Algorytmu symulowanego wyżarzania

Problem paletowy

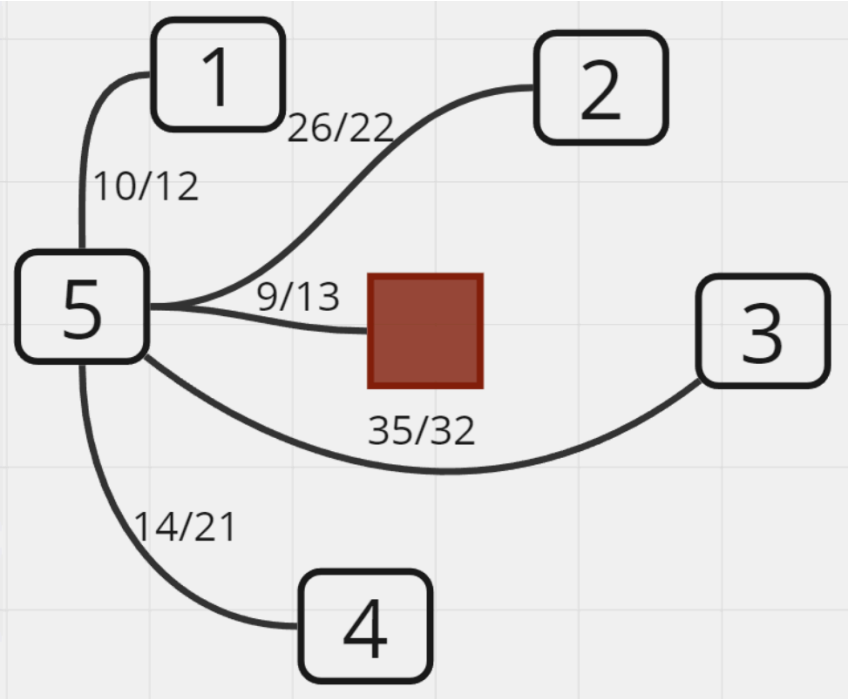
Opracowanie:

Jakub Mieszczak
Marek Mrówka

Wprodzadzenie do problemu

Opis zadania

Zadanie polega na załadowaniu customowej palety z produktami spożywczymi w magazynie, które będą następnie wysłane do sklepu. Pracownik powinien załadować paletę jak najszybciej poruszając się pomiędzy półkami żeby zmaksymalizować wydajność pracy.. Ograniczeniem pracownika jest maksymalny ciężar jaki może nosić – 30 kg. Pracownik może nosić jednocześnie produkty z wielu półek tak długo jak mieści się w określonym ciężarze. Kolejnym ograniczeniem są przerwy pracownika, pracownik musi robić przerwę co każde 5 tysięcy kroków na 2 min.



schemat przykładowej mapy magazynu

Przyjęte uproszczenia

- pracownik pracuje bez przerwy do wykonania zadania
- pracownik odkłada na paletę cały noszony towar
- przejścia między obiektami zawsze zajmują tyle samo czasu
- wszystkie półki mają taki sam priorytet
- nie ma produktów przekraczających swoją masą maksymalny udźwig pracownika
- masa jaką nosi pracownik nie wpływa na jego szybkość przemieszczania się

Teoria

Symulowane wyżarzanie – jedna z technik projektowania algorytmów heurystycznych (metaheurystyka). Cechą charakterystyczną tej metody jest występowanie parametru sterującego zwanego *temperaturą*, który maleje w trakcie wykonywania algorytmu. Im wyższą wartość ma ten parametr, tym bardziej chaotyczne mogą być zmiany. Podejście to

jest inspirowane zjawiskami obserwowanymi w metalurgii – im większa temperatura metalu, tym bardziej jest on plastyczny.

Jest to metoda iteracyjna: najpierw losowane jest pewne rozwiązanie, a następnie jest ono w kolejnych krokach modyfikowane. Jeśli w danym kroku uzyskamy rozwiązanie lepsze, wybieramy je zawsze. Istotną cechą symulowanego wyżarzania jest jednak to, że z pewnym prawdopodobieństwem może być również zaakceptowane rozwiązanie gorsze (ma to na celu umożliwienie wyjście z maksimum lokalnego).

Prawdopodobieństwo przyjęcia gorszego rozwiązania wyrażone jest wzorem $e^{(f(X)-f(X'))/T}$ (rozkład Boltzmanna), gdzie *X* jest poprzednim rozwiązaniem, *X'* nowym rozwiązaniem, a *f* funkcją oceny jakości – im wyższa wartość *f(X)*, tym lepsze rozwiązanie. Ze wzoru można zauważyć, że prawdopodobieństwo przyjęcia gorszego rozwiązania spada wraz ze spadkiem temperatury i wzrostem różnicy jakości obu rozwiązań.

Model matematyczny

$$f(x) = \sum_{1}^{j} d_{ij}(x) + T * P$$

$d_{ij} - czas\ potrzebny\ do\ przebycia\ między\ punktami$ "i" i "j"

$T - czas\ przerwy$

$P - liczba\ przerw\ (\frac{liczba\ kroków}{dopuszczalna\ liczba\ kroków\ między\ przerwami})$

Ograniczenia

$$W \leq W_{max}$$

W – ciężar noszonych przedmiotów

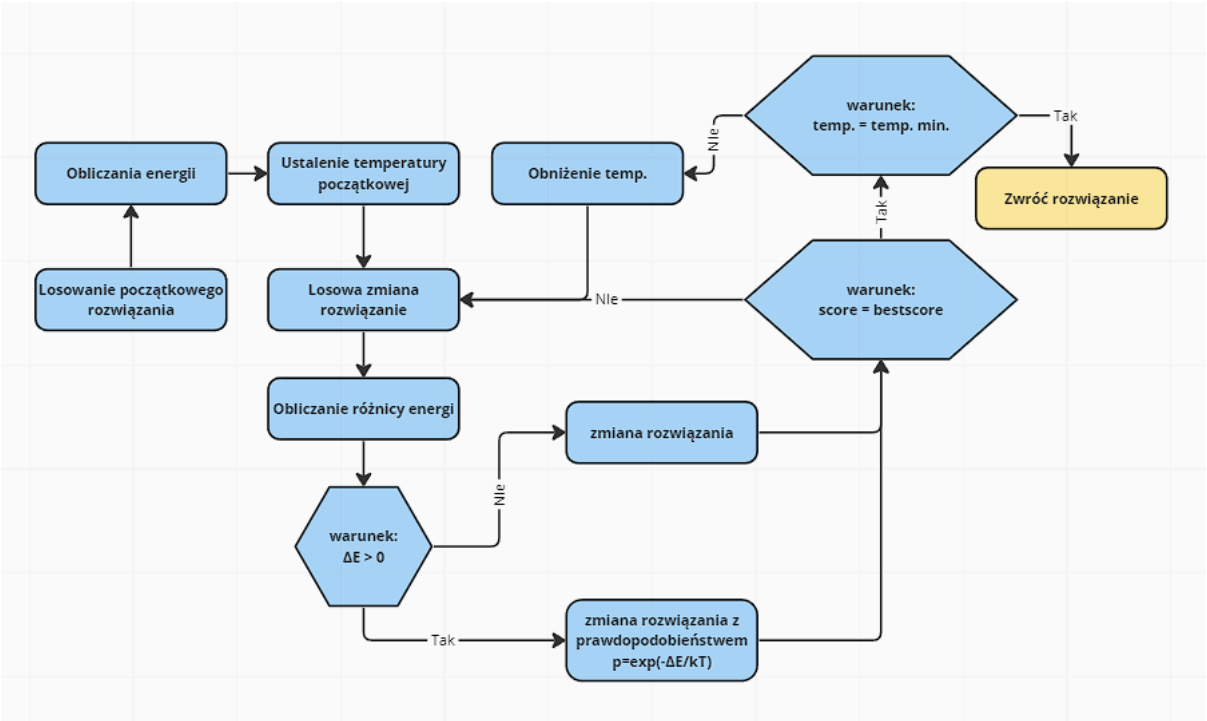
W_{max} – maksymalny dozwolony ciężar

D – liczba wykonanych kroków

W_{max} – liczba kroków między przerwami

$$P \leq \frac{D}{D_{max}}$$

Algorytm



pseudokod zaimplementowanego algorytmu

Dane wejściowe

Reprezentacja ogólna

Przedmiot	Półka	Waga	Ilość
ubrania	1	4	60
jedzenie	3	2	20
AGH	4	7	30
narzędzia	2	4	12
zabawki	5	2	25
napoje	1	3	80
słodycze	3	1	30
książki	2	2	45
elektronika	3	3	20
kosmetyki	1	1	35
biżuteria	4	1	10
gry	2	2	25
instrumenty	5	10	10

przykład produktów do przeniesienia przez pracownika

5					x
4				x	16/25
3			x	13/20	35/40
2		x	7/11	23/32	15/22
1	x	11/20	15/30	22/35	12/8
i/j	1	2	3	4	5

odległości oraz czas poruszania się między półkami

Reprezentacja w kodzie

```
# Neighbor matrix of path: {start point : {end point: [time, steps]}
neighbor_matrix={0 : {0 : [0, 0],
                      1 : [10, 10],
                      2 : [12, 12],
                      3 : [11, 11],
                      4 : [9, 9],
                      5 : [8, 8]},
                 1 :  {0 : [10, 0],
                       1 : [0, 0],
                       2 : [11, 20],
                       3 : [15, 30],
                       4 : [22, 25],
                       5 : [12, 8]},
                 2 :  {0 : [12, 0],
                       1 : [11, 20],
                       2 : [0, 0],
                       3 : [7, 11],
                       4 : [22, 35],
                       5 : [12, 8]},
                 3 :  {0 : [11, 0],
                       1 : [15, 30],
                       2 : [7, 11],
                       3 : [0, 0],
                       4 : [13, 20],
                       5 : [35, 40]},
                 4 :  {0 : [9, 0],
                       1 : [22, 35],
                       2 : [23, 32],
                       3 : [13, 20],
                       4 : [0, 0],
                       5 : [16, 25]},
                 5 :  {0 : [8, 0],
                       1 : [12, 8],
                       2 : [15, 22],
                       3 : [35, 40],
                       4 : [16, 25],
                       5 : [0, 0]}}
```

przykładowe połączenia pomiędzy węzłami

```
stock_status={
    1 : [[60, 4], [80, 3], [35, 1]],
    2 : [[12, 4], [45, 2], [25, 2]],
    3 : [[20, 2], [30, 1], [20, 3]],
    4 : [[30, 7], [10, 1]],
    5 : [[25, 2], [10, 10]]}
```

przykładowa liczba kroków i czas przebycia trasy pomiędzy węzłami

Opis zadania

W aplikacji zaimplementowaliśmy trzy różne ułożenia magazynu do dynamicznego wyboru przez użytkownika. Pierwsze ułożenie jest prostym, opartym o 5 półek w magazynie oraz niewielką ilość przedmiotów na nich, przeciętny czas losowej ścieżki mieści się w granicach około 5000 sekund. Drugie ułożenie opiera się na wiele półek z dużą ilością przedmiotów, tutaj czas potrzebny na pokonanie ścieżki jest liczone w dziesiątkach tysięcy. Ostatnie ułożenie jest generowane losowe, losowa jest zarówno liczba półek, przedmiotów na nich jak i czas przejścia między nimi oraz liczba kroków między półkami.

Parametry

Poniższe parametry decydują w jaki sposób wygląda problem oraz jak zostanie rozwiązany:

- max_weight - maksymalny ciężar noszony przez pracownika
- temperature - początkowa temperatura występująca w algorytmie
- współczynnik o jaki początkowa temperatura będzie się zmniejszała w kolejnych iteracjach algorytmu
- min_temperature - temperatura po której osiągnięciu algorytm ma się zakończyć
- max_step - maksymalna liczba kroków po której pracownik musi wykonać przerwę
- break_time - czas trwania przerwy w sekundach
- n_vertex - liczba wierzchołków jaka będzie zamieniana w pewnym otoczeniu przy generowaniu nowej ścieżki, można też wybierać tą liczbę przy pomocy aktualnej temperatury

```
# Parameters
max_weight=30 #maximum weight that can be carried by the employee
temperature=100 #starting temperature
alpha=0.99 #alpha parameter
min_temperatura=1 #target temperature
max_step=5000 #maximum number of steps before the mandatory break
break_time=100 #break time
n_vertex=2 #the number of vertices that are randomly selected
```

domyślne parametry algorytmu

Funkcja generująca pierwszą ścieżkę

Funkcja generuje pierwszą przykładową ścieżkę która w algorytmie SA będzie poprawiana w kolejnych iteracjach. Zbiera ona informacje na temat danych wejściowych (liczbę półek oraz przedmiotów na nich), następnie w sposób losowy tworzymy połączenia między wszystkimi pułkami z których zabieramy po jednym przedmiocie tak żeby nic na nich nie zostało. Kolejno sprawdzamy jaki ciężar był noszony po podniesieniu każdego przedmiotu, tam gdzie ciężar przekroczył maksymalny udźwig wstawiliśmy w ścieżce powrót do palety żeby odłożyć przedmioty, w ten sposób nasze kryteria były spełnione. Na sam koniec zwracamy pierwszą ścieżkę oraz liczbę jej wierzchołków.

```
def generate_first_path(): #generationg first random path
    # sum all items on all shelfs
    all_items=[]
    for key, value in stock_status.items():
        for elem in value:
            for el in range(elem[0]):
                all_items.append((key,elem[1]))
    # generate random path between every item
    random_points=random.sample(all_items,len(all_items))
    first_path=[]
    cur_weight=0
    for elem in random_points:
        cur_weight=cur_weight+elem[1]
        # if possible take next item
        if cur_weight<=max_weight:
            first_path.append([elem[0],cur_weight])
        # else put down items
        else:
            cur_weight=0
            first_path.append([0,cur_weight])
            cur_weight=cur_weight+elem[1]
            first_path.append([elem[0],cur_weight])
    return first_path, len(all_items)
```

funkcja do generowania pierwszej ścieżki

Funkcja generująca następne ścieżki

Ten kod generuje kolejne ścieżki w oparciu o istniejącą, wprowadzając określone zmiany. Zaczynamy od odtworzenia ścieżki bez odkładania przedmiotów na paletę ponieważ to odkładanie może się zmienić jeśli po drodze zmienimy odwiedzany wierzchołki. Następnie dokonujemy zmian w ścieżce zgodnie z listą indeksów wierzchołków do zmiany. Po dokonaniu zmiany jeszcze raz sprawdzamy w którym miejscu ścieżki przekraczamy maksymalny ciężar i wracamy do palety.

```
def generate_next_path(cur_path, list_vertex_id): #generating next path,
changing cur_path by list_vertex_id
    cur_path_info=[]
    cur_path_info=[[cur_path[0][0],cur_path[0][1]]]
    # get only path without putting down items
    for id, elem in enumerate(cur_path[1:]):
        if cur_path[id+1]!= [0, 0]:
            if cur_path[id]!= [0, 0]:

cur_path_info.append([cur_path[id+1][0],cur_path[id+1][1]-cur_path[id][1]])
            else:
                cur_path_info.append([cur_path[id+1][0],cur_path[id+1][1]])
    # change vertex by provided list
    for elem in list_vertex_id:
        cur_path_info[elem[0]],
cur_path_info[elem[1]]=cur_path_info[elem[1]], cur_path_info[elem[0]]
#changing vertex
    next_path=[]
    cur_weight=0
    #taking items
    for elem in cur_path_info:
        # if possible take next item
        cur_weight=cur_weight+elem[1]
        if cur_weight<=max_weight:
            next_path.append([elem[0],cur_weight])
        # else put down items
        else:
            cur_weight=0
            next_path.append([0,cur_weight])
            cur_weight=cur_weight+elem[1]
            next_path.append([elem[0],cur_weight])

    return next_path
```

funkcje do generowania kolejnych ścieżek

Funkcja sumująca liczbę kroków oraz czas przejścia między półkami

Zaimplementowana została funkcja sumująca czas oraz liczbę kroków wykonanych przez pracownika. Jest ona potrzebna podczas porównywania poprzedniej oraz kolejnej ścieżki obliczając wartość funkcji celu.

```
def steps_time(path): #number of steps in path, time spend by walking
(without breaks)
    n_steps=0
    s_time=0
    #sum steps and time
    for id, point in enumerate(path[:-1]):
        if path[id][0]!=path[id+1][0]:
            n_steps=n_steps+neighbor_matrix[path[id][0]][path[id+1][0]][1]
            s_time=s_time+neighbor_matrix[path[id][0]][path[id+1][0]][0]
    return {"Liczba kroków:" : n_steps, "Całkowity czas:" : s_time}
```

funkcja obliczająca liczbę kroków i całkowity czas podróży nieuwzględniający przerw

Funkcja celu

Funkcja celu obliczająca za pomocą całkowitej liczbie kroków oraz całkowitemu czasowi. Obliczamy to dodając całkowity czas do dzielenia bez reszty liczby kroków przez dopuszczalną liczbę kroków bez odbycia przerwy pomnożona przez czas przerwy.

```
def funkcja_celu(droga, czas): #calculate cost
    return czas+(droga//max_step)*break_time
```

funkcja celu

Funkcja decydująca o zmianach w aktualnej ścieżce

Funkcja generująca listę indeksów wierzchołków które należy zmienić. Losujemy wartość z przedziału 2 - n_vertex. Jest też możliwość żeby losować wartości z przedziału 2 - int(temperature) + 2 w tym przypadku poprawa wartości w algorytmie SA na początku jest bardzo wolna i wzrasta wraz z zmniejszającą się temperaturą. Wynika z tego że najlepszą poprawę otrzymujemy zmieniając jednocześ 2 wierzchołki między sobą.

```
def random_vertex(temperature, n_items): #generate random list of random
vertex
    # n_vertex=random.randint(2, int(temperature+2))
    n_vertex=random.randint(2, int(n_vertex))
    lista=[random.sample(range(0, int(n_items)), 2) for elem in
range(n_vertex)]
    return lista
```

funkcja generująca losowe wierzchołki które mają zostać zamienione miejscami

Funkcja wyświetlająca wykres

Funkcja wyświetlająca wykres zmian wartości najlepszego wyniku w czasie.

```
def plot_data(x, y): #plot output
    plt.plot(x, y)
    plt.title("SA algorithm")
    plt.xlabel("Path number")
    plt.ylabel("Score")
    plt.show()
```

funkcja do wyświetlania wykresów

Implementacja algorytmu SA

Algorytm działa podobnie do każdego innego algorytmu SA, jest jednak implementowany aby działać dla naszego konkretnego problemu. Poniższym algorytmie kolejno:

- tworzymy pierwszą ścieżkę która w dalszej fazie będzie poddawania zmianą
- obliczamy dla niej pierwszą wartość oraz dodatnie jej do listy najlepszych wartości
- tworzenie odpowiednich list potrzebnych do późniejszego wyświetlenia wykresu
- następnie tak długo ja aktualna temperatura jest większa od minimalnej temperatury dopuszczalnej:
    - generujemy losowe indeksy wierzchołków które w kolejnej ścieżkę zamienią się miejscami
    - tworzymy nową ścieżkę z zmienionymi określonymi wierzchołkami
    - obliczamy liczbę kroków oraz czas podróży pracownika
    - obliczamy funkcję celu
    - jeśli nowy wynik jest lepszy od poprzedniego lub warunek losowy oparty o temperaturę na to pozwala aktualizujemy aktualny najleszy wynik, dodajemy do wykresu oraz ustalamy że nowa ścieżka jest aktualną ścieżką
    - w przeciwnym wypadku odrzucamy nową ścieżkę
    - aktualizujemy aktualną temperaturę poprzez pomnożenie jej przez parametr alpha
- po osiągnięciu minimalnej dopuszczalnej temperatury zwracamy najlepszą aktualną ścieżkę oraz najlepszy wynik algorytmu

```
def simulated_annealing(temperature, alpha, min_temperatura): #implementation
os SA function
    first_path, n_items=generate_first_path() #generating first path
    aktualna=steps_time(first_path)['Liczba kroków:']
    best_score=999999 # initialazing best_score param
    cur_path=first_path #setting first path as current path
    new_score=steps_time(cur_path) #calculating new_score
    new_score=funkcja_celu(new_score['Liczba kroków:'],new_score['Całkowity
czas:'])

    # ploting
    all_scores=[new_score]
    all_iterations=[0]
    acc=1
```

```
    while temperature>min_temperatura: #while currtent temperature is higher
then minimal temperature
        vertex_id_change=random_vertex(temperature, n_items) #generate IDs of
vertex to change
        new_path = generate_next_path(cur_path,vertex_id_change) #generate
next path
        new_score=steps_time(new_path) #calculate score
        new_score=funkcja_celu(new_score['Liczba
kroków:'],new_score['Całkowity czas:'])
        delta_dlugosc = new_score - funkcja_celu(steps_time(cur_path)['Liczba
kroków:'],steps_time(cur_path)['Całkowity czas:']) # calculate delta

        if delta_dlugosc < 0 or random.uniform(0, 1) <
math.exp(-delta_dlugosc / (temperature)): #if new score is better then
previos or random state is True - add new best score
            cur_path = new_path
            acc=acc+1
            all_scores.append(new_score) #add new score
            all_iterations.append(acc)
            if best_score>new_score: #if current best score is better then
previos
                best_score=new_score #set new best score
            temperature = temperature * alpha #change temperature by alpha

    plot_data(all_iterations, all_scores) #plot results
    return cur_path, best_score #return path and best score
```

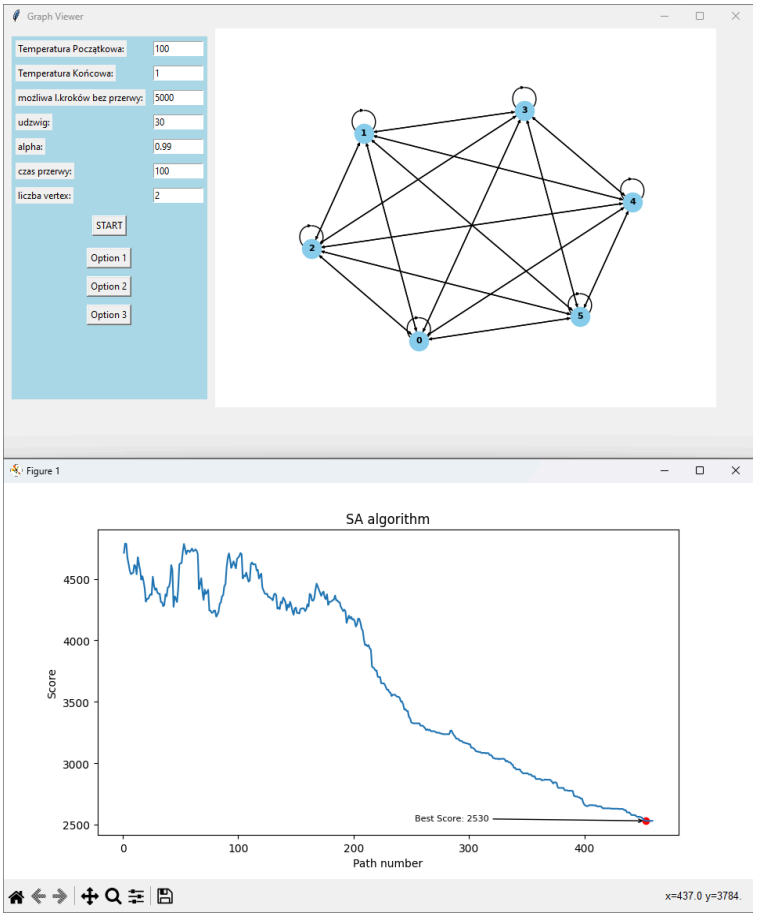
Implementacja algorytmu SA do zadanego problemu

Aplikacja

Aplikacja została przygotowana w taki sposób, aby do pierwszego uruchomienia z domyślnymi parametrami wystarczyło wcisnąć klawisz “START”. Użytkownika ma możliwość zmiany parametrów:

- temperatury początkowej oraz końcowej
- liczba kroków wykonywana przez pracownika po której musi wykonać przerwę
- czas przerwy w sekundach
- udźwig pracownika
- współczynnik alpha
- liczba zmienianych wektorów podczas zmiany rozwiązania

Program aplikacji zwraca rozwiązanie graficzne - wykres rozwiązań w czasie, możliwe jest wypisanie trasy do przejścia w konsoli.



wygląd programu

Testy działania

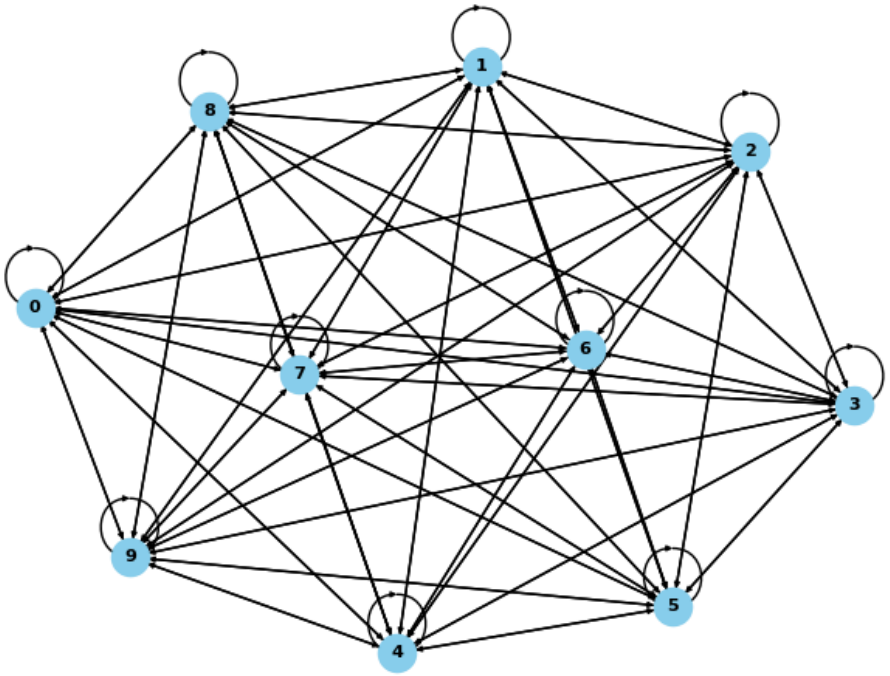
Tworzenie projektu podzieliliśmy na etapy, wykonując odpowiednie różne rodzaje testów::

1. Stworzenie algorytmu dla przykładowego zestawu danych.
   Po wykonaniu pierwszej iteracji kodu przeprowadziliśmy funkcjonalne testowanie modułu, dostarczając nowe dane i weryfikując czy zachowanie algorytmu jest zgodne co do oczekiwanych rezultatów.
   Charakter testu w pierwszej części projektu miało charakter biało skrzynkowy, pozwoliło to na bieżące korygowanie nieprawidłowości.
2. Stworzenie Panelu użytkownika do zmiany parametrów oraz Stworzenie Panelu wyświetlającego wykres działania algorytmu.
   Następnym etapem, było zaprojektowanie interakcji użytkownika. Umieściliśmy odpowiednie zmienne oraz przyciski funkcyjne w aplikacji. Przeprowadziliśmy wyłącznie testy akceptacyjne modułu, których skutkiem było umieszczenie kolejnych zmiennych w panelu.
3. Integracja Panelu z algorytmem
   Przeprowadziliśmy testy integracyjne pomiędzy modułami.
   a. Sprawdziliśmy przekazywanie zmiennych z panelu do algorytmu

    b. Sprawdziliśmy tworzenie poglądowego schematu węzłów na podstawie wyboru przycisku - option
    c. Zweryfikowaliśmy odpowiednie przedstawianie wyników działania algorytmu w formie graficznej.
4. Testy akceptacyjne
    a. Przeprowadziliśmy całościowe testy systemowe i akceptacyjne działania aplikacji, poprzez testowanie działania algorytmu korzystając z UI aplikacji. Stworzyliśmy tablice testów.
    b. Testy rozpoczęliśmy

Wyniki działania

Dla wszystkich scenariuszy przyjęliśmy przedstawione na schemacie węzły oraz rozłożenie przedmiotów na pułkach zawarte w kodzie.



schemat węzłów

```
def opcja1():
    global stock_status, neighbor_matrix
    stock_status = {
    1: [[80, 4], [80, 3], [35, 1], [45, 5], [20, 2], [30, 1], [20, 3], [40, 4], [15, 2], [25, 2]],
    2: [[12, 4], [45, 2], [25, 2], [30, 3], [15, 1], [35, 4], [22, 5], [18, 2], [10, 3], [28, 4]],
    3: [[20, 2], [30, 1], [20, 3], [40, 4], [15, 2], [25, 2], [12, 1], [33, 3], [18, 4], [26, 2]],
    4: [[30, 7], [10, 1], [22, 5], [18, 2], [10, 3], [28, 4], [15, 2], [25, 2], [12, 4], [45, 2]],
    5: [[25, 2], [10, 10], [30, 3], [15, 1], [35, 4], [22, 5], [18, 2], [45, 2], [25, 2], [12, 4]],
    6: [[60, 3], [20, 2], [40, 4], [15, 2], [25, 2], [12, 1], [33, 3], [18, 4], [26, 2], [80, 4]],
    7: [[15, 1], [35, 4], [22, 5], [18, 2], [10, 3], [28, 4], [30, 1], [20, 3], [40, 4], [45, 5]],
    8: [[18, 2], [10, 3], [28, 4], [15, 2], [25, 2], [12, 4], [45, 2], [25, 2], [30, 3], [15, 1]],
    9: [[22, 5], [18, 2], [10, 3], [28, 4], [15, 2], [25, 2], [12, 4], [45, 2], [25, 2], [10, 10]]
    }
```

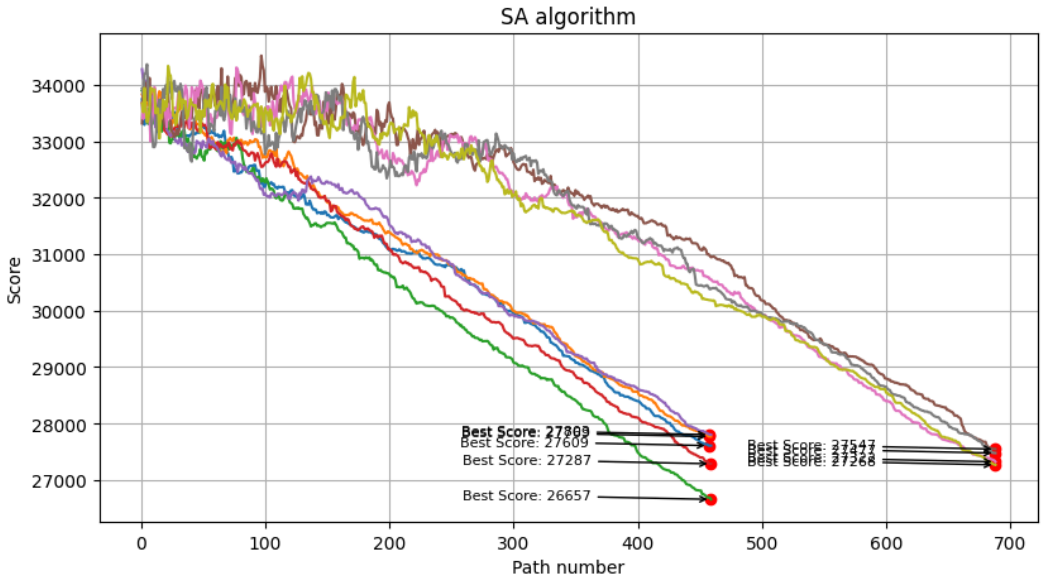
półki

Scenariusz 1

wpływ zmiany temperatury początkowej na rozwiązanie.

scenariusz 1	temperatura	min. temperatura	alpha	max. udźwig	max. liczba kroków	l. zmiany wektorów	czas przerwy

						vertex	
Przykład 1	100/1000	1	0.99	30	5000	2	100



wykres 1.1

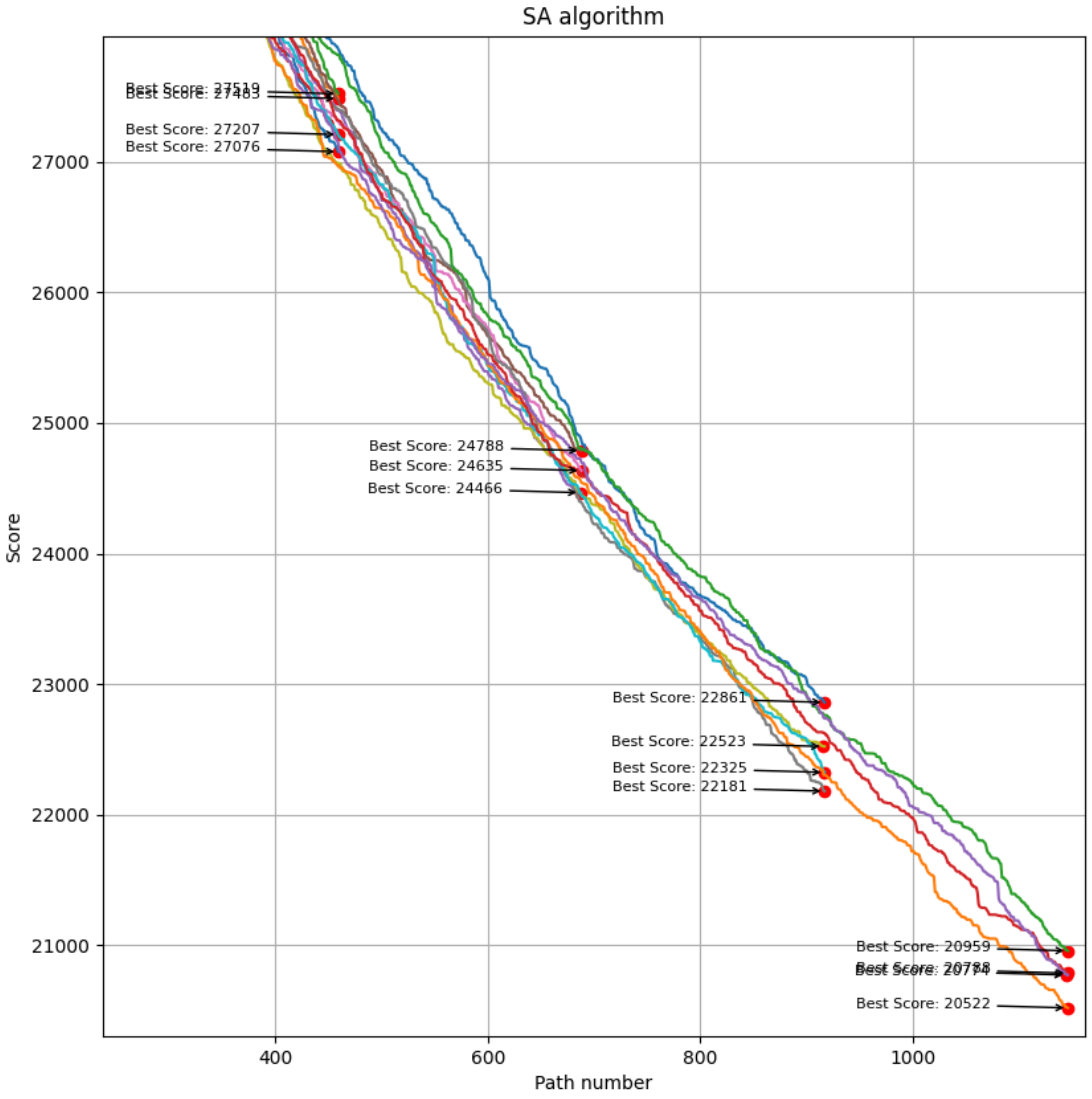
temperatura: 100	temperatura: 1000
Minimalna długość trasy: 26621 Elapsed Time: 20.884956121444702 seconds Minimalna długość trasy: 27769 Elapsed Time: 27.589223384857178 seconds Minimalna długość trasy: 26657 Elapsed Time: 23.567301273345947 seconds Minimalna długość trasy: 27287 Elapsed Time: 21.869060516357422 seconds	Elapsed Time: 27.511443853378296 seconds Minimalna długość trasy: 27547 Elapsed Time: 28.99459719657898 seconds Minimalna długość trasy: 27322 Elapsed Time: 26.636009216308594 seconds Minimalna długość trasy: 27477 Elapsed Time: 25.98638343811035 seconds Minimalna długość trasy: 27268
średnia długość trasy: 27 083,5	średnia długość trasy: 27 403,5
średni czas obliczeń: 23,47 [s]	średni czas obliczeń: 27,25 [s]

Na podstawie przykładu, można zaobserwować negatywny wpływ zwiększenia temperatury początkowej na wydajność znajdowania rozwiązań przez algorytm. Przy wyższej temperaturze początkowej średni czas obliczeń oraz długość trasy uległy pogorszeniu. Na podstawie wykresu można zaobserwować, że w przypadku parametru: 1000 algorytm zbyt długo dopuszcza znaczne pogorszenie rozwiązania. Przy okolicach 300 - iteracji algorytm zaczyna “podążanie” w kierunku minimalnego rozwiązania, podczas gdy dla mniejszego parametru ten proces rozpoczyna się już w okolicach 150- iteracji. Algorytm z większym parametrem wykonał więcej iteracji, co zwiększyło czas obliczeń, ale nie spowodowało poprawy rozwiązań.
Na podstawie pomiarów nie umieszczonych w sprawozdaniu, zaobserwowaliśmy, że ustawienie temperatury początkowej powyżej 100 dla zestawów parametrów podobnych do opisanych ma negatywny wpływ na wydajność znajdowania rozwiązań przez algorytm.

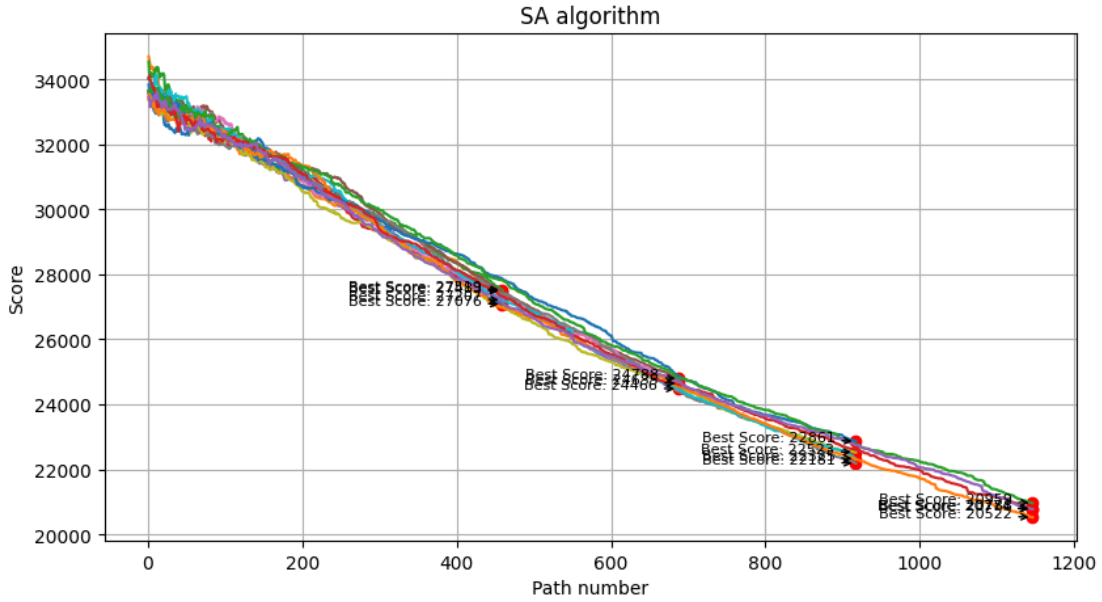
Scenariusz 2

wpływ zmiany temperatury końcowej na rozwiązanie.

scenariusz 2	temperatura	min. temperatura	alpha	max. udźwig	max. liczba kroków	l. zmiany wektorów vertex	czas przerwy
Przykład 2	100	1/0.1/0.01/ 0.001	0.99	30	5000	2	100



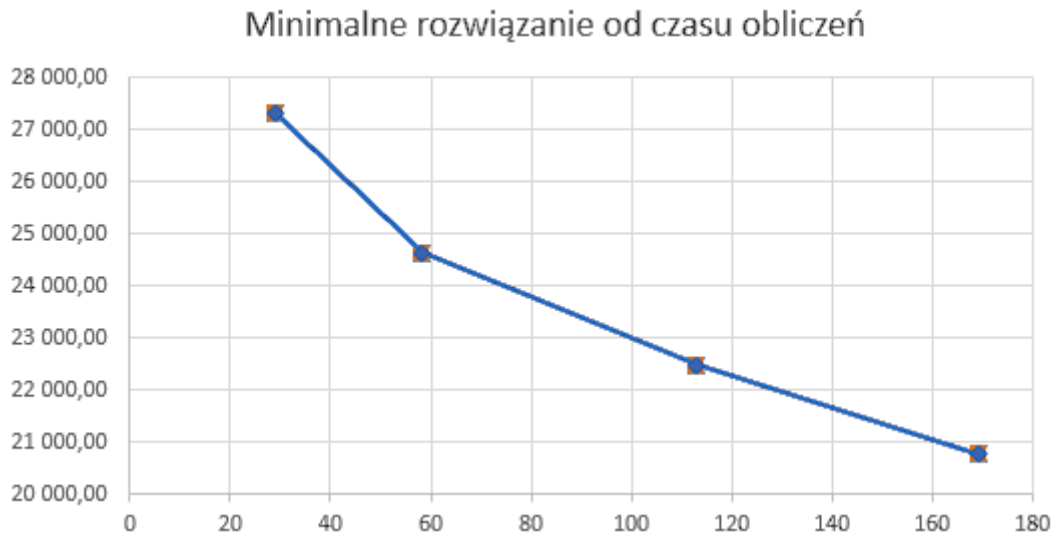
wykres 2.1



wykres 2.2

min. temperatura: 1	min. temperatura: 0.1
Elapsed Time: 25.85597062110901 seconds Minimalna długość trasy: 27076 Elapsed Time: 30.679415225982666 seconds Minimalna długość trasy: 27483 Elapsed Time: 36.15631604194641 seconds Minimalna długość trasy: 27519 Elapsed Time: 23.53501582145691 seconds Minimalna długość trasy: 27207	Elapsed Time: 54.217973947525024 seconds Minimalna długość trasy: 24466 Elapsed Time: 58.94709873199463 seconds Minimalna długość trasy: 24788 Elapsed Time: 61.63309192657471 seconds Minimalna długość trasy: 24635
średnia długość trasy: 27 321,25	średnia długość trasy: 24 629,67
średni czas obliczeń: 29,06 [s]	średni czas obliczeń: 58,27 [s]
min. temperatura: 0.01	min. temperatura: 0.001
Elapsed Time: 105.4277777671814 seconds Minimalna długość trasy: 22181 Elapsed Time: 134.44754576683044 seconds Minimalna długość trasy: 22523 Elapsed Time: 118.4098744392395 seconds Minimalna długość trasy: 22325 Elapsed Time: 94.12714195251465 seconds Minimalna długość trasy: 22861	Elapsed Time: 192.3413565158844 seconds Minimalna długość trasy: 20522 Elapsed Time: 166.24411916732788 seconds Minimalna długość trasy: 20959 Elapsed Time: 165.91505241394043 seconds Minimalna długość trasy: 20788 Elapsed Time: 152.85058784484863 seconds Minimalna długość trasy: 20774
średnia długość trasy: 22 472,5	średnia długość trasy: 20 760,75
średni czas obliczeń: 113,1 [s]	średni czas obliczeń: 169,34 [s]

Obniżenie temperatury końcowej wpływa na przedłużenie działania algorytmu w późniejszym etapie działania. Prawdopodobieństwo losowej zmiany rozwiązania staje się coraz mniejsze przez obniżenie temperatury. Algorytm znajduje lepsze rozwiązania kosztem zwiększonego nakładu obliczeń. Można zaobserwować poprawę rozwiązania przedstawioną na wykresie.

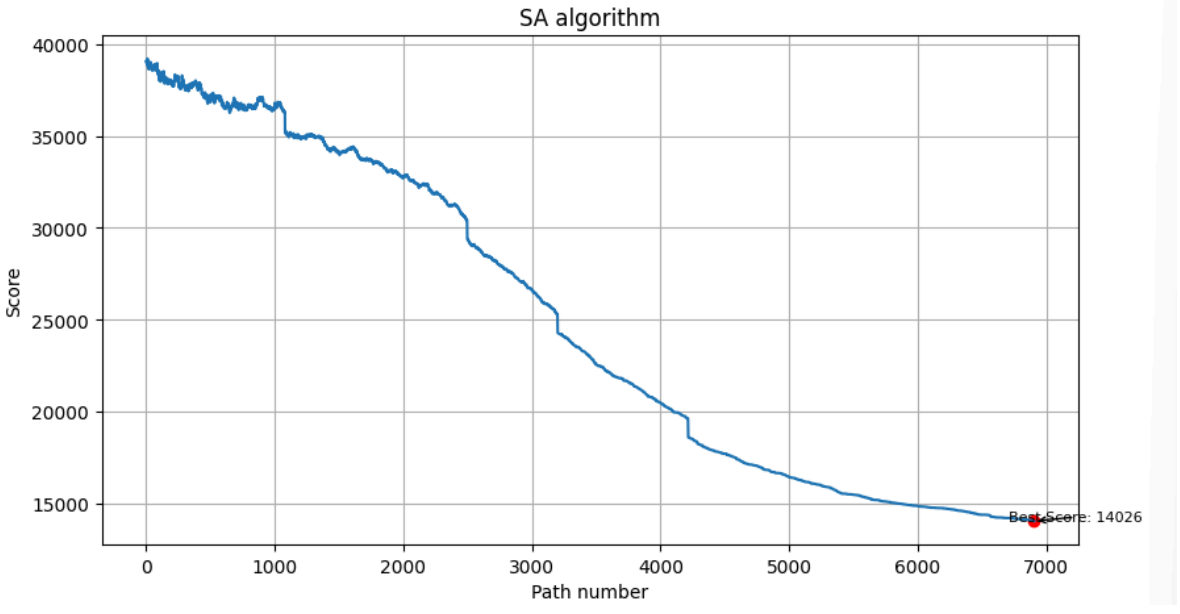


wykres 2.3

Dodatkowo poprawa rozwiązania nie zachodzi w nieskończoność. Można zaobserwować na wykresie 2.4 wykonanego dla innego zestawu parametrów, że poprawa rozwiązania w czasie staje się coraz wolniejsza.

scenariusz 2	temperatura	min. temperatura	alpha	max. udźwig	max. liczba kroków	l. zmiany wektorów vertex	czas przerwy
Przykład 3	100	0.01	0.999	30	5000	2	1000

Elapsed Time: 5572.209220409393 seconds Minimalna długość trasy: 14026

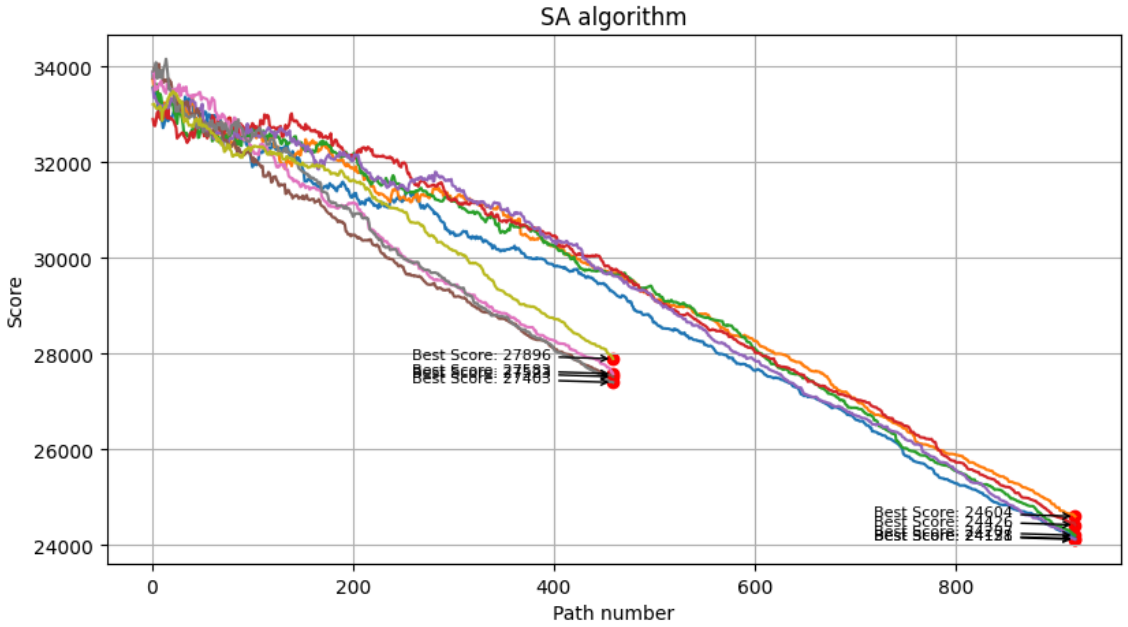


wykres 2.4

Scenariusz 3

wpływ zmiany współczynnika alpha na rozwiązanie.

scenariusz 3	temperatura	min. temperatura	alpha	max. udźwig	max. liczba kroków	l. zmiany wektorów vertex	czas przerwy
Przykład 4	100	1	0.99/0.995	30	5000	2	100



wykres 3.1

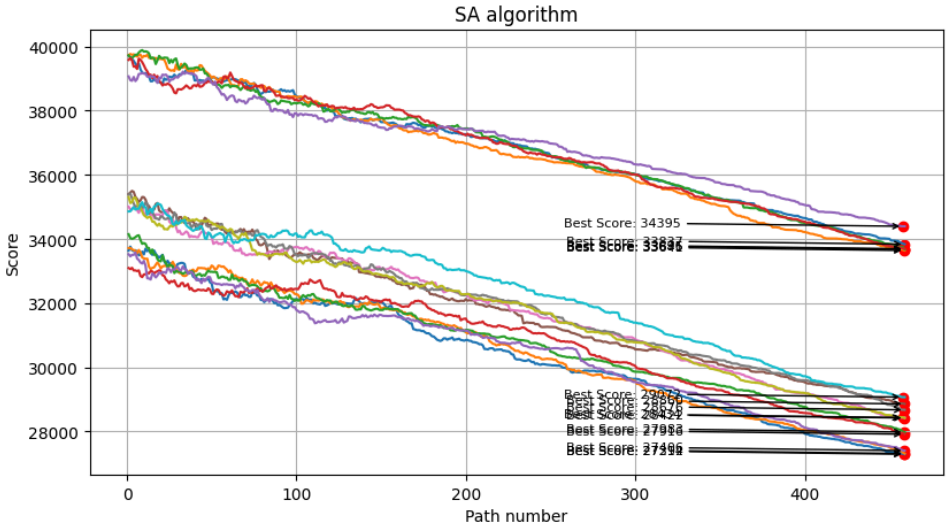
alpha: 0.995	alpha: 0.99
Elapsed Time: 77.92458391189575 seconds Minimalna długość trasy: 24138 Elapsed Time: 66.67519950866699 seconds Minimalna długość trasy: 24604 Elapsed Time: 62.74939775466919 seconds Minimalna długość trasy: 24207 Elapsed Time: 63.555673122406006 seconds Minimalna długość trasy: 24426	Elapsed Time: 26.10110902786255 seconds Minimalna długość trasy: 27523 Elapsed Time: 20.828898429870605 seconds Minimalna długość trasy: 27583 Elapsed Time: 23.214759588241577 seconds Minimalna długość trasy: 27403 Elapsed Time: 19.485361099243164 seconds Minimalna długość trasy: 27896
średnia długość trasy: 27 083,5	średnia długość trasy: 27 403,5
średni czas obliczeń: 67,72 [s]	średni czas obliczeń: 27,25 [s]

Czas obliczeń dla mniejszego współczynnika alpha był krótszy. Temperatura była szybciej obniżana co sprawiło, że gorsze rozwiązania były rzadziej występujące i w podobnej liczbie iteracji szybciej osiągnięto minimalizację, za cenę większej szansy na wybór minimum lokalnego.

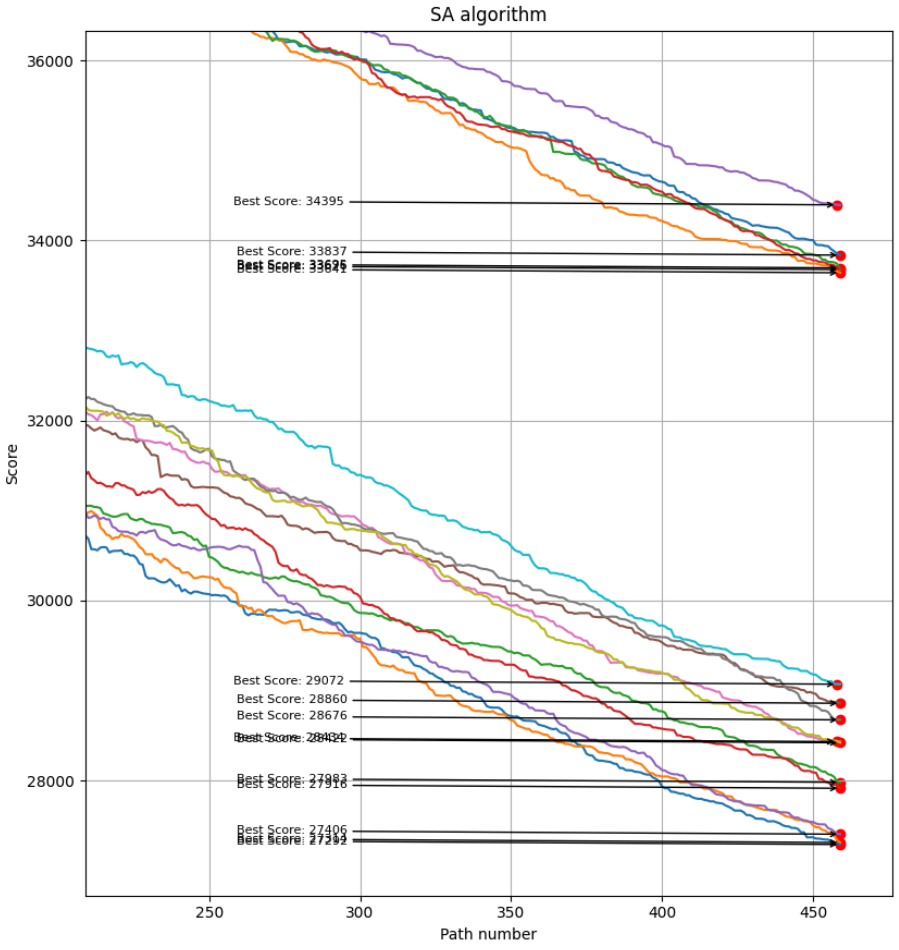
Scenariusz 4

wpływ zmiany współczynnika max. udzwig na rozwiązanie.

scenariusz 4	temperatura	min. temperatura	alpha	max. udźwig	max. liczba kroków	l. zmiany wektorów vertex	czas przerwy
Przykład 5	100	1	0.99	10/20/30	5000	2	100



wykres 4.1



wykres 4.2

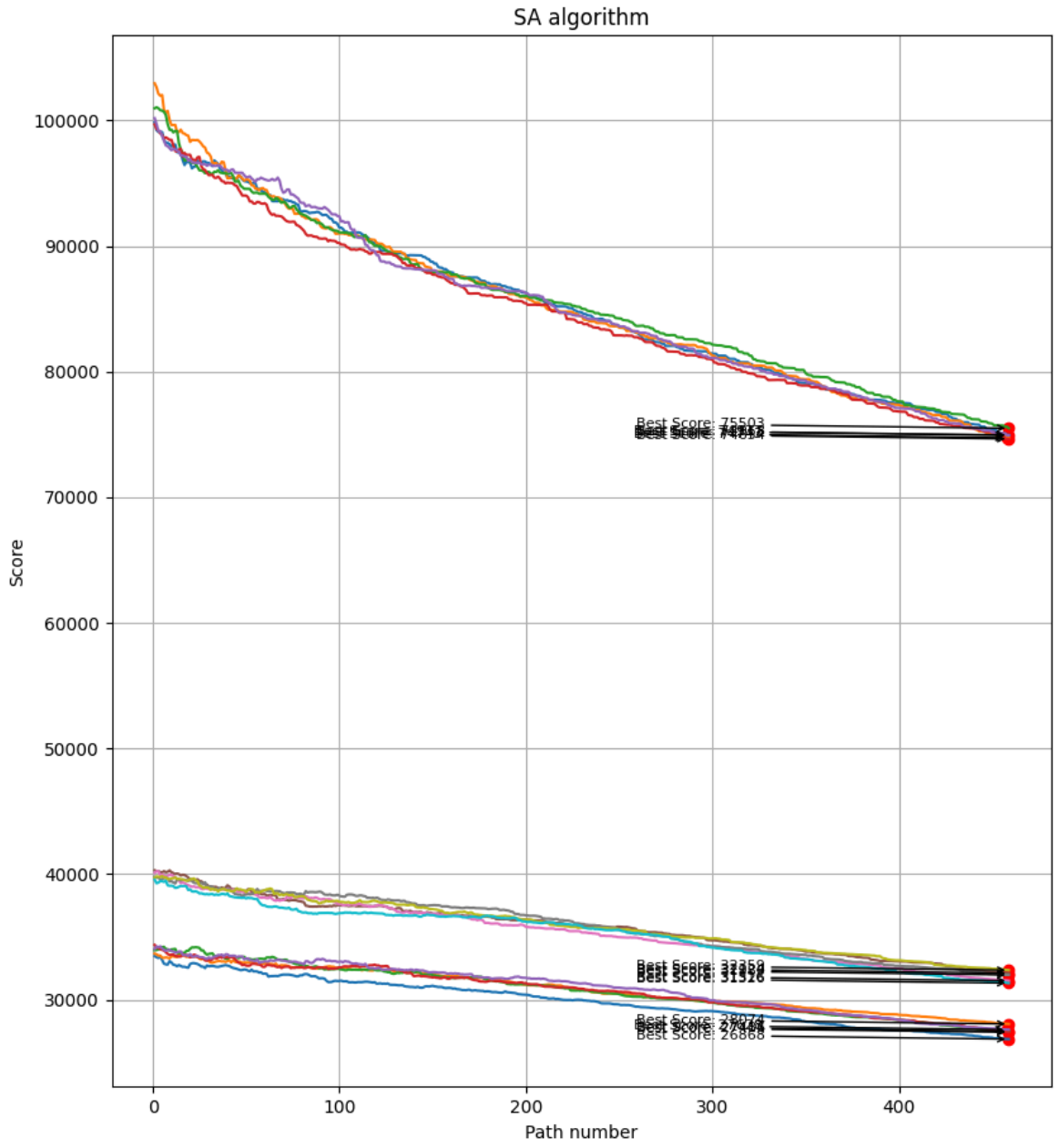
max. udźwig: 10	max. udźwig: 20	max. udźwig: 30
Elapsed Time: 19.4296817779541 seconds Minimalna długość trasy: 34221 Elapsed Time: 20.240243196487427 seconds Minimalna długość trasy: 33641 Elapsed Time: 20.491721868515015 seconds Minimalna długość trasy: 33676 Elapsed Time: 19.36230182647705 seconds Minimalna długość trasy: 33695 Elapsed Time: 17.7212553024292 seconds Minimalna długość trasy: 34395	Elapsed Time: 25.33977246284485 seconds Minimalna długość trasy: 28860 Elapsed Time: 23.320800304412842 seconds Minimalna długość trasy: 28434 Elapsed Time: 21.15684723854065 seconds Minimalna długość trasy: 28676 Elapsed Time: 23.680185317993164 seconds Minimalna długość trasy: 28422 Elapsed Time: 20.481062650680542 seconds Minimalna długość trasy: 29072	Elapsed Time: 22.115771532058716 seconds Minimalna długość trasy: 27292 Elapsed Time: 20.93751311302185 seconds Minimalna długość trasy: 27314 Elapsed Time: 19.643566131591797 seconds Minimalna długość trasy: 27983 Elapsed Time: 18.863018035888672 seconds Minimalna długość trasy: 27916 Elapsed Time: 22.07667851448059 seconds Minimalna długość trasy: 27406
średnia długość trasy: 33 925,6	średnia długość trasy: 28 692,8	średnia długość trasy: 27 582,2
średni czas obliczeń: 19,44 [s]	średni czas obliczeń: 22,79 [s]	średni czas obliczeń: 20,73 [s]

Podczas zwiększania parametru obserwowane jest polepszenie wyniku minimalizacji. Jest to zgodne z założeniami. Pracownik, który może przenieść większą przedmiotów może wykonać mniej kroków w krótszym czasie.

Scenariusz 5

wpływ zmiany współczynnika max. liczba kroków na rozwiązanie.

scenariusz 5	temperatura	min. temperatura	alpha	max. udźwig	max. liczba kroków	l. zmiany wektorów vertex	czas przerwy
Przykład 6	100	1	0.99	30	50/500/5000	2	100



wykres 5.1

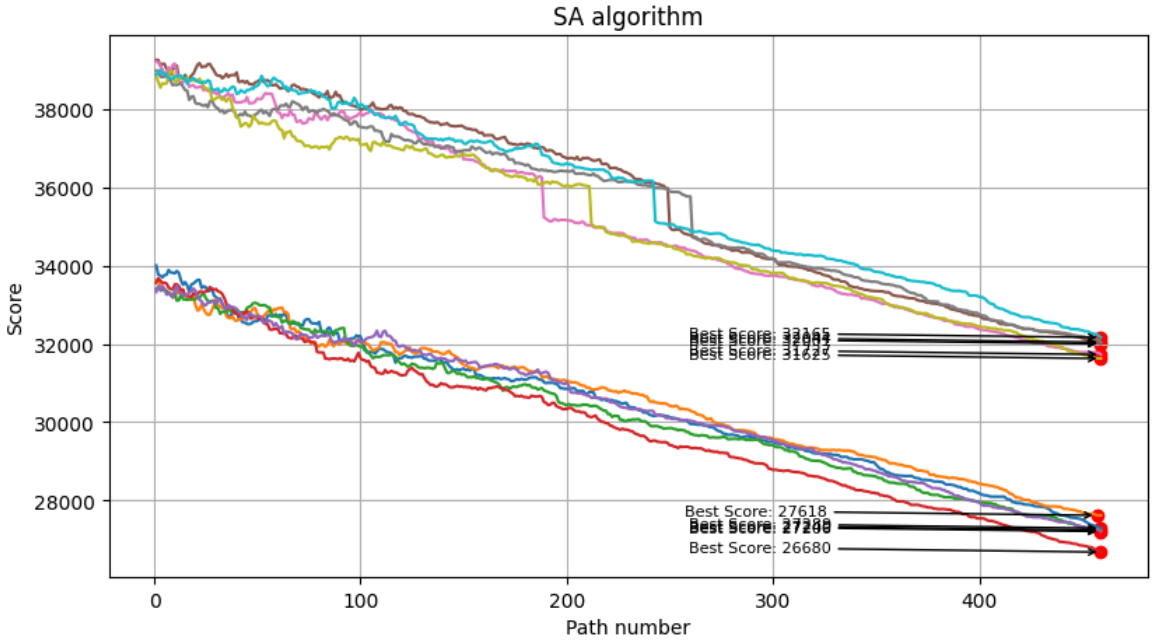
max. liczba kroków 50	max. liczba kroków 500	max. liczba kroków 5000
Elapsed Time: 43.65671372413635 seconds Minimalna długość trasy: 76156 Elapsed Time: 40.25183653831482 seconds Minimalna długość trasy: 74743 Elapsed Time: 37.202476978302 seconds Minimalna długość trasy: 75503 Elapsed Time: 47.56446933746338 seconds Minimalna długość trasy: 74654 Elapsed Time: 39.88501834869385 seconds Minimalna długość trasy: 74937	Elapsed Time: 26.510663747787476 seconds Minimalna długość trasy: 32124 Elapsed Time: 25.752484798431396 seconds Minimalna długość trasy: 31516 Elapsed Time: 24.132275104522705 seconds Minimalna długość trasy: 31980 Elapsed Time: 21.930818796157837 seconds Minimalna długość trasy: 32359 Elapsed Time: 26.11539387702942 seconds Minimalna długość trasy: 31326	Elapsed Time: 25.612193822860718 seconds Minimalna długość trasy: 26868 Elapsed Time: 18.63398790359497 seconds Minimalna długość trasy: 28074 Elapsed Time: 23.387192964553833 seconds Minimalna długość trasy: 27486 Elapsed Time: 21.53399968147278 seconds Minimalna długość trasy: 27414 Elapsed Time: 20.186485528945923 seconds Minimalna długość trasy: 27628
średnia długość trasy: 75 350,6	średnia długość trasy: 31 861	średnia długość trasy: 27 494
średni czas obliczeń: 41,71 [s]	średni czas obliczeń: 24,89 [s]	średni czas obliczeń: 21,87 [s]

Podobnie jak w przypadku maksymalnego udźwigu zwiększenie parametru upraszcza działanie algorytmu.

Scenariusz 6

wpływ zmiany czas przerwy na rozwiązanie.

scenariusz 6	temperatura	min. temperatura	alpha	max. udźwig	max. liczba kroków	l. zmiany wektorów vertex	czas przerwy
Przykład 7	100	1	0.99	30	5000	2	100/1000

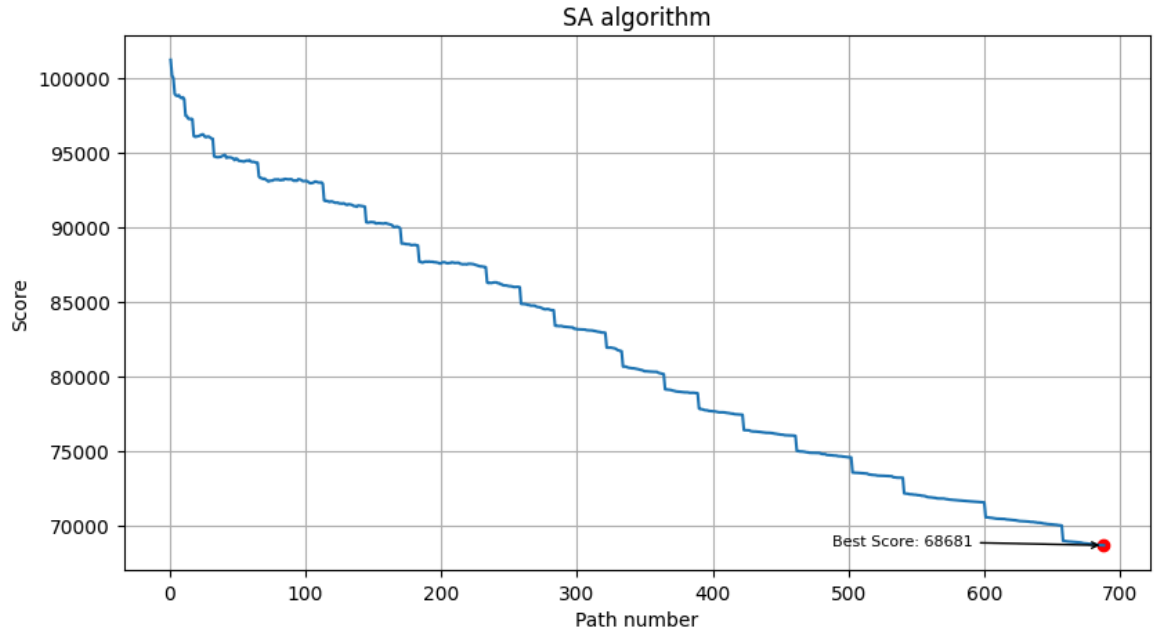


wykres 6.1

czas przerwy: 100	czas przerwy: 1000
Elapsed Time: 19.967219352722168 seconds Minimalna długość trasy: 27341 Elapsed Time: 20.97133159637451 seconds Minimalna długość trasy: 27618 Elapsed Time: 26.027530670166016 seconds Minimalna długość trasy: 27240 Elapsed Time: 23.803950309753418 seconds Minimalna długość trasy: 26680 Elapsed Time: 23.64582061767578 seconds Minimalna długość trasy: 27206	Elapsed Time: 22.84904670715332 seconds Minimalna długość trasy: 32001 Elapsed Time: 23.87616467475891 seconds Minimalna długość trasy: 31727 Elapsed Time: 22.161442041397095 seconds Minimalna długość trasy: 32044 Elapsed Time: 25.581143856048584 seconds Minimalna długość trasy: 31625 Elapsed Time: 20.415771961212158 seconds Minimalna długość trasy: 32165
średnia długość trasy: 27 217	średnia długość trasy: 31 912
średni czas obliczeń: 22,88 [s]	średni czas obliczeń: 22,98 [s]

scenariusz 6	temperatura	min. temperatura	alpha	max. udźwig	max. liczba kroków	l. zmiany wektorów vertex	czas przerwy
Przykład 8	100	0.01	0.995	30	5000	2	1000

Elapsed Time: 473.9123389720917 seconds Minimalna długość trasy: 18223



wykres 6.2

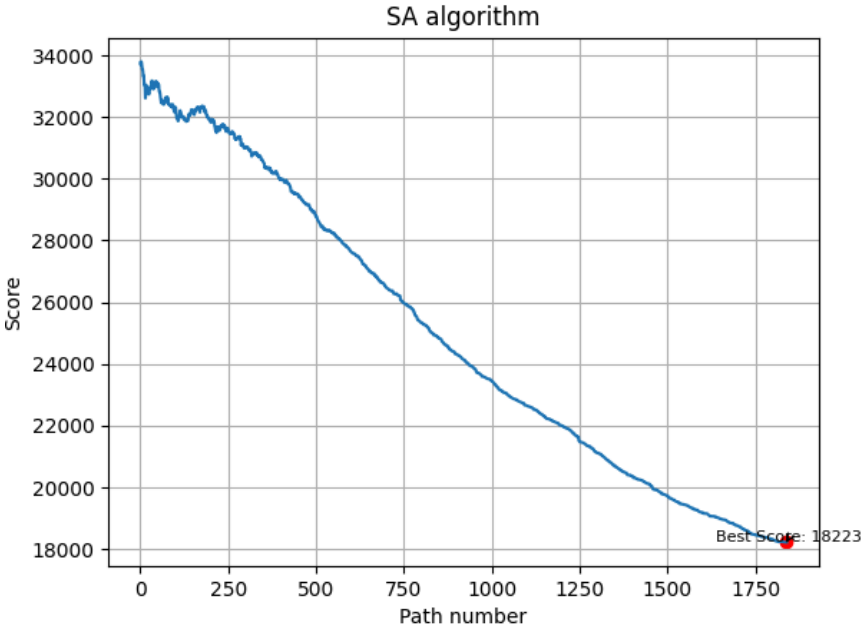
Zmiana parametru wpływa na faworyzowanie rozwiązań w których pracownik wykonuje mniej kroków. Na wykresie 6.1 wyraźne jest tąpnięcie dla wszystkich czterech symulacji przeprowadzonych dla parametru 1000. Oznacza to, że algorytm znalazł rozwiązanie w którym pracownik wykonuje mniej kroków więc liczba przerw zostaje zmniejszona. Na symulacji sporządzonej dla innych parametrów - 6.2 wyraźnie widać zmniejszenie ilości przerw.

Scenariusz 7

Próba znalezienia rozwiązania optymalnego czasowo.

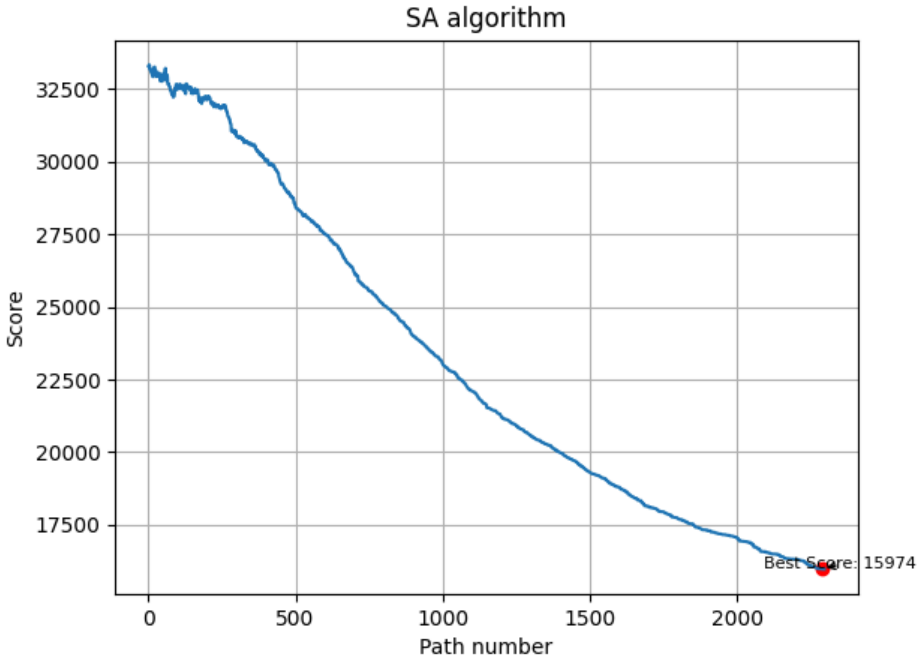
scenariusz 7	temperatura	min. temperatura	alpha	max. udźwig	max. liczba kroków	l. zmiany wektorów vertex	czas przerwy
Przykład 9	100	0.001	0.995	30	5000	2	100
Przykład 10	100	0.01	0.999	30	5000	2	1000

Elapsed Time: 441.72 seconds Minimalna długość trasy: 18223



wykres 7.1

Elapsed Time: 881.66seconds Minimalna długość trasy: 15974



wykres 7.2

Wnioski

Jak widać algorytm symulowanego wyżarzania jest bardzo pomocny dzięki mechanizmom pozwalającym na unikanie minimów lokalnych za pomocą ustawiania odpowiedniej temperatury. Może być on stosowany do rozwiązywania wielu problemów a jego efektywność będzie w głównej mierze zależeć od złożoności problemu oraz umiejętności dostosowania go do określonego zagadnienia. W naszym przypadku w testach na konkretnych scenariuszach mogliśmy zauważyć że bardzo wiele zależy od danych wejściowych oraz parametrów jakie wybieramy. W jednym scenariuszu wykresy przypominały standardowe przebiegi algorytmu z charakterystycznymi skokami na początku wynikającymi z wysokiej temperatury a przy innym wykres spadał schodkowo przez dominowanie niektórych parametrów. Podsumowując algorytm ten pozwolił rozwiązać opisany problem oraz dostarczyć wiele przydatnych informacji.