszukiwanie lokalne | 100\_20\_65\_9 | 171,00 | 191,00 | 17,69 |
| Symulowane wyżarzanie | 100\_20\_65\_9 | 125,00 | 125,50 | 0,53 |
| Algorytm genetyczny | 100\_5\_20\_9\_D3 | 387,00 | 388,30 | 1,06 |
| Algorytm zachłanny | 100\_5\_20\_9\_D3 | 428,00 | 433,10 | 3,41 |
| Przeszukiwanie lokalne | 100\_5\_20\_9\_D3 | 402,00 | 438,20 | 27,57 |
| Symulowane wyżarzanie | 100\_5\_20\_9\_D3 | 387,00 | 387,90 | 0,32 |
| Algorytm genetyczny | 100\_5\_22\_15 | 485,00 | 487,30 | 1,16 |
| Algorytm zachłanny | 100\_5\_22\_15 | 526,00 | 534,50 | 5,84 |
| Przeszukiwanie lokalne | 100\_5\_22\_15 | 487,00 | 530,40 | 48,26 |
| Symulowane wyżarzanie | 100\_5\_22\_15 | 485,00 | 485,70 | 0,48 |
| Algorytm genetyczny | 100\_5\_46\_15 | 531,00 | 534,30 | 3,65 |
| Algorytm zachłanny | 100\_5\_46\_15 | 620,00 | 630,40 | 5,64 |
| Przeszukiwanie lokalne | 100\_5\_46\_15 | 557,00 | 601,20 | 41,96 |
| Symulowane wyżarzanie | 100\_5\_46\_15 | 529,00 | 532,10 | 4,33 |
| Algorytm genetyczny | 100\_5\_48\_9 | 492,00 | 493,50 | 0,85 |
| Algorytm zachłanny | 100\_5\_48\_9 | 513,00 | 518,50 | 3,57 |
| Przeszukiwanie lokalne | 100\_5\_48\_9 | 509,00 | 526,80 | 15,86 |
| Symulowane wyżarzanie | 100\_5\_48\_9 | 491,00 | 491,60 | 0,52 |
| Algorytm genetyczny | 100\_5\_64\_15 | 482,00 | 484,40 | 1,07 |
| Algorytm zachłanny | 100\_5\_64\_15 | 508,00 | 514,40 | 4,33 |
| Przeszukiwanie lokalne | 100\_5\_64\_15 | 521,00 | 554,20 | 37,88 |
| Symulowane wyżarzanie | 100\_5\_64\_15 | 482,00 | 482,70 | 0,48 |
| Algorytm genetyczny | 100\_5\_64\_9 | 475,00 | 476,80 | 1,23 |
| Algorytm genetyczny | 100\_5\_64\_9 | 475,00 | 476,80 | 1,03 |
| Algorytm zachłanny | 100\_5\_64\_9 | 500,00 | 508,70 | 4,81 |
| Algorytm zachłanny | 100\_5\_64\_9 | 504,00 | 508,30 | 4,19 |
| Przeszukiwanie lokalne | 100\_5\_64\_9 | 480,00 | 521,10 | 33,77 |
| Przeszukiwanie lokalne | 100\_5\_64\_9 | 495,00 | 527,00 | 32,15 |
| Symulowane wyżarzanie | 100\_5\_64\_9 | 475,00 | 475,80 | 0,63 |
| Symulowane wyżarzanie | 100\_5\_64\_9 | 476,00 | 476,00 | 0,00 |
| Algorytm genetyczny | 15\_3\_5\_3 | 230,00 | 230,00 | 0,00 |
| Algorytm zachłanny | 15\_3\_5\_3 | 230,00 | 230,00 | 0,00 |
| Przeszukiwanie lokalne | 15\_3\_5\_3 | 230,00 | 230,00 | 0,00 |
| Symulowane wyżarzanie | 15\_3\_5\_3 | 230,00 | 230,00 | 0,00 |
| Algorytm genetyczny | 15\_6\_10\_6 | 102,00 | 102,00 | 0,00 |
| Algorytm zachłanny | 15\_6\_10\_6 | 102,00 | 102,00 | 0,00 |
| Przeszukiwanie lokalne | 15\_6\_10\_6 | 102,00 | 102,00 | 0,00 |
| Symulowane wyżarzanie | 15\_6\_10\_6 | 102,00 | 102,00 | 0,00 |
| Algorytm genetyczny | 15\_9\_12\_9 | 90,00 | 90,00 | 0,00 |
| Algorytm zachłanny | 15\_9\_12\_9 | 90,00 | 90,00 | 0,00 |
| Przeszukiwanie lokalne | 15\_9\_12\_9 | 90,00 | 90,70 | 2,21 |
| Symulowane wyżarzanie | 15\_9\_12\_9 | 90,00 | 90,00 | 0,00 |
| Algorytm genetyczny | 200\_10\_128\_15 | 461,00 | 463,40 | 2,37 |
| Algorytm zachłanny | 200\_10\_128\_15 | 539,00 | 549,10 | 7,94 |
| Przeszukiwanie lokalne | 200\_10\_128\_15 | 514,00 | 586,80 | 47,64 |
| Symulowane wyżarzanie | 200\_10\_128\_15 | 460,00 | 460,90 | 0,32 |
| Algorytm genetyczny | 200\_10\_135\_9\_D6 | 534,00 | 534,00 | 0,00 |
| Algorytm zachłanny | 200\_10\_135\_9\_D6 | 589,00 | 596,20 | 4,61 |
| Przeszukiwanie lokalne | 200\_10\_135\_9\_D6 | 630,00 | 705,30 | 59,08 |
| Symulowane wyżarzanie | 200\_10\_135\_9\_D6 | 534,00 | 534,00 | 0,00 |
| Algorytm genetyczny | 200\_10\_50\_15 | 486,00 | 486,80 | 0,63 |
| Algorytm zachłanny | 200\_10\_50\_15 | 516,00 | 520,00 | 2,21 |
| Przeszukiwanie lokalne | 200\_10\_50\_15 | 510,00 | 613,10 | 63,08 |
| Symulowane wyżarzanie | 200\_10\_50\_15 | 485,00 | 485,50 | 0,53 |
| Algorytm genetyczny | 200\_10\_50\_9 | 486,00 | 487,10 | 0,74 |
| Algorytm zachłanny | 200\_10\_50\_9 | 501,00 | 503,30 | 1,34 |
| Przeszukiwanie lokalne | 200\_10\_50\_9 | 507,00 | 616,50 | 84,02 |
| Symulowane wyżarzanie | 200\_10\_50\_9 | 486,00 | 486,00 | 0,00 |
| Algorytm genetyczny | 200\_10\_84\_9 | 507,00 | 507,50 | 0,53 |
| Algorytm zachłanny | 200\_10\_84\_9 | 538,00 | 543,20 | 2,86 |
| Przeszukiwanie lokalne | 200\_10\_84\_9 | 541,00 | 621,30 | 67,69 |
| Symulowane wyżarzanie | 200\_10\_84\_9 | 506,00 | 506,80 | 0,63 |
| Algorytm genetyczny | 200\_10\_85\_15 | 474,00 | 475,40 | 0,84 |
| Algorytm zachłanny | 200\_10\_85\_15 | 497,00 | 501,30 | 2,06 |
| Przeszukiwanie lokalne | 200\_10\_85\_15 | 554,00 | 624,40 | 60,16 |
| Symulowane wyżarzanie | 200\_10\_85\_15 | 474,00 | 474,00 | 0,00 |
| Algorytm genetyczny | 200\_20\_145\_15 | 236,00 | 237,40 | 1,51 |
| Algorytm zachłanny | 200\_20\_145\_15 | 280,00 | 286,00 | 2,91 |
| Przeszukiwanie lokalne | 200\_20\_145\_15 | 275,00 | 323,60 | 27,46 |
| Symulowane wyżarzanie | 200\_20\_145\_15 | 236,00 | 236,90 | 0,74 |
| Algorytm genetyczny | 200\_20\_150\_9\_D5 | 900,00 | 900,00 | 0,00 |
| Algorytm zachłanny | 200\_20\_150\_9\_D5 | 900,00 | 900,00 | 0,00 |
| Przeszukiwanie lokalne | 200\_20\_150\_9\_D5 | 900,00 | 912,00 | 21,71 |
| Symulowane wyżarzanie | 200\_20\_150\_9\_D5 | 900,00 | 900,00 | 0,00 |
| Algorytm genetyczny | 200\_20\_54\_15 | 258,00 | 260,30 | 1,57 |
| Algorytm zachłanny | 200\_20\_54\_15 | 296,00 | 300,70 | 3,13 |
| Przeszukiwanie lokalne | 200\_20\_54\_15 | 367,00 | 406,60 | 20,75 |
| Symulowane wyżarzanie | 200\_20\_54\_15 | 257,00 | 258,60 | 0,84 |
| Algorytm genetyczny | 200\_20\_55\_9 | 248,00 | 248,50 | 0,71 |
| Algorytm zachłanny | 200\_20\_55\_9 | 277,00 | 280,50 | 2,46 |
| Przeszukiwanie lokalne | 200\_20\_55\_9 | 296,00 | 337,90 | 23,85 |
| Symulowane wyżarzanie | 200\_20\_55\_9 | 247,00 | 248,00 | 0,67 |
| Algorytm genetyczny | 200\_20\_97\_15 | 336,00 | 336,00 | 0,00 |
| Algorytm zachłanny | 200\_20\_97\_15 | 372,00 | 374,10 | 1,20 |
| Przeszukiwanie lokalne | 200\_20\_97\_15 | 336,00 | 400,70 | 42,22 |
| Symulowane wyżarzanie | 200\_20\_97\_15 | 336,00 | 336,00 | 0,00 |
| Algorytm genetyczny | 200\_20\_97\_9 | 241,00 | 243,20 | 1,55 |
| Algorytm zachłanny | 200\_20\_97\_9 | 273,00 | 276,10 | 2,18 |
| Przeszukiwanie lokalne | 200\_20\_97\_9 | 298,00 | 353,20 | 38,24 |
| Symulowane wyżarzanie | 200\_20\_97\_9 | 241,00 | 242,30 | 0,95 |
| Algorytm genetyczny | 200\_40\_130\_9\_D4 | 513,00 | 513,00 | 0,00 |
| Algorytm zachłanny | 200\_40\_130\_9\_D4 | 513,00 | 513,00 | 0,00 |
| Przeszukiwanie lokalne | 200\_40\_130\_9\_D4 | 513,00 | 539,70 | 48,74 |
| Symulowane wyżarzanie | 200\_40\_130\_9\_D4 | 513,00 | 513,00 | 0,00 |
| Algorytm genetyczny | 200\_40\_133\_15 | 136,00 | 137,60 | 1,51 |
| Algorytm zachłanny | 200\_40\_133\_15 | 179,00 | 182,00 | 1,56 |
| Przeszukiwanie lokalne | 200\_40\_133\_15 | 202,00 | 222,80 | 18,21 |
| Symulowane wyżarzanie | 200\_40\_133\_15 | 139,00 | 140,30 | 0,95 |
| Algorytm genetyczny | 200\_40\_45\_15 | 159,00 | 159,00 | 0,00 |
| Algorytm zachłanny | 200\_40\_45\_15 | 170,00 | 170,90 | 0,57 |
| Przeszukiwanie lokalne | 200\_40\_45\_15 | 188,00 | 221,60 | 20,53 |
| Symulowane wyżarzanie | 200\_40\_45\_15 | 159,00 | 159,00 | 0,00 |
| Algorytm genetyczny | 200\_40\_45\_9 | 138,00 | 141,10 | 1,97 |
| Algorytm zachłanny | 200\_40\_45\_9 | 155,00 | 158,30 | 1,77 |
| Przeszukiwanie lokalne | 200\_40\_45\_9 | 206,00 | 230,40 | 17,61 |
| Symulowane wyżarzanie | 200\_40\_45\_9 | 141,00 | 141,60 | 0,70 |
| Algorytm genetyczny | 200\_40\_90\_9 | 135,00 | 138,30 | 1,70 |
| Algorytm zachłanny | 200\_40\_90\_9 | 165,00 | 169,80 | 2,35 |
| Przeszukiwanie lokalne | 200\_40\_90\_9 | 203,00 | 235,30 | 25,11 |
| Symulowane wyżarzanie | 200\_40\_90\_9 | 137,00 | 139,30 | 1,49 |
| Algorytm genetyczny | 200\_40\_91\_15 | 131,00 | 133,50 | 1,43 |
| Algorytm zachłanny | 200\_40\_91\_15 | 181,00 | 181,90 | 0,74 |
| Przeszukiwanie lokalne | 200\_40\_91\_15 | 213,00 | 242,00 | 26,50 |
| Symulowane wyżarzanie | 200\_40\_91\_15 | 132,00 | 135,00 | 1,63 |

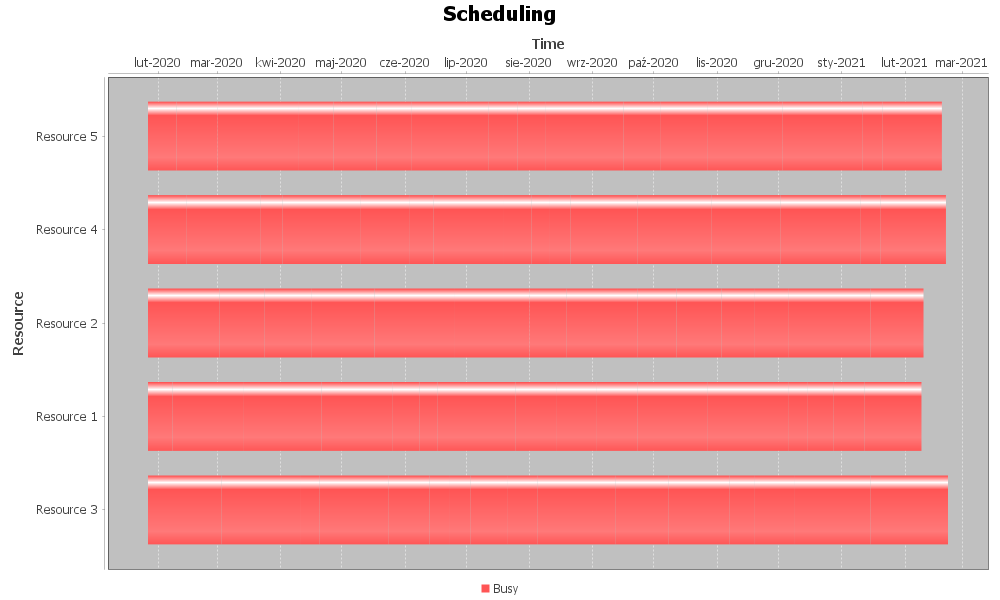
Kilka zdań (2-4) kończących/podsumowujących punkt.

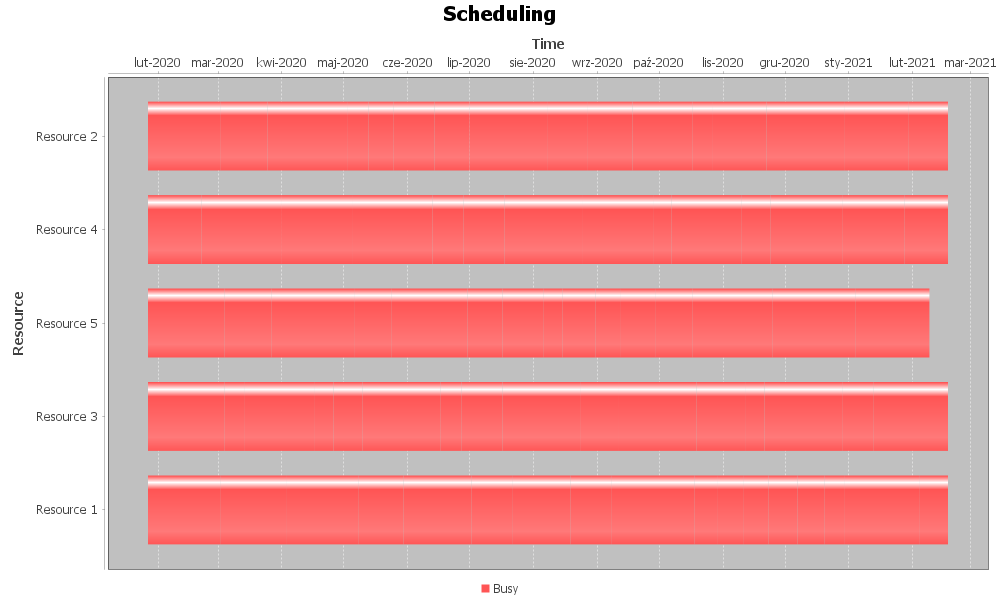
## Wizualizacja wyników badań

Dodatkowo, aby zwizualizować różnice w generowanych rozwiązaniach, przy zastosowaniu programu …. Przeprowadzono ….na rysunkach …. przedstawiono graficznie przykładowe harmonogramy dla metod w następującej kolejności: przeszukiwanie lokalne, algorytm zachłanny, algorytm genetyczny, symulowane wyżarzanie. Ze względu na to, że biblioteka do wykresów, jaką zastosowałem wymagała podania dat, postanowiłem generować je dla dnia dzisiejszego i przyjąć jako jednostkę dzień pracy.









Na tych wykresach wyraźnie widać dlaczego rozwiązania z pierwszej połowy są takie nieoptymalne – występują na nich przerwy, w których dane zasoby nie pracują nad żadnymi zasobami. Widać także, że te z drugiej połowy są bliskie optymalnym, ponieważ nie ma już tutaj praktycznie żadnych przerw, a wszystkie zasoby pracują przez podobny czas.

## Wnioski z przeprowadzonej analizy uzyskanych wyników

Na podstawie przeprowadzonej analizy uzyskanych wyników z przeprowadzonego badania polegającego na …… sformułowano następujące wnioski.

Przeszukiwanie lokalnie osiągnęło zdecydowanie najgorsze rezultaty, z o wiele większym odchyleniem standardowym od reszty – pokazuje to, że jest to niestabilna metoda. Algorytm zachłanny osiągnął podobne rezultaty minimalne, jednakże osiągając o wiele lepsze rezultaty średnie, czyniąc go bardziej niezawodnym.

Poprzednie dwie metody (tzn. przeszukiwanie lokalne i algorytm zachłanny) zostały zaimplementowane głownie w celu porównania ich z dwoma pozostałymi, bardziej skomplikowanymi metodami. Widać wyraźnie, że zarówno algorytm genetyczny, jak i symulowane wyżarzanie, osiągnęły znacznie lepsze rozwiązania, na lekką korzyść drugiego z nich. Biorąc pod uwagę, że średni czas iteracji takiej metody wyniósł lekko ponad minutę, jest to zupełnie akceptowalny czas który użytkownicy mogą poczekać na otrzymanie gotowego rezultatu, który będzie lepszy niż te uzyskane o wiele prostszymi metodami.

# Zakończenie

Po przeanalizowaniu problemu planowania projektów z wieloma wymaganymi umiejętnościami i ograniczonymi zasobami, udało się zdefiniować wymagane elementy problemu, a następnie zaimplementować zaprezentowane w pracy metody generowania rozwiązań. Otrzymane rezultaty pokazują, że zostało to zakończone pomyślnie, gdyż uzyskane w ten sposób harmonogramy są bliskie optymalnym, a czas ich generacji jest akceptowalny do użytku w prawdziwych problemach tego typu.

W dalszym kroku można by było bardziej rozbudować te metody, zaimplementować chociażby więcej operatorów dla algorytmów genetycznych, które być może pozwoliły by na uzyskanie lepszych rezultatów. Można by także spróbować polepszyć prędkość działania aplikacji, a optymalizacja to mogła by być osiągnięta przez ulepszanie ich kodu, zmianę języka programowania, czy też próbę wykorzystania możliwości obliczeniowych jakie umożliwiają karty graficzne.

Można by także zaimplementować i zbadać zupełnie nowe algorytmy, czy metaheurystyki. Zastanawiałem się chociażby nad algorytmami rojowymi – na przykład algorytmem roju pszczół, czy algorytmem roju cząstek. Niestety jednak wymagają one definicję funkcji odległości pomiędzy dwoma rozwiązaniami, co dla mojej reprezentacji problemu nie jest możliwe, ponieważ nie da się zdefiniować różnicy w przypadku kwestii przydziałów różnych zasobów do zadań.

# Literatura

Bryant, R. E., German, S., Velev, M. N. i Murray, N. V. (1999). *Microprocessor Verification Using Efficient Decision Procedures for a Logic of Equality with Uninterpreted Functions.* Springer Berlin Heidelberg. doi:10.1007/3-540-48754-9\_1

Cook, S. (1971). *The complexity of theorem proving procedures.* Proceedings of the Third Annual ACM Symposium on Theory of Computing. doi:10.1145/800157.805047

Fogel, D. i Ghozeil, A. (1997). A note on representations and variation operators. *IEEE Transactions on Evolutionary Computation, 1*(2), strony 159-161. doi:10.1109/4235.687882

He, J., Chen, T. i Yao, X. (2015). On the Easiest and Hardest Fitness Functions. *IEEE Transactions on Evolutionary Computation, 19*(2), 295–305. doi:10.1109/tevc.2014.2318025

Himmelblau, D. M. (1972). *Applied Nonlinear Programming.* McGraw-Hill.

IEEE Standard for Floating-Point Arithmetic. (2008). 1-70. doi:10.1109/IEEESTD.2008.4610935

Keane, A. (1996). A brief comparison of some evolutionary optimization methods. W *Modern Heuristic Search Methods.* John Wiley.

Kirkpatrick, S., Gelatt, C. D. i Vecchi, M. P. (1983, may). Optimization by Simulated Annealing. *Science, 220*(4598), strony 671-680. doi:10.1126/science.220.4598.671

Michalewicz, Z. i Fogel, D. B. (2004). *How to Solve It: Modern Heuristics.* Springer. doi:10.1007/978-3-662-07807-5

Nam, G.-J., Sakallah, K. i Rutenbar, R. (2002). A new FPGA detailed routing approach via search-based Boolean satisfiability. IEEE. doi:10.1109/TCAD.2002.1004311

P.B., M., M., S. i K., S. (2015). A new benchmark dataset for Multi-Skill Resource-Constrained Project Scheduling Problem. W M. Ganzha, L. Maciaszek i M. Paprzycki (Red.), *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems.* *5*, strony 129–138. ACSIS. doi:10.15439/2015F273

Rego, C., Gamboa, D., Glover, F. i Osterman, C. (2011). Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research, 211*(3), 427-441. doi:10.1016/j.ejor.2010.09.010

Sörensen, K. i Glover, F. (2013). Metaheuristics. W *Encyclopedia of Operations Research and Management Science* (strony 960-970). Springer US.

Syswerda, G. (1991). Schedule optimization using genetic algorithms. W *Handbook of Genetic Algorithms* (strony 332–349). Van Nostrand Reinhold.