1. **Wprowadzenie do problemu**
   1. **Opis analizowanego zadania**

Przedmiotem optymalizacji jest dobór tras zadanej liczby wózków widłowych pomiędzy stanowiskami w zakładzie produkcyjnym, których położenie na hali można przybliżyć do punktów w odpowiednio zadanym układzie współrzędnych. Odległość między stanowiskami jest zadana metryką Manhattan. Każde stanowisko ma przypisany czas, jaki wózek widłowy musi poświęcić na jego obsługę (może to być np. wymiana ciężkiego kręgu blachy, która jest półproduktem lub podniesienie ciężkiej części aparatury produkcyjnej w celu dokonania dziennego przeglądu czujników, pobrania materiału do testów, itp.). Założeniem problemu jest, że wózki rozpoczynają pracę na początku układu współrzędnych, czyli w punkcie (0,0) i każdy ma określone tempo wykonywania pracy przy stanowisku. Pomiędzy stanowiskami wózki poruszają się z jednakową prędkością ze względu na ograniczenie prędkości. Celem optymalizacji jest takie dobranie tras wózków widłowych pomiędzy stanowiskami, aby czas pracy ostatniego pracującego wózka był **minimalny**.

Opisany wcześniej problem można zastosować do szeregu podobnych zadań. Jako przykład może posłużyć zadanie możliwie najszybszego dokonania rutynowego przeglądu poprawności działania systemów pomiarowych i innej aparatury w zakładzie produkcyjnym przez zespół automatyków lub metrologów o różnych zdolnościach (i co za tym idzie tempie wykonywania pracy).

* 1. **Przyjęte uproszczenia**

Na potrzeby ułatwienia rozwiązania problemu metodą przybliżoną przyjęto następujące uproszczenia, które nie zmieniają ogólnych założeń zadania:

* Odległość między stanowiskami jest zadana metryką prostokątną (która jest najbardziej sensowna w przypadku hali produkcyjnej).
* Wózki pomiędzy stanowiskami poruszają się z jednakową prędkością, a zużycie paliwa jest takie samo lub nie wpływa na postać rozwiązania. Jedyna istotna różnica występuje podczas obsługi stanowisk (prędkość ich obsługi jest dana współczynnikiem wózka).
* Stanowiska mają równy priorytet obsługi, nie występują kary ani nagrody za obsłużenie ich w jakiejś określonej kolejności.
* Odległość między stanowiskami jest zadana metryką prostokątną, nie są przewidziane awarie wózków, wzajemne się ich blokowanie, a drogi do stanowisk są na tyle szerokie, że wózki mogą się wyminąć.
* Każde stanowisko musi być obsłużone dokładnie raz i nie może być obsługiwane przez dwa wózki jednocześnie (np. w celu przyspieszenia).
* Wszystkie wózki rozpoczynają pracę w początku układu współrzędnych (0,0), który jest garażem i nie ma znaczenia, gdzie skończą pracę.
* Obliczana wartość na podstawie odległości między stanowiskami jest czasem potrzebnym do przejechania i obsługa stanowiska jest również dana czasem, dlatego minimalizacji poddawany jest czas pracy wózków.
* Stanowiska z przypisanym czasem obsługi równym 0 nie trzeba obsługiwać.

# Model matematyczny

* 1. Dane wejściowe

Do danych wejściowych, na bazie których będzie optymalizowany problem należą:

* Liczba wózków widłowych obsługujących halę produkcyjną wraz z ich współczynnikami prędkości. Format danej wejściowej jest następujący:

**α1,2,…,n = [1.2, 0.8, 1.1, 1, 1.3]**

Gdzie każda kolejna wartość oznacza współczynnik wózka widłowego. Liczba wózków jest równa długości wektora, w tym przypadku n = 5. Plik wsadowy jest plikiem tekstowym z wartościami rozdzielonymi znakiem spacji.

* Macierz reprezentująca rozkład stanowisk zakładu, gdzie współrzędne odpowiadają ich położeniu, natomiast wartość jest nominalnym czasem potrzebnym do obsługi stanowiska przez wózek o współczynniku równym 1. Opisywana macierz może mieć przykładowo postać:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***4*** | 0 | 0 | 0 | 0 | K6=50 |
| ***3*** | 0 | K5=30 | 0 | 0 | 0 |
| ***2*** | K4=25 | 0 | 0 | 0 | 0 |
| ***1*** | 0 | 0 | 0 | K3=5 | 0 |
| ***0*** | Miejsce startu | 0 | K1=10 | 0 | K2=15 |
| ***y/x*** | ***0*** | ***1*** | ***2*** | ***3*** | ***4*** |

Gdzie K1,…,6 są stanowiskami wymagającymi obsłużenia z nominalnymi czasami obsługi zawartymi w macierzy.

Plik tekstowy z zawartą macierzą wejściową ma następującą postać:

0 0 10 0 15

0 0 0 5 5

25 0 0 0 0  
0 30 0 0 0

0 0 0 0 50

Gdzie numer wiersza odpowiada całkowitej wartości współrzędnej y, natomiast w jednym wierszu znajdują się kolejne wartości odpowiadające kolejnym całkowitym współrzędnym x.

* 1. Postać rozwiązania

Aby otrzymać postać rozwiązania, którą w łatwy sposób będzie można modyfikować na potrzeby algorytmu, macierz wejściowa zostaje w programie przekonwertowana na inną postać, mianowicie:

Gdzie m jest liczbą stanowisk z niezerowym czasem obsługi (czyli z koniecznością odwiedzenia), natomiast i-ta wartość w wektorze *input* zawiera *numer* stanowiska oraz jego współrzędne i koszt. Stanowiska zostały *ponumerowane* w ten sposób, że zaczynając od punktu (0,0), kolejne elementy wektora *input* odpowiadają stanowiskom z rosnącą współrzędną ***x*** i po osiągnięciu zakresu zwiększana jest współrzędna ***y***. Dla macierzy z wcześniejszego przykładu wektor *input* ma następującą postać:

Przy takiej reprezentacji stanowisk można wprowadzić następującą postać rozwiązania:

Gdzie **0** jest znacznikiem nowego wózka, to znaczy wartości między dwoma znacznikami lub między znacznikiem, a końcem wektora są numerami stanowisk obsłużonych przez dany wózek odpowiadającymi numerom stanowisk we wcześniejszej reprezentacji *input*. Na podstawie takiej postaci rozwiązania można z wektora X i listy *input* stanowiącej o stanowiskach i ich czasach obsłużenia obliczyć funkcję celu. Taka postać rozwiązania została wprowadzona, aby można było na niej przeprowadzać operacje mutacji i krzyżowania typowe dla algorytmów ewolucyjnych.

Przykładowa postać rozwiązania problemu jest permutacją wartości od 1 do m, gdzie m jest liczbą stanowisk, znacznika pierwszego wózka na początku wektora i n-1 znaczników nowego wózka, gdzie n jest liczbą wózków.

* 1. Funkcja celu

Funkcja celu w rozważanym problemie ma być maksymalnym czasem osiągniętym przez wszystkie wózki i jest ona **minimalizowana**.

i = 1,…,n – dostępne wózki widłowe   
j = 1,…,sj – stanowiska obsłużone przez i-ty wózek   
αi – współczynnik szybkości obsługiwania stanowisk przez i-ty wózek   
kij – czas obsługi j-tego stanowiska na liście i-tego wózka   
g(j, j-1) – czas dojazdu do j-tego stanowiska ze stanowiska j-1

# Rozwiązanie problemu algorytmem ewolucyjnym

* 1. Wstęp teoretyczny

W celu znalezienia dobrego rozwiązania rozważanego problemu zastosowano algorytm ewolucyjny w wydaniu klasycznym, czyli niewykorzystujący wiedzy o rozwiązywanym problemie. Algorytmy ewolucyjne przeszukują przestrzeń rozwiązań w celu znalezienia najlepszych. Działanie tych algorytmów wzorowane jest ewolucją biologiczną, gdzie najlepiej przystosowane osobniki (rozwiązania) utrzymają się w populacji i/lub dokonują między sobą krzyżowania/mutacji. W ten sposób populacja rozwiązań problemu z biegiem iteracji jest coraz bardziej przystosowana, to znaczy rozwiązania są lepsze z punktu widzenia problemu. W przypadku naszego zagadnienia, gdzie przestrzeń rozwiązań jest duża już dla małego rozmiaru problemu (około 25!\*15000 rozwiązań dla macierzy wymiaru 5 i 3 wózków) algorytm ewolucyjny wydaje się być rozsądnym wyborem.

* 1. Genotyp i populacja początkowa

Każdy osobnik populacji reprezentuje jakieś rozwiązanie dopuszczalne postawionego problemu oraz posiada zbiór informacji stanowiący genotyp, który jest podstawą do utworzenia fenotypu – ocenianego funkcją oceny. W przypadku rozważanego przez nas problemu genotyp i fenotyp są równoważne, a funkcją oceny jest funkcja celu. Postać genotypu rozwiązania została przedstawiona w punkcie 2.2. i reprezentuje kolejność odwiedzania danych stanowisk przez poszczególne wózki.

Populacja początkowa naszego problemu jest tworzona w sposób całkowicie losowy. Losowy osobnik populacji jest tworzony przez losową permutację liczb z zakresu 1..m (m to liczba „niezerowych” stanowisk) i wstawienie w losowe miejsca tej permutacji n - 1 znaczników nowego wózka (znacznik pierwszego wózka jest zawsze na początku rozwiązania). Optymalny rozmiar populacji jest przedmiotem badania w punkcie *Testy*.

Populacja w ramach działania algorytmu ma stały rozmiar, to znaczy w każdej iteracji nie jest tworzona nowa na bazie poprzedniej, tylko niektóre osobniki są zastępowane nowymi. Takie podejście zmniejsza złożoność obliczeniową problemu.

* 1. Metoda selekcji

Metoda selekcji określa, w jaki sposób zostaną wybrane rozwiązania (osobniki), które wezmą udział w reprodukcji i mutacji. Ważne przy wyborze metody selekcji jest to, aby w gronie rodziców znajdowały się rozwiązania najlepiej przystosowane (z najmniejszą wartością funkcji celu w naszym modelu). W naszym algorytmie zastosowaliśmy do wyboru rodziców **metodę turniejową**. W wybranej losowo w grupie turniejowej osobniki są porównywane względem siebie na podstawie obliczonej i przypisanej do nich wcześniej funkcji celu. Następnie dwoje najlepszych osobników staje się rodzicami i z użyciem **operatora krzyżowania** tworzone jest dwoje potomków (z uwagi na specyfikę problemu potomek jest zawsze bardziej podobny do jednego z rodziców) i lepszy z nich zastępuje najgorszego osobnika w grupie turniejowej. Zaletą takiego podejścia jest to, że można wybrać rodziców bez żadnych informacji statystycznych opisujących populację. Dzięki wykorzystaniu wartości funkcji celu do porównania nie ma potrzeby wyliczania przy każdym porównaniu funkcji przystosowania, co wpływa pozytywnie na złożoność obliczeniową algorytmu.

Rozmiar turnieju jest stały podczas działania algorytmu i stanowi przedmiot badania w punkcie *Testy*.

* 1. Czas działania algorytmu/liczba iteracji

Aby zakończyć działanie algorytmu i podać najlepsze znalezione rozwiązanie można zaimplementować jakieś wymaganie względem otrzymanego rozwiązania, które jeśli jest spełnione, to algorytm kończy działanie i podaje wynik. W naszym modelu mogłaby to być określona z góry lub estymowana wartość funkcji celu, której przekroczenie skutkowałoby zakończeniem działania. Jednak ze względu na specyfikę problemu i zastosowany algorytm niezwykle trudne byłoby podanie takiej wartości. Ponadto algorytm mógłby nigdy jej nie osiągnąć działać w nieskończoność. Możliwe jest również przerwanie pracy w przypadku, gdy do populacji od jakiegoś czasu nie dodawane są lub dodawane jest bardzo mało zmian. W takiej sytuacji zdecydowaliśmy się jednak podnieść współczynnik mutacji i kontynuować działanie, natomiast liczbę iteracji narzucić z góry.

Wyznaczenie optymalnej liczby iteracji algorytmu jest przedmiotem badań w rozdziale *Testy*.

* 1. Operatory krzyżowania

Rdzeniem algorytmu ewolucyjnego jest krzyżowanie osobników lepiej przystosowanych ze sobą. W tym celu zaimplementowaliśmy funkcję realizującą operację krzyżowania dwóch osobników, która bierze od jednego rodzica wycinek genu i wstawia go do genu drugiego rodzica w tym samym miejscu, z którego został wyjęty. Ze względu na permutacyjny charakter rozwiązania dziecko jest bardziej podobne do jednego z rodziców, dlatego przyjęliśmy, że przeżywa to przystosowane lepiej. Poniższy przykład przedstawia sposób, w jaki tworzony jest nowy osobnik.

[0, 2, 3, 4, 0, **9**, **5**, 0, **6**, 8, 7, 1] x [**0**, **8**, **7**, 6, 5, **0**, **3**, **2**, **1**, **0**, 9, **4**] =

[**0**, **8**, **7**, **3**, **9**, **0**, **5**, **6**, **2, 0**, **1**, **4**] – podobny bardziej do rodzica 2

[0, 8, 7, 6, **5**, 0, **3**, **2**, 1, 0, 9, 4] x [**0**, 2, 3, **4**, **0**, **9**, 5, **0**, **6**, **8**, **7**, **1**] =

[**0**, **4**, **9**, **6**, **0**, **5**, **3**, **0**, **2**, **8**, **7**, **1**] – podobny bardziej do rodzica 2

Realizacja operatora jest zrealizowana w ten sposób, że po usunięciu znaczników wybierany jest fragment genu o długości ustalanej programowo w losowym miejscu i w tym samym miejscu wstawiane jest w drugim genomie. Wartości przed i po tym wstawionym fragmencie ustawiane są z pozostałych rodzica drugiego z zachowaniem ich kolejności. Następnie wstawiane są znaczniki nowego wózka w miejscach, w których były u rodzica 2. W ten sposób potomek zachowuje podobieństwo do rodzica 1 pod względem wybranego wycinka i jego miejsca w genie, natomiast znaczniki nowego wózka i kolejności dziedziczy po rodzicu 2.

Szansa na wywołanie krzyżowania dla wybranych rodziców jest stała przez cały czas działania algorytmu i jest przedmiotem badania w punkcie *Testy*.

* 1. Operatory mutacji

Operatory mutacji są ważnym elementem algorytmu ewolucyjnego. Wprowadzają one losowe, drobne zmiany w genie mutującego osobnika. Zapewniają one różnorodność populacji, a ich wpływ w polepszaniu rozwiązania jest szczególnie duży w końcowych iteracjach, gdzie populacja znajduje się w okolicy minimum lokalnego. W zaimplementowanej przez nas wersji algorytmu osobnik jest zastępowany swoją mutacją tylko wtedy, gdy poprawia ona wynik. Takie podejście nie zwiększa aż tak różnorodności, jednak nie występuje szansa zepsucia dobrego rozwiązania. Zdecydowaliśmy się na takie podejście ze względu na specyfikę zagadnienia i typ zastosowanych przez nas operatorów.

* Zamiana całej kolejki jednego wózka z kolejką innego:

Zastosowanie tego operatora wydaje się być dobrym rozwiązaniem ze względu na występowanie w modelu współczynników prędkości wózków. Zastąpienie wózka z długą kolejką innym, szybciej pracującym wózkiem może sprawić, że funkcja celu będzie miała mniejszą wartość. Poniżej zaprezentowane jest działanie opisanego operatora:

[0, **1, 2, 3, 4**, 0, **5, 6, 7**, 0, 8, 9] → [0, **5, 6, 7**, 0, **1, 2, 3, 4**, 0, 8, 9]

* Przesunięcie znacznika nowego wózka w prawo lub w lewo

Ta mutacja zmienia nieznacznie genotyp, jednak może znacząco wpłynąć na funkcję oceny (funkcje celu). Poniżej jej działanie:

[0, 1, 2, 3, **4**, **0**, 5, 6, 7, 0, 8, 9] → [0, 1, 2, 3, **0**, **4**, 5, 6, 7, 0, 8, 9]

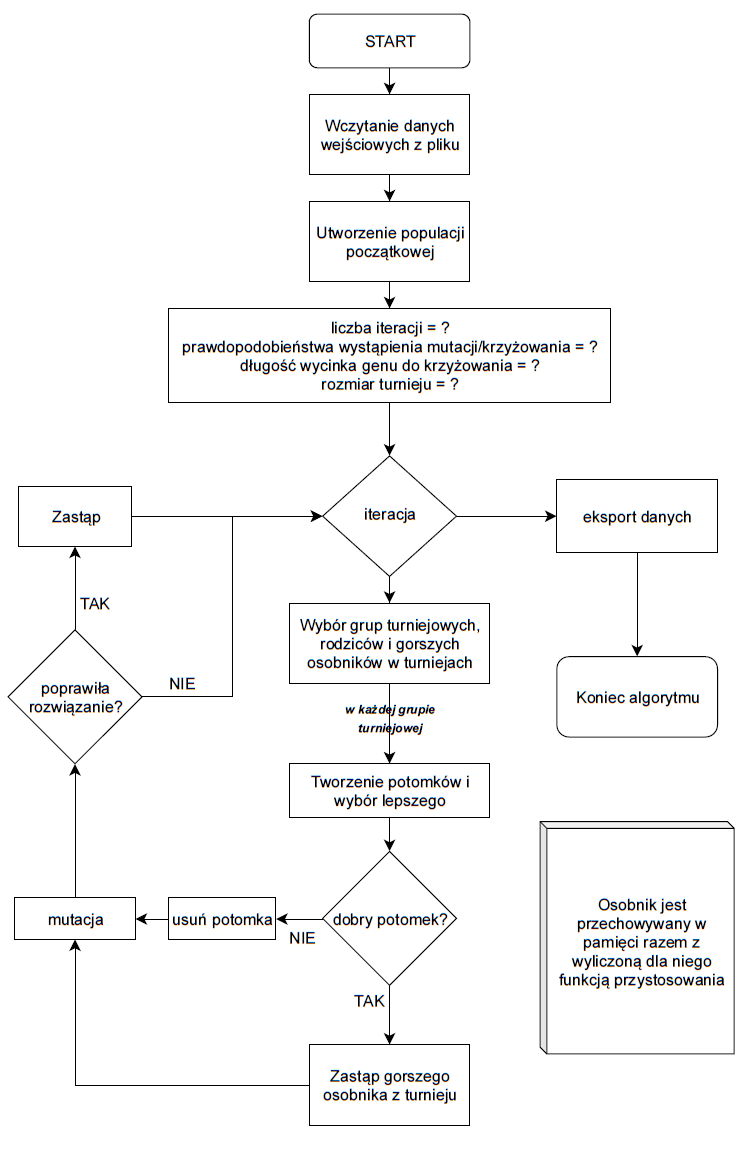
* Losowa permutacja stanowisk w obrębie listy jednego wózka

[0, **1**, **2**, **3**, **4**, 0, 5, 6, 7, 0, 8, 9] → [0, **3**, **2**, **4**, **1**, 0, 5, 6, 7, 0, 8, 9]

Powyższe operatory nie mogą wystąpić jednocześnie, są wprowadzane tylko wtedy, kiedy poprawią wartość rozwiązania, a ich udział we wszystkich mutacjach, jak i ogólne prawdopodobieństwo wystąpienia mutacji jest obiektem analizy w punkcie *Testy*.

Oprócz zastosowania operatorów w algorytmie, zbadaliśmy również wpływ stopniowego zwiększania z biegiem iteracji szansy ich wystąpienia ponad normę oraz gwałtownego podniesienia szansy na mutację w sytuacji, w której od wielu iteracji różnorodność populacji się nie zmienia. Oba te działania pozytywnie wpłynęły zarówno na szybkość osiągnięcia optymalnego rozwiązania, jak i na jego wartość, dlatego wprowadziliśmy je na stałe do naszego algorytmu.

* 1. Uproszczony schemat blokowy



* 1. Oprogramowanie użyte do realizacji projektu

Algorytm został przez nas napisany w języku **Python**, z interpreterem w wersji **3.6.** Wybraliśmy ten język programowania ze względu na dostępny szeroki zakres bibliotek, łatwość programowania oraz to, że jest całkowicie darmowy. Wadą zastosowania tego języka jest jednak to, że jest on wolniejszy w porównaniu do innych języków wysokiego poziomu, jak np. C++. Nie przeszkodziło to jednak w płynnym przeprowadzeniu testów ze względu na nie tak duże rozmiary problemów oraz dyspozycją wydajnego sprzętu. Implementując algorytm korzystaliśmy głównie z biblioteki ***random*** oraz ***math*** pozwalających wprowadzać elementy losowości i bardziej zaawansowanych obliczeń matematycznych.

**Diariusz z podróży rakietą optymalności po oceanie rozwiązań problemu.**