Asymmetric Travelling Salesman Problem - Projekt optymalizacyjny

Kacper Wnęk, Krzysztof Sawicki, Natalia Safiejko, Piotr Kosakowski, Wojciech Grabias

11 czerwca 2023

1 Streszczenie

Przedstawiony raport opisuje implementację i badania metody symulowanego wyżarzania w rozwiązaniu asymetrycznego problemu komiwojażera (ATSP). Metoda symulowanego wyżarzania jest metaheurystycznym algorytmem optymalizacji inspirowanym procesem chłodzenia stopionego metalu. Raport zawiera opis implementacji algorytmu oraz wyniki przeprowadzonych badań, których celem było znalezienie optymalnych parametrów dla różnych instancji problemu. Badania obejmowały zmianę funkcji prawdopodobieństwa, parametrów chłodzenia, początkowej temperatury i liczby iteracji. Dla problemów o różnych rozmiarach znaleziono najlepsze wyniki. Owe wyniki były zgodne dla różnych funkcji prawdopodobieństwa. Znaleziona została również optymalna liczba iteracji. Porównano algorytmy symulowanego wyżarzania (SA) i symulowanego wyżarzania z parallel tempering (TP_SA) na różnych problemach komiwojażera. TP_SA osiągał niewielką przewagę w niektórych przypadkach, ale dla większych problemów SA wypadł znacznie lepiej i charakteryzował się stabilnością wyników. Jednym z głównych problemów implementowanego algorytmu symulowanego wyżarzania było wpadanie w minima lokalne, co utrudniało znalezienie optymalnego rozwiązania.

Spis treści

2 Wstęp

W niniejszym raporcie przedstawiamy wyniki naszego projektu z przedmiotu "Warsztaty Badawcze". Celem projektu było napisanie algorytmu rozwiązującego asymetryczny problem komiwojażera (Asymmetric Traveling Salesman Problem; ATSP) jak najlepiej w rozsądnym czasie. Asymetryczny problem komiwojażera polega na znalezieniu minimalnego cyklu Hamiltona w pełnym skierowanym grafie ważonym. Rozwiązanie zagadnienia ATSP pozwala na optymalizacje tras w dystrybucji towarów, zbieraniu odpadów oraz w wielu innych gałęziach dotyczących transportu. Problem jest NP-zupełny, więc najprawdopodobniej nie istnieje algorytm znajdujący optymalne rozwiązanie w czasie wielomianowym[3]. Istnieje wiele algorytmów służących rozwiązaniu wyżej wspomnianego zagadnienia, jednak nasz zespół w tym celu zaimplementował metodę optymalizacji symulowanego wyżarzania (simulated annealing), opisanym w [1], oraz przeprowadził badania mające na celu znalezienie optymalnych parametrów dla różnych instancji problemu.

3 Opis implementowanej metody

Implementowana przez nas metoda to symulowane wyżarzanie (simulated annealing) - metaheurystyczny algorytm optymalizacji. Jest on inspirowany procesem chłodzenia stopionego metalu, gdzie kontrolowane schładzanie prowadzi do minimalizacji energii systemu. Algorytm symulowanego wyżarzania próbuje znaleźć optymalne rozwiązanie poprzez losowe eksplorowanie przestrzeni rozwiązań, a także akceptowanie gorszych rozwiązań w początkowej fazie działania, aby uniknąć utknięcia w lokalnym minimum. Implementacja inspirowana jest poruszonym w [4] ujęciu stosowania symulowanego wyżarzania w ogólnym problemie komiwojażera.

Algorytm polega na:

1. Wybraniu losowego rozwiązania początkowego.

```
S = random.sample(range(len(miasta)), len(miasta))
```

2. Wybraniu temperatury początkowej.

3. Powtarzaniu następującej sekwencji aż do spełnienia warunku zakończenia:

   - Wygenerowanie losowych sąsiednich rozwiązań.

```
S_p = S.copy()
a = random.randint(0, len(S_p) - 1)
b = random.randint(0, len(S_p) - 1)
S_p[a], S_p[b] = S_p[b], S_p[a]
```

   - Obliczenie różnicy kosztu (funkcji celu) między aktualnym rozwiązaniem a sąsiednim.

```
C1 = oblicz_koszt(S, odleglosc)
C2 = oblicz_koszt(S_p, odleglosc)
```

   - Jeśli różnica kosztu jest mniejsza niż zero, przyjęcie sąsiedniego rozwiązania jako aktualne. Jeśli różnica kosztu jest większa lub równa zero, obliczenie prawdopodobieństwa akceptacji na podstawie funkcji prawdopodobieństwa i temperatury.
     Porównanie wylosowanej wartości z prawdopodobieństwem akceptacji, jeśli jest mniejsza, przyjęcie sąsiedniego rozwiązania jako aktualne.

```
if C2 < C1:
    S = S_p
```

```
else:
    delta_e = abs(C1-C2)
    p = Prob_function(delta_e, T, prob_parameter)
    if random.random() < p:
        S = S_p
```

Gdzie Prob function to jedna z poniższych funkcji prawdopodobieństwa.

```
def metropolis_probability(delta_e, temperature, prob_parameter=None):
    return min(1, math.exp(-delta_e / temperature))

def boltzmann_probability(delta_e, temperature, prob_parameter=None):
    return math.exp(-delta_e / temperature)

def exponential_decrease_probability(delta_e, temperature, prob_parameter):
    return math.exp(-prob_parameter * delta_e / temperature)

def gaussian_probability(delta_e, temperature, prob_parameter):
    return math.exp(-(delta_e ** 2) / (2 * (prob_parameter ** 2) * temperature))

def power_law_probability(delta_e, temperature, prob_parameter):
    return 1 / ((delta_e + 1) ** prob_parameter * temperature)
```

Z czego rozkład gaussa posiadał parametr sigma wpływający na odchylenie standardowe. Rozkład power law parametr gamma, który wpływa na kształt rozkładu, decydując o tym, czy dominują większe czy mniejsze wartości. Z kolei exponential decrease posiadało parametr decay factors, który kontroluje tempo spadku w funkcji wykładniczej. Pozostałe rozkłady były pezparametrowe.

- Zmniejszenie temperatury według określonej funkcji zmiany temperatury T function (zmiana liniowa bądź wykładnica)

```
T = T_function(T_init, T, i, cool_parameter)
if T < break_point:
    break
```

4. Zwróceniu najlepszego znalezionego rozwiązania.

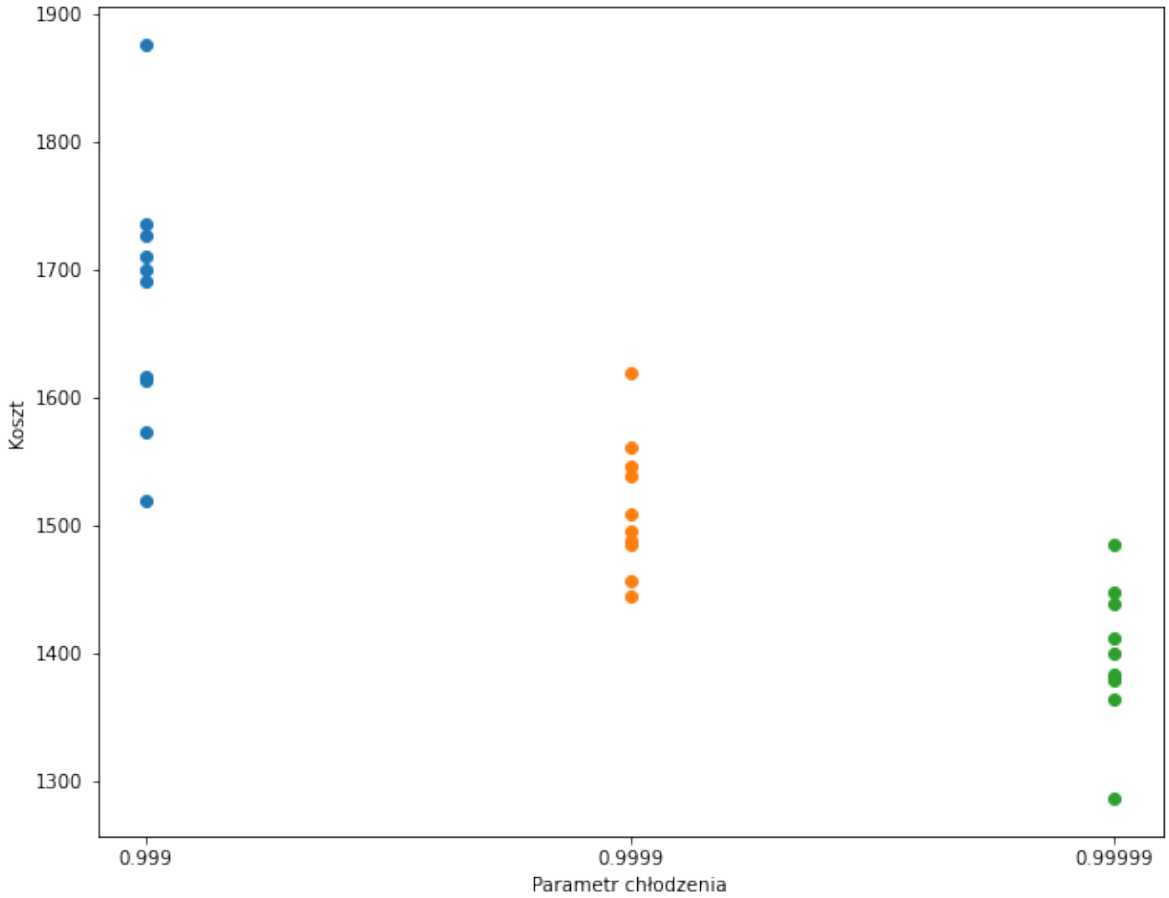
4 Główne wyniki

W celu znalezienia najlepszych parametrów przeprowadziliśmy testy na różnych problemach, które pomogły wybrać nam te najbardziej efektywne. Zmienialiśmy funkcję prawdopodobieństwa, jej parametry, parametr chłodzenia oraz początkową temperaturę.

Podczas zmiany jednego parametru reszta pozostawała taka sama. Wyjściowym zestawem był: temperatura początkowa 15000, parametr chłodzenia 0.99999 i funkcja prawdopodobieństwa metropolis, ponieważ w wyniku początkowych badań korzystając z tych parametrów były osiągane najlepsze wyniki. (dodatkowo funkcja metropolis jest tą najbardziej podstawową funkcją wykorzystywaną najczęściej w tym algorytmie).

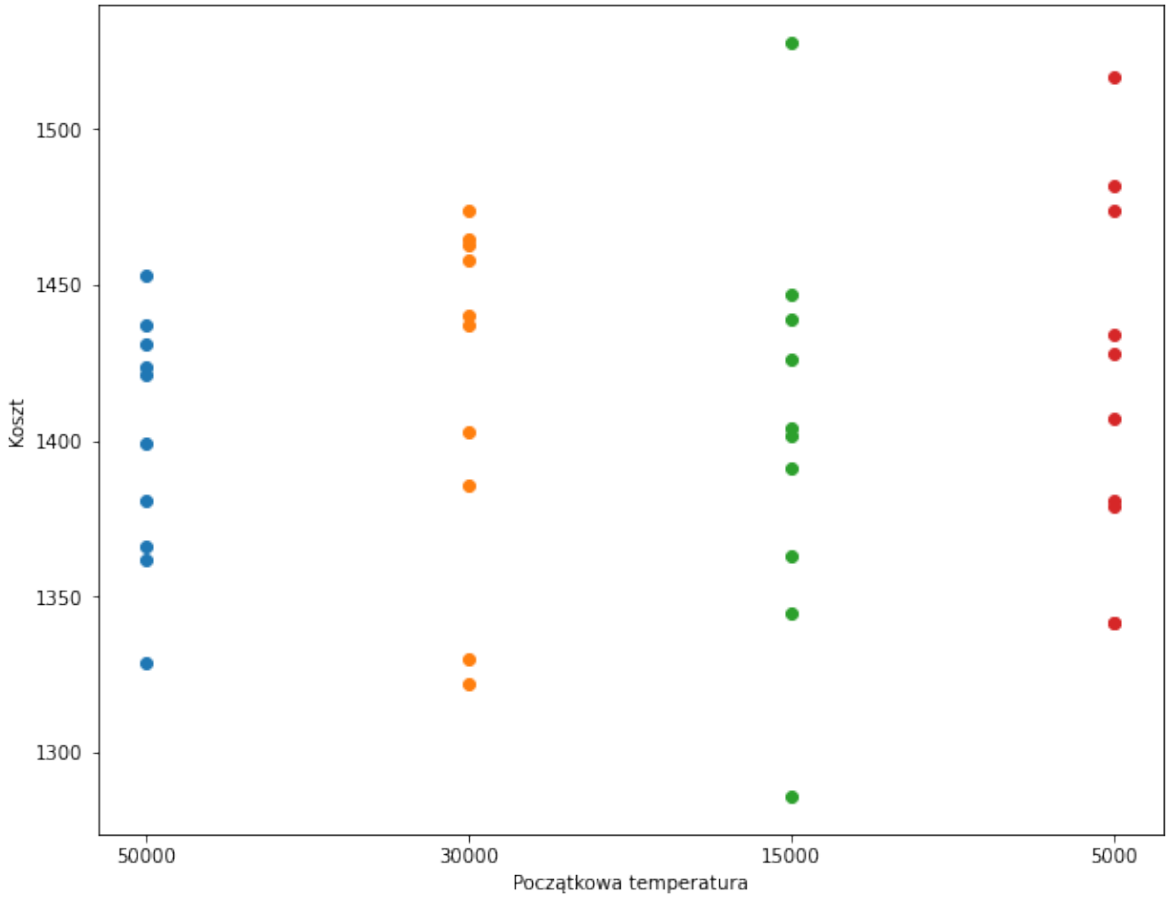
Wybraliśmy przykładowe problemy i wykresy przedstawiające wyniki naszych eksperymentów.

4.1 Problem ftv33



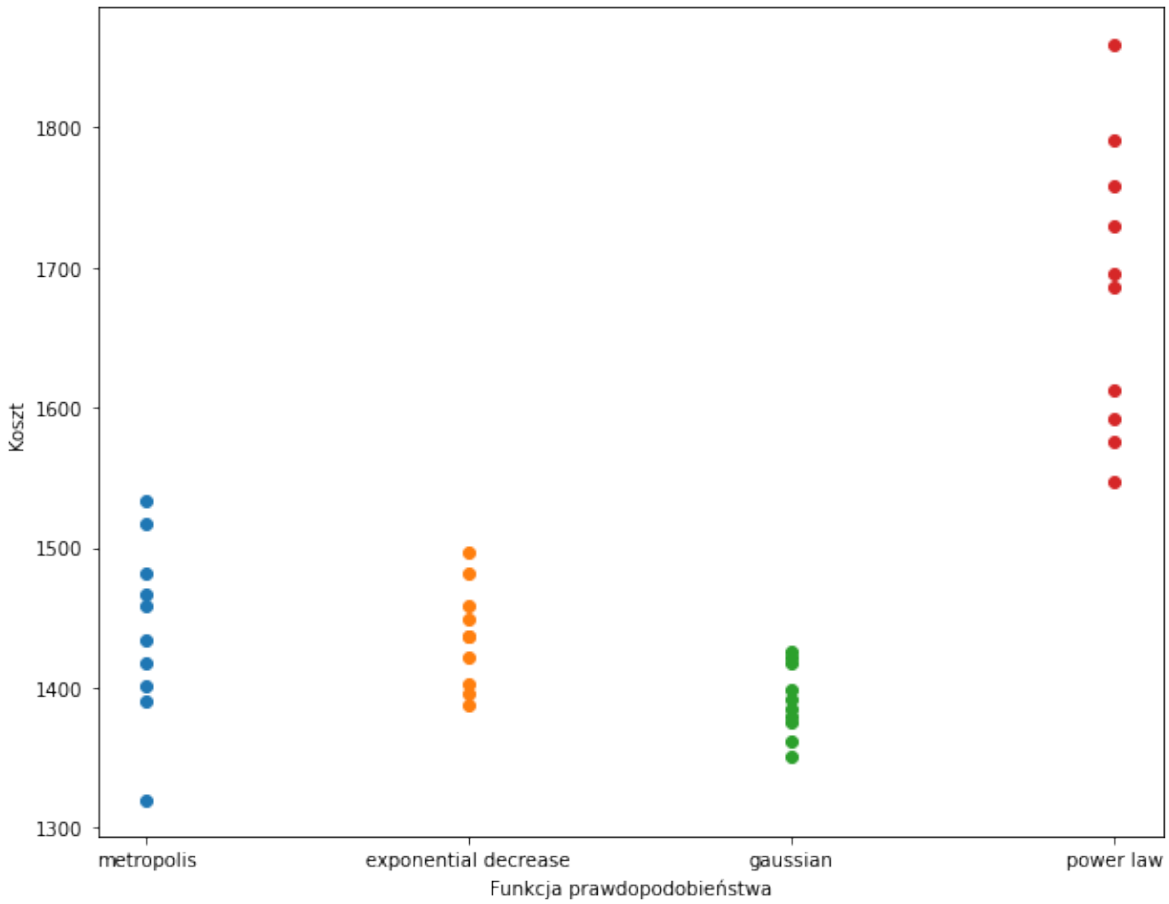
Obrazek 1: Zależność kosztu od parametru chłodzenia dla problemu ftv33, temp: 15000, f: metropolis

Na podstawie wykresu można wywnioskować, że im wolniej maleje temperatura, tym lepsze wyniki.



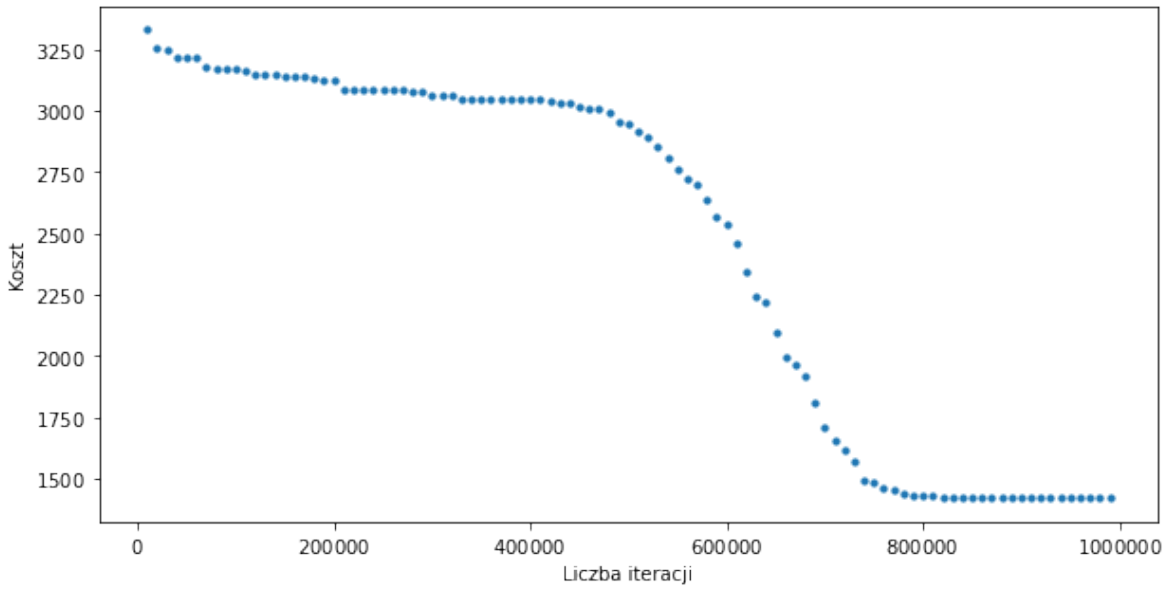
Obrazek 2: Zależność kosztu od początkowej temperatury dla problemu ftv33, parametr chłodzenia: 0.99999, f: metropolis

Nieoczywistym wnioskiem jest, że nie zawsze im wyższa temperatura początkowa, tym niższy koszt. W tym przypadku można zaobserwować brak wpływu temperatury na wynik.



Obrazek 3: Zależność kosztu od funkcji prawdopodobieństwa dla problemu ftv33, temp: 15000, parametr chłodzenia: 0.99999

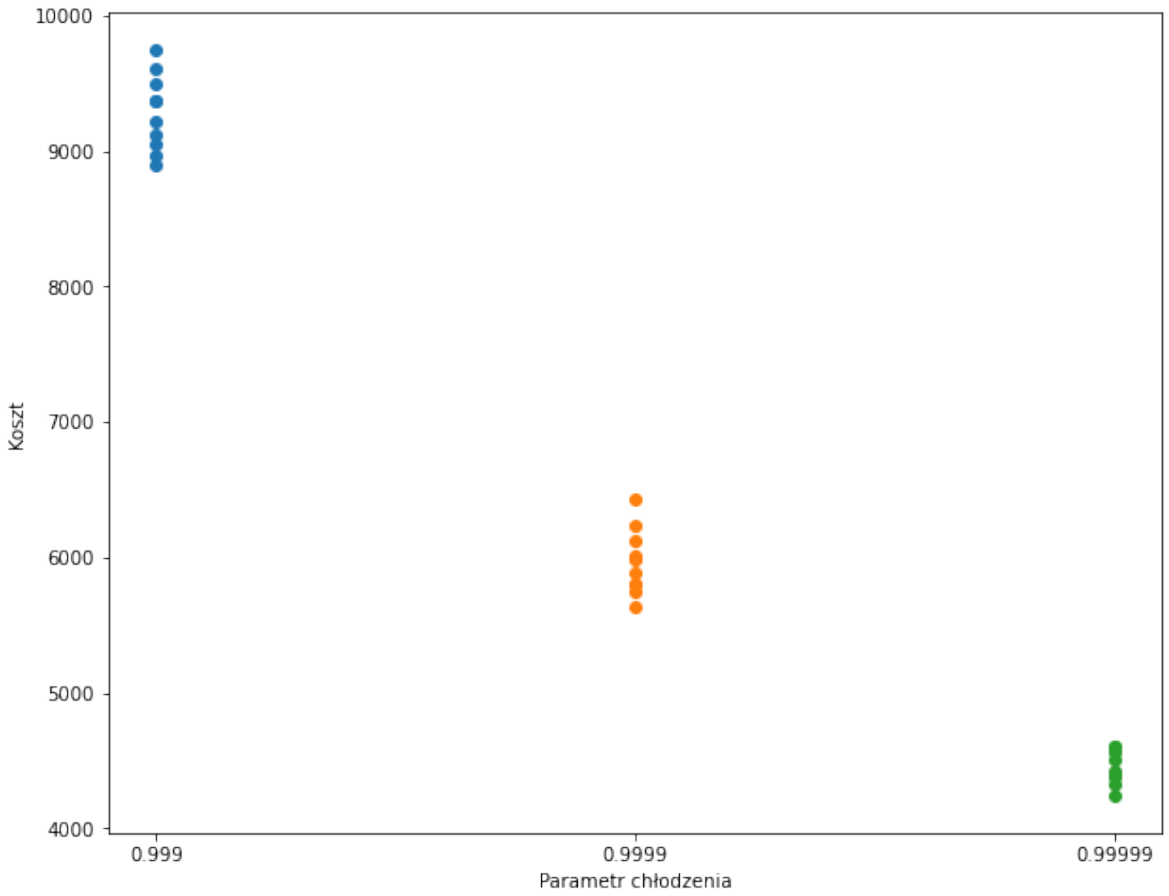
Najlepszy wynik został osiągnięty dla funkcji metropolis.



Obrazek 4: Zależność kosztu od liczby iteracji dla problemu ftv33, temp: 15000, parametr chłodzenia: 0.99999, f: metropolis

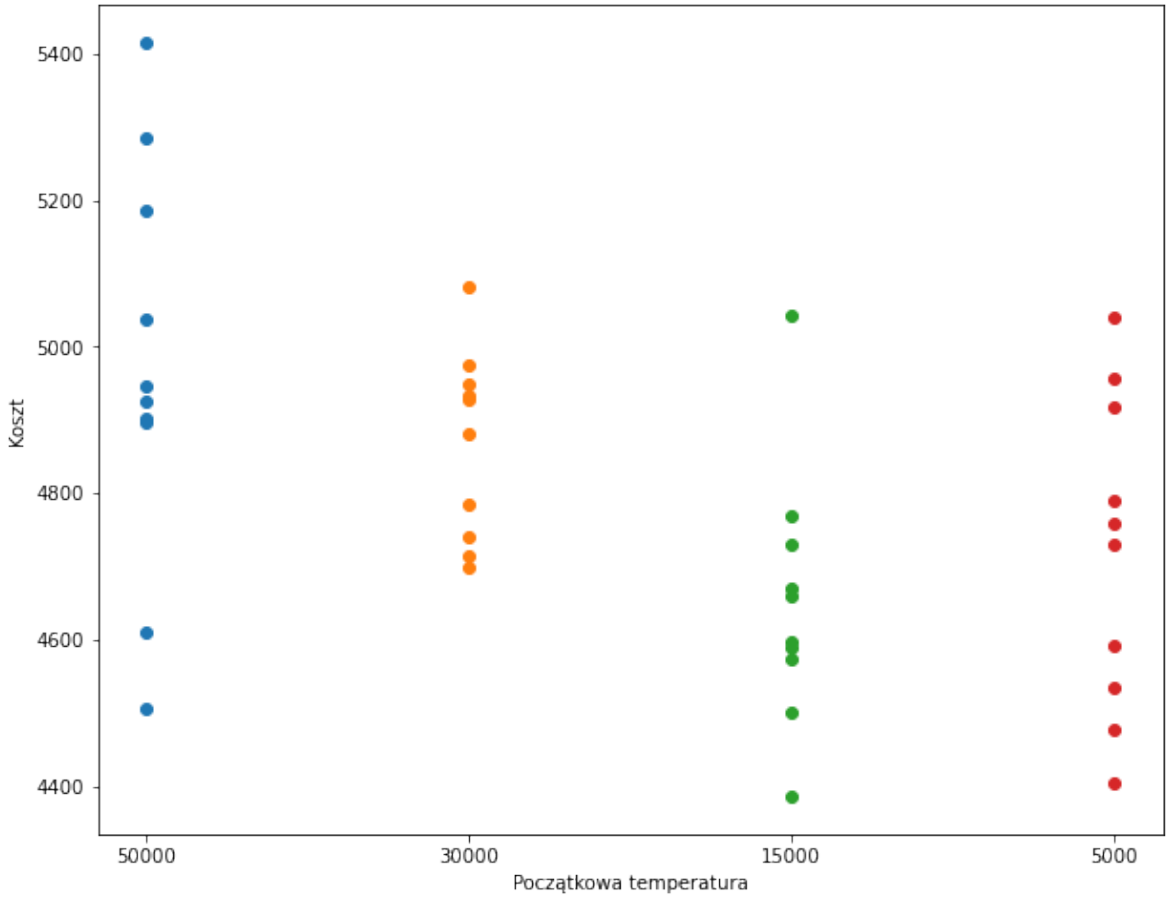
Na podstawie tych wykresów stwierdziliśmy, że najlepsze parametry dla problemu ftv33 to funkcja metropolis probability, z parametrem chłodzenia równym 0.99999.

4.2 Problem ftv170



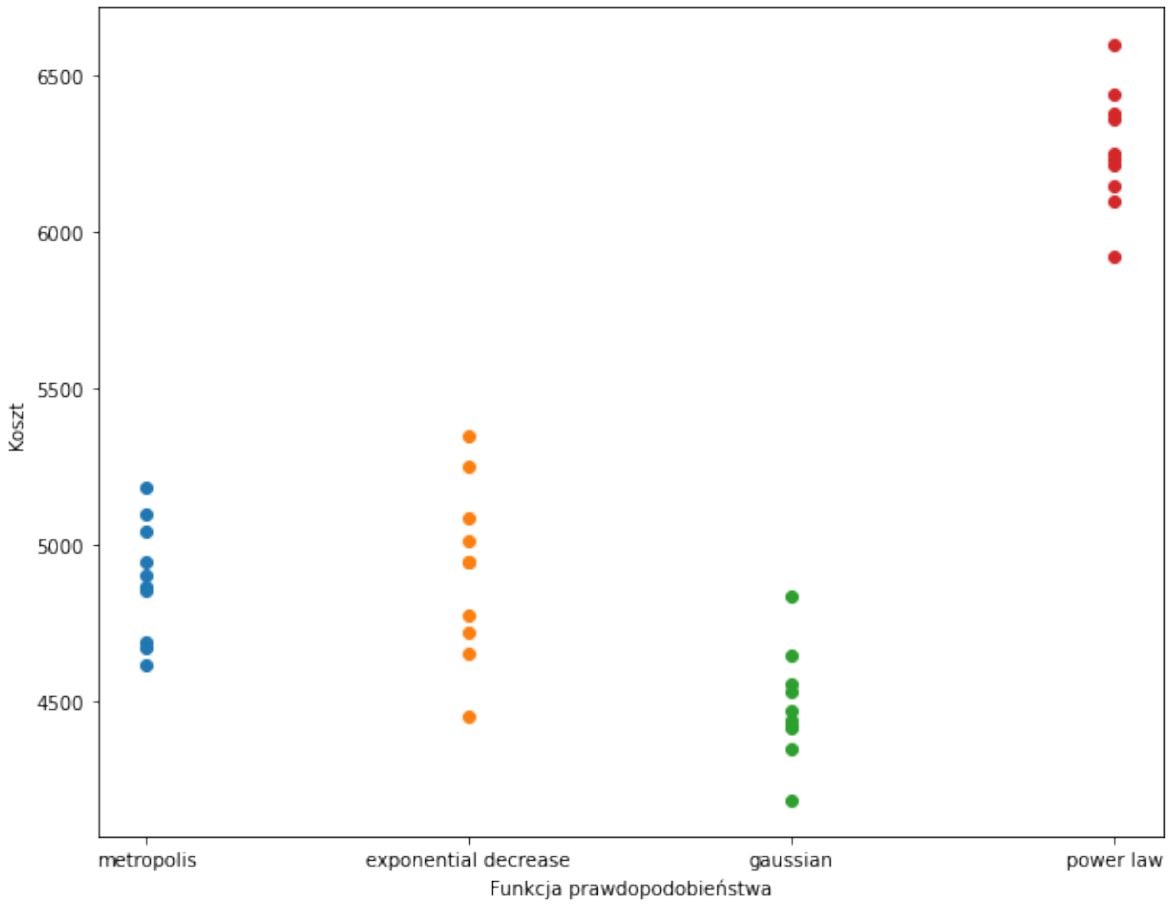
Obrazek 5: Zależność kosztu od parametru chłodzenia dla problemu ftv170, temp:15000, f: metropolis

Zdecydowanie najlepszym wyborem parametru chłodzenia w tym przypadku jest 0.99999.



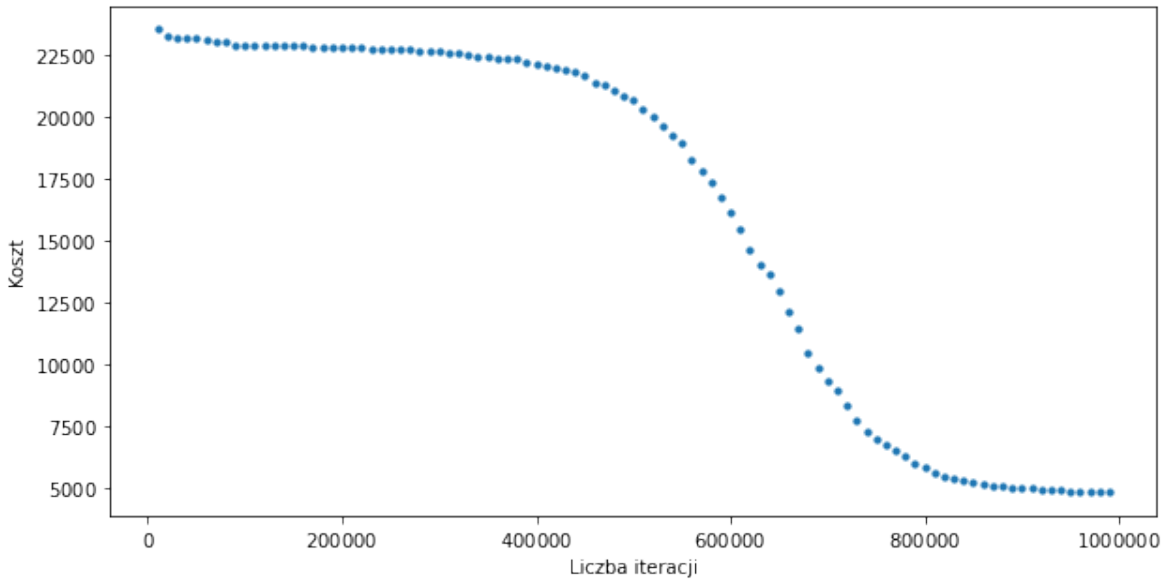
Obrazek 6: Zależność kosztu od początkowej temperatury dla problemu ftv170, parametr chłodzenia: 0.99999, f: metropolis

W przypadku temperatury ponownie lepsze okazały się te niższe- 15000 i 5000.



Obrazek 7: Zależność kosztu od funkcji prawdopodobieństwa dla problemu ftv170, temp: 15000, parametr chłodzenia: 0.99999

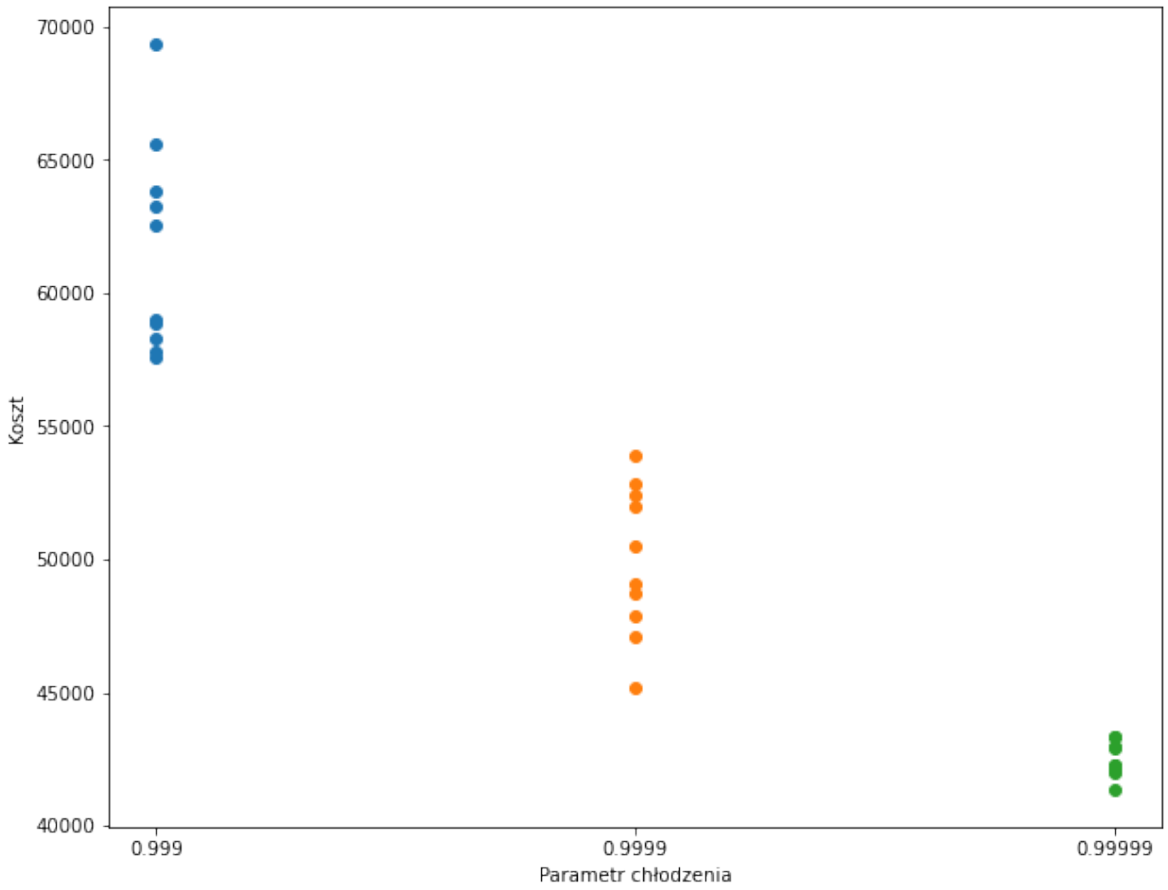
Najlepsze wyniki były osiągane korzystając z funkcji gaussian.



Obrazek 8: Zależność kosztu od liczby iteracji dla problemu ftv170, temp: 15000, parametr chłodzenia: 0.99999, f: metropolis

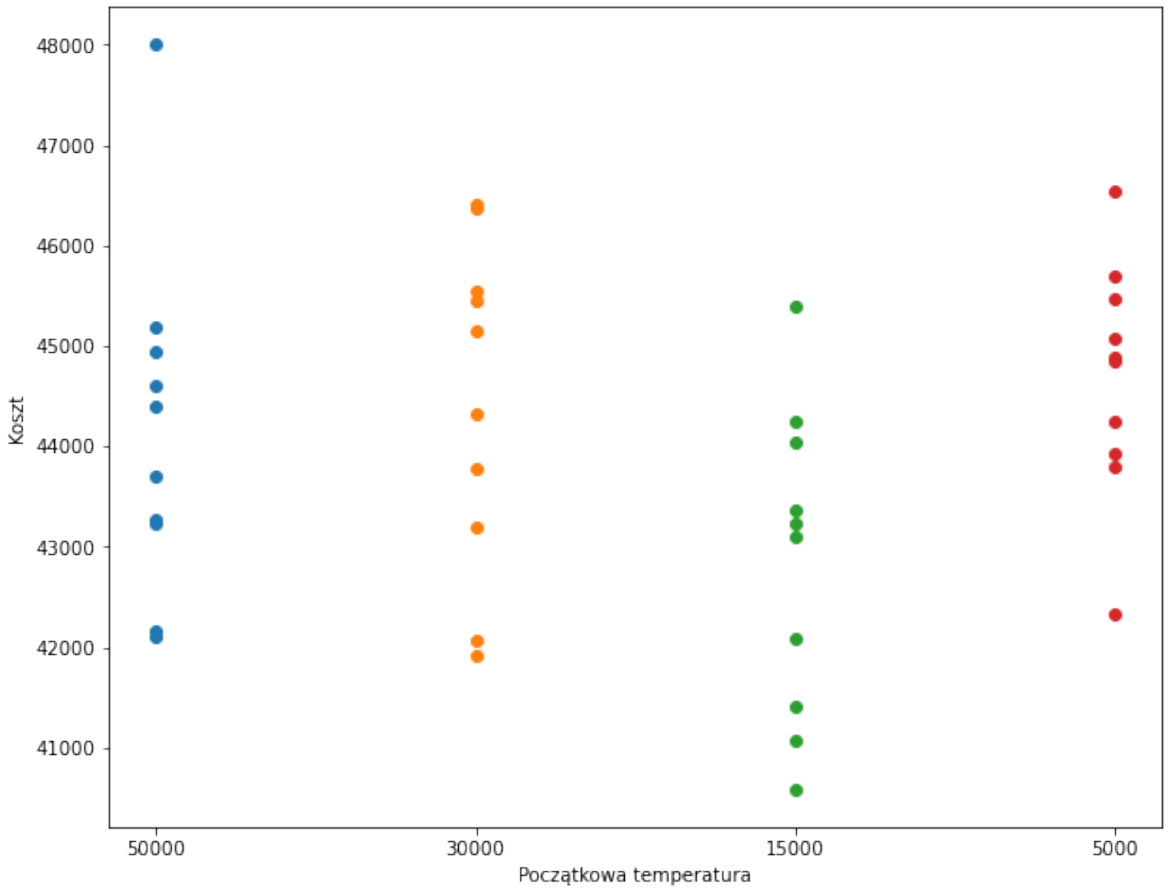
Najlepsze parametrami dla problemu ftv33 okazały się być:funkcja gaussian probability, z parametrem chłodzenia równym 0.99999 i początkową temperaturą równą 15000.

4.3 Problem kro124p



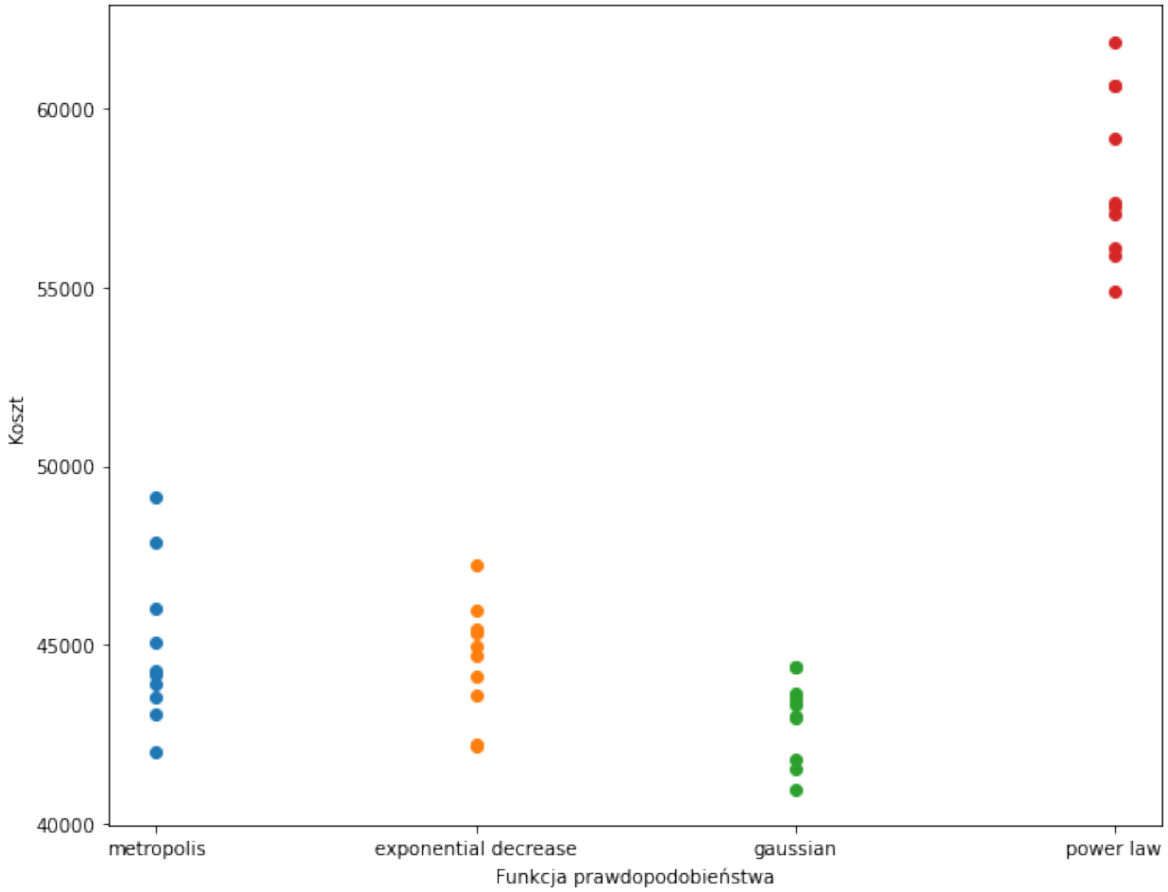
Obrazek 9: Zależność kosztu od parametru chłodzenia dla problemu kro124p, temp: 15000, f: metropolis

Ponownie najlepszy parametr chłodzenia to 0.99999.



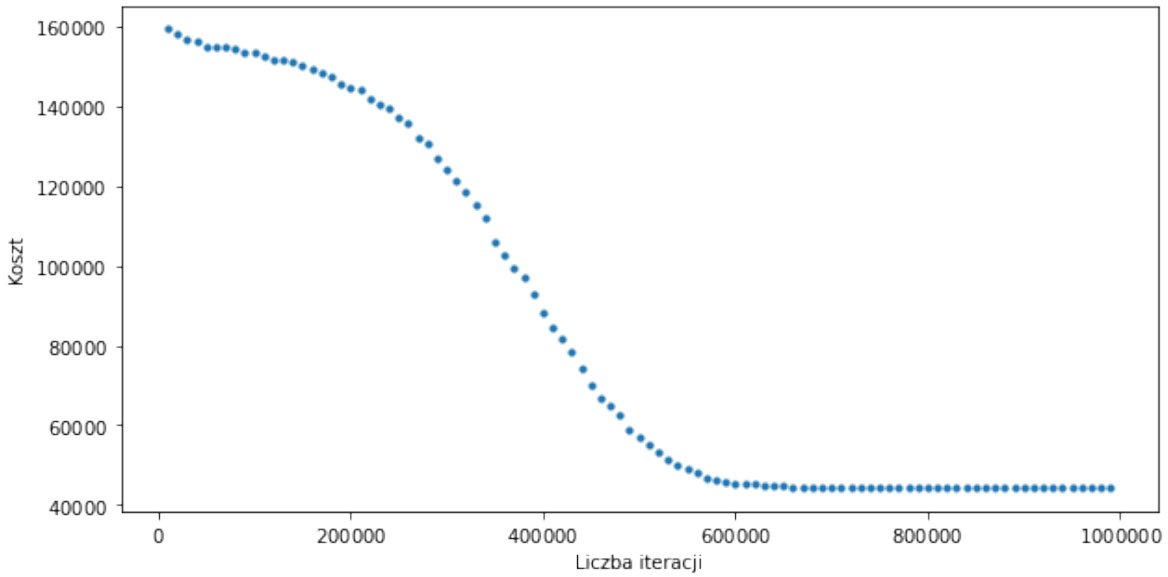
Obrazek 10: Zależność kosztu od początkowej temperatury dla problemu kro124p, parametr chłodzenia: 0.99999, f: metropolis

Także temperatura 15000 kolejny raz okazała się być najbardziej optymalna.



Obrazek 11: Zależność kosztu od funkcji prawdopodobieństwa dla problemu kro124p, , temp:15000, parametr chłodzenia:0.99999

Korzystanie z funkcji gaussian w przypadku tego problemu daje najlepsze wyniki.



Obrazek 12: Zależność kosztu od liczby iteracji dla problemu kro124p, temp: 15000, parametr chłodzenia: 0.99999, f: metropolis

W tym przypadku najlepsze parametry to funkcja gaussian probability, z parametrem chłodzenia równym 0.99999 i początkową temperaturą równą 15000.

4.4 Najlepsze zestawy parametrów

Korzystaliśmy z funkcji, która zwracała najlepsze parametry biorąc pod uwagę końcowy koszt oraz czas. Opis jej działania: W obrębie jednego problemu funkcja skaluje min-max kosztu funkcji na [0,1] dla danych parametrów. Obliczana jest unormowana odległość od najlepszego rozwiązania. Szukamy takich parametrów, dla których średnia odległość po wszystkich rozważanych problemach będzie najmniejsza. Zwracane jest 5 najlepszych zestawów przedstawionych poniżej:
Rankingi zestawów parametrów dla poszczególnych problemów:

	funkcja prawdopodobieństwa	parametr funkcji	parametr chłodzenia	temperatura początkowa
1.	gaussian	3.0	0.999999	15000
2.	exponential decrease	3.0	0.999999	15000
3.	gaussian	2.0	0.999999	15000
4.	exponential decrease	0.3	0.999999	15000
5.	boltzman	-	0.999999	15000

Obrazek 13: Ranking parametrów ft70

Funkcja boltzman jest bezparametrowa, dlatego nie ma podanego parametru.

	funkcja prawdopodobieństwa	parametr funkcji	parametr chłodzenia	temperatura początkowa
1.	gaussian	3.0	0.99999	15000
2.	exponential decrease	3.0	0.99999	15000
3.	gaussian	2.5	0.99999	15000
4.	gaussian	1.5	0.99999	15000
5.	exponential decrease	0.1	0.99999	15000

Obrazek 14: Ranking parametrów ftv53

	funkcja prawdopodobieństwa	parametr funkcji	parametr chłodzenia	temperatura początkowa
1.	gaussian	0.4	0.999999	15000
2.	gaussian	0.1	0.999999	15000
3.	gaussian	1.0	0.999999	15000
4.	gaussian	3.0	0.999999	15000
5.	gaussian	2.0	0.999999	15000

Obrazek 15: Ranking parametrów kro124p

	funkcja prawdopodobieństwa	parametr funkcji	parametr chłodzenia	temperatura początkowa
1.	gaussian	0.4	0.999999	15000
2.	gaussian	2.0	0.999999	15000
3.	gaussian	0.6	0.999999	15000
4.	gaussian	0.1	0.999999	15000
5.	exponential decrease	1.5	0.999999	15000

Obrazek 16: Ranking parametrów rbg323

Ostatecznie stwierdziliśmy, że dwa najlepsze zestawy to funkcja gaussian z parametrem 0.4 i gaussian z parametrem 3.0. Wyniki obu zestawów były podobne, dlatego zdecydowaliśmy się na wybór jednego, którego dalej testowaliśmy- funkcja gaussian z parametrem 0.4.

4.5 Wyniki wyścigu

Otrzymane wnioski wykorzystaliśmy podczas wyścigów algorytmów.

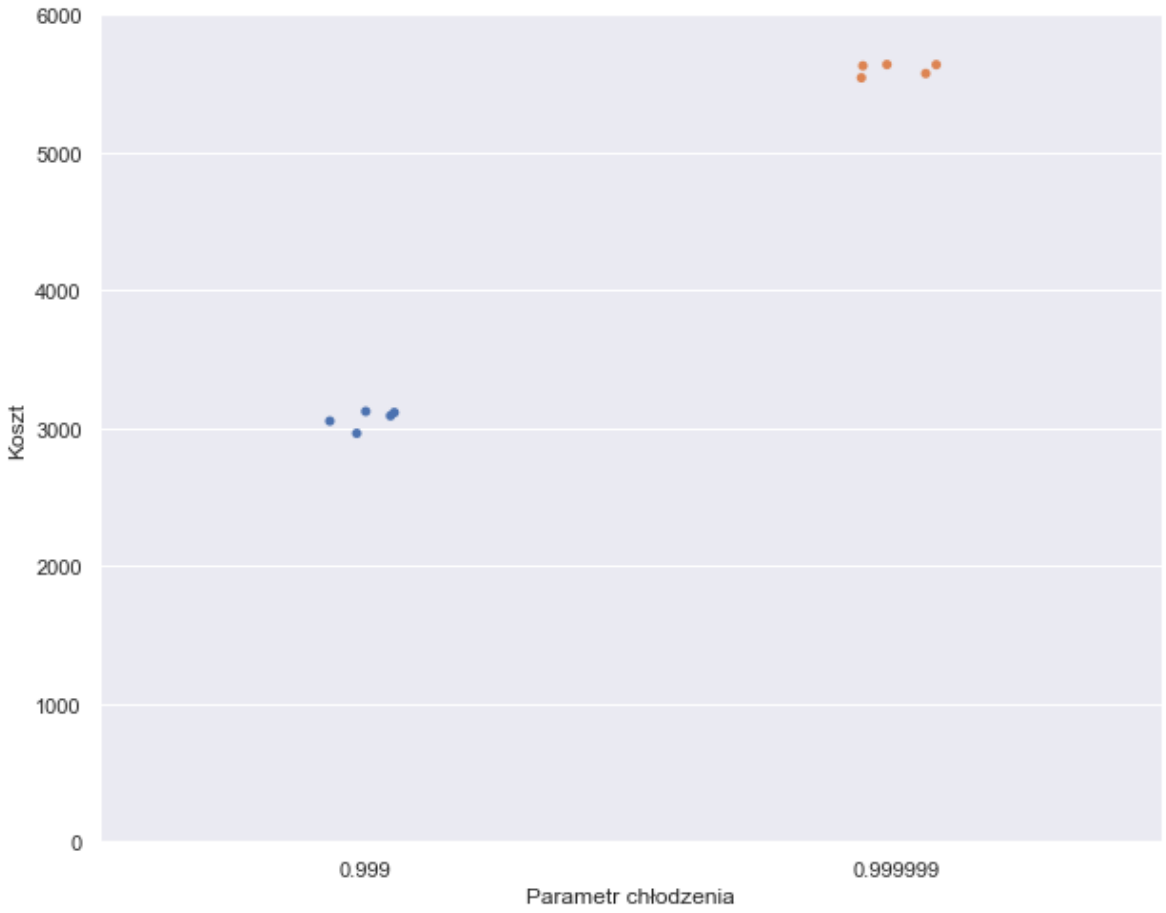
- problem ftv38
  koszt: 1652
  ścieżka: [3, 38, 2, 4, 35, 33, 30, 26, 23, 24, 36, 21, 22, 25, 31, 32, 34, 29, 28, 27, 20, 13, 12, 11, 15, 6, 7, 5, 8, 37, 9, 10, 16, 14, 17, 18, 19, 0, 1]

- problem ft70
  koszt: 40286
  ścieżka: [29, 23, 56, 54, 52, 51, 50, 17, 16, 15, 6, 38, 35, 42, 41, 40, 36, 39, 37, 69, 65, 68, 67, 63, 66, 64, 21, 27, 24, 28, 30, 34, 32, 12, 14, 10, 9, 11, 7, 13, 8, 20, 18, 19, 46, 45, 49, 47, 55, 53, 2, 4, 22, 25, 3, 5, 1, 0, 26, 33, 31, 62, 61, 60, 59, 58, 57, 44, 48, 43]

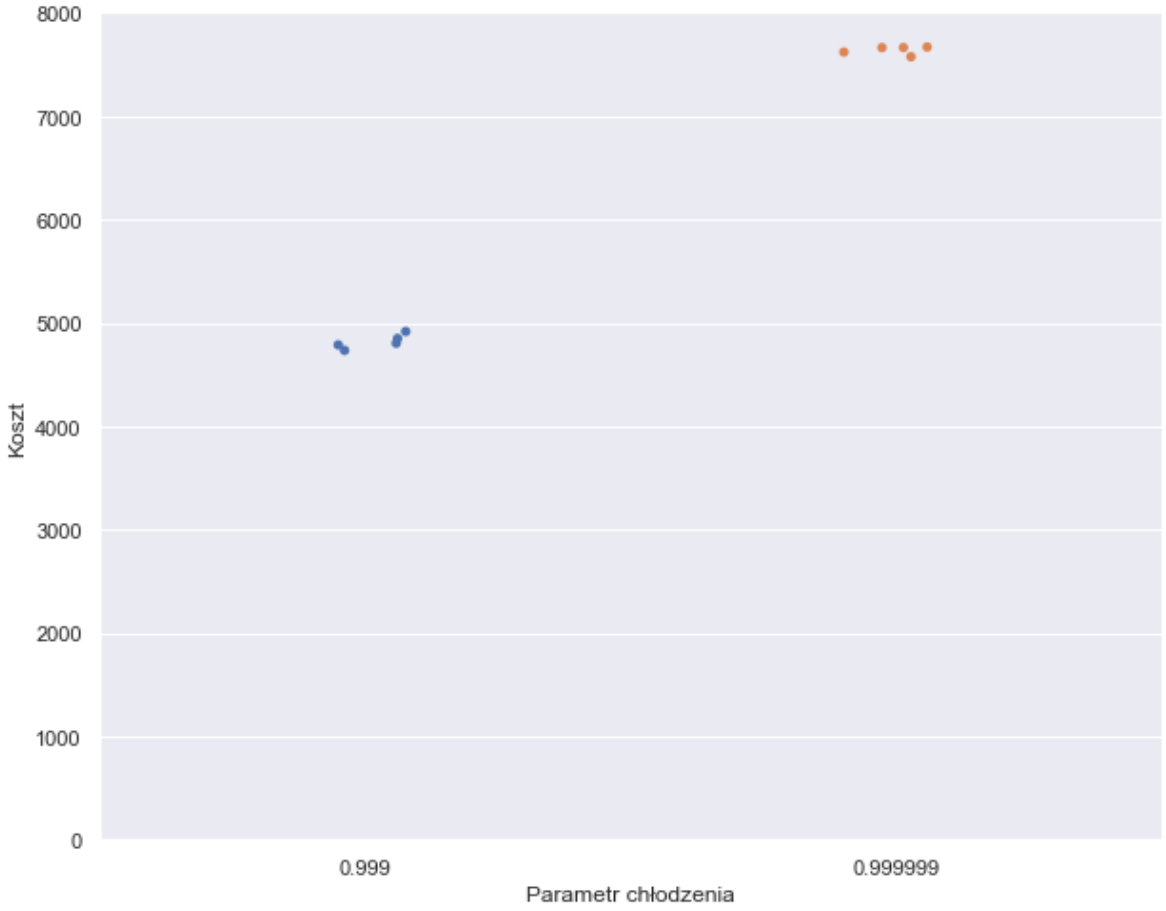
- problem ftv38
  koszt: 7019
  ścieżka: [131, 130, 66, 269, 147, 366, 306, 242, 262, 224, 77, 88, 334, 122, 272, 344, 193, 348, 212, 367, 328, 358, 58, 97, 208, 266, 163, 183, 113, 368, 332, 53, 341, 149, 85, 91, 71, 346, 137, 21, 168, 32, 124, 55, 63, 398, 110, 285, 196, 292, 288, 268, 199, 261, 0, 233, 222, 86, 310, 382, 302, 127, 148, 372, 307, 390, 161, 24, 335, 352, 246, 11, 396, 65, 343, 74, 384, 392, 376, 289, 40, 134, 205, 362, 339, 29, 151, 329, 60, 14, 379, 119, 181, 207, 185, 120, 68, 78, 259, 225, 312, 175, 106, 95, 2, 273, 172, 173, 235, 117, 381, 43, 234, 64, 182, 126, 287, 18, 248, 349, 359, 264, 115, 178, 31, 282, 342, 209, 146, 156, 143, 94, 249, 23, 394, 371, 364, 41, 313, 36, 93, 247, 291, 378, 320, 16, 338, 81, 57, 123, 166, 9, 164, 1, 283, 258, 238, 221, 318, 304, 388, 290, 128, 211, 165, 135, 322, 192, 237, 319, 391, 252, 360, 125, 70, 333, 385, 69, 49, 239, 72, 99, 229, 136, 256, 355, 103, 80, 167, 356, 274, 253, 317, 204, 47, 96, 220, 201, 142, 255, 155, 383, 297, 174, 38, 397, 203, 218, 6, 354, 59, 210, 375, 13, 98, 401, 278, 325, 299, 215, 267, 92, 144, 159, 244, 198, 160, 111, 271, 154, 350, 284, 377, 50, 270, 12, 214, 217, 20, 337, 386, 157, 108, 109, 33, 105, 275, 380, 277, 361, 82, 369, 186, 389, 4, 25, 374, 56, 46, 351, 293, 227, 150, 303, 301, 340, 8, 61, 311, 232, 114, 133, 353, 177, 73, 300, 162, 190, 243, 400, 326, 226, 314, 279, 327, 286, 195, 363, 52, 223, 263, 79, 280, 104, 34, 197, 112, 202, 373, 170, 260, 228, 51, 324, 45, 365, 236, 121, 44, 189, 37, 141, 116, 179, 138, 22, 308, 102, 67, 305, 316, 39, 230, 27, 17, 100, 153, 26, 7, 35, 330, 158, 323, 296, 118, 101, 219, 281, 321, 188, 240, 140, 76, 42, 15, 345, 231, 357, 184, 295, 241, 145, 48, 90, 5, 331, 75, 107, 28, 54, 265, 402, 10, 276, 132, 294, 171, 191, 187, 84, 370, 395, 216, 89, 336, 87, 315, 3, 19, 251, 206, 399, 194, 129, 347, 309, 213, 250, 298, 30, 200, 393, 169, 180, 387, 62, 254, 83, 139, 245, 176, 257, 152]

4.6 Problemy związane z ograniczeniem czasowym

Podczas badań zauważyliśmy, że (szczególnie przy większych problemach) przy większych parametrach chłodzenia nasz algorytm nie zdąży skończyć obliczeń i zwraca wynik, który po 5 min został uznany za najlepszy. Często jest to niezadowalający rezultat, który różni się od oczekiwanego kosztu o kilka tysięcy. Dobrze to widać na poniższych wykresach:



Obrazek 17: Parametry i wyniki dla problemu rbg323



Obrazek 18: Parametry i wyniki dla problemu rbg443

Korzystając z parametru 0.999 algorytm został przerwany poprzez osiągnięcie minimalnej temperatury. Natomiast w przypadku parametru 0.999999 w momencie przerwania algorytmu w obu przypadkach temperatura wynosiła około 7000. Nie zdążyła spaść, dlatego wyniki nie są zadowalające. Wzięliśmy to pod uwagę i wyciągnęliśmy wnioski.

5 Podsumowanie i wnioski

Najczęściej powtarzającym się wnioskiem w naszych testach było to, że najlepsze wyniki są osiągane dla temperatury 15000 i parametrem chłodzenia 0.999999- czyli im wolniej spada temperatura tym lepiej. Trzeba było jednak dostosowywać to do rozmiaru problemu. Nie mogliśmy wybierać zbyt dużego parametru chłodzenia, ponieważ algorytm nie skończyłby obliczeń w przeciągu 5 minut. Najbardziej optymalne wyniki otrzymywaliśmy korzystając z funkcji Gaussian Probability z parametrem 0.4.

W niektórych przypadkach- na przykład przy problemie br17 wybór parametrów nie był aż tak istotny- najbardziej optymalny wynik był osiągany bez względu na funkcję czy szybkość chłodzenia. Jedyna różnica to czas obliczeń- również niewielka.

Algorytm zazwyczaj w krótkim czasie znajdował bardzo dobre rozwiązania- o koszcie zbliżonym do tego najbardziej optymalnego. W przeciągu 5 minut otrzymywaliśmy wynik różniący się od najlepszego o maksymalnie 10%

Implementowany algorytm działa w taki sposób, że kandydat na minimum rzadko kiedy jest zmieniany na inny (widoczna pozioma linia na wykresie 12). Dlatego też czasami korzystniej byłoby w ciągu 5 minut przeliczyć kilka razy dany zestaw przy szybszym chłodzeniu niż czekać, aż algorytm zwróci jedno rozwiązanie. Na początku należało ustawić liczbę iteracji, po której algorytm ma zakończyć pracę. Naszą domyślną wartością było 1 000 000. Zaobserwowaliśmy, że zazwyczaj najlepszy wynik jest osiągany po około 800 000 iteracjach, więc jeśli zależałoby nam na jak najszybszym znalezieniu

rozwiązania, moglibyśmy ograniczyć ich liczbę (algorytm wcześniej zakończyłby pracę).

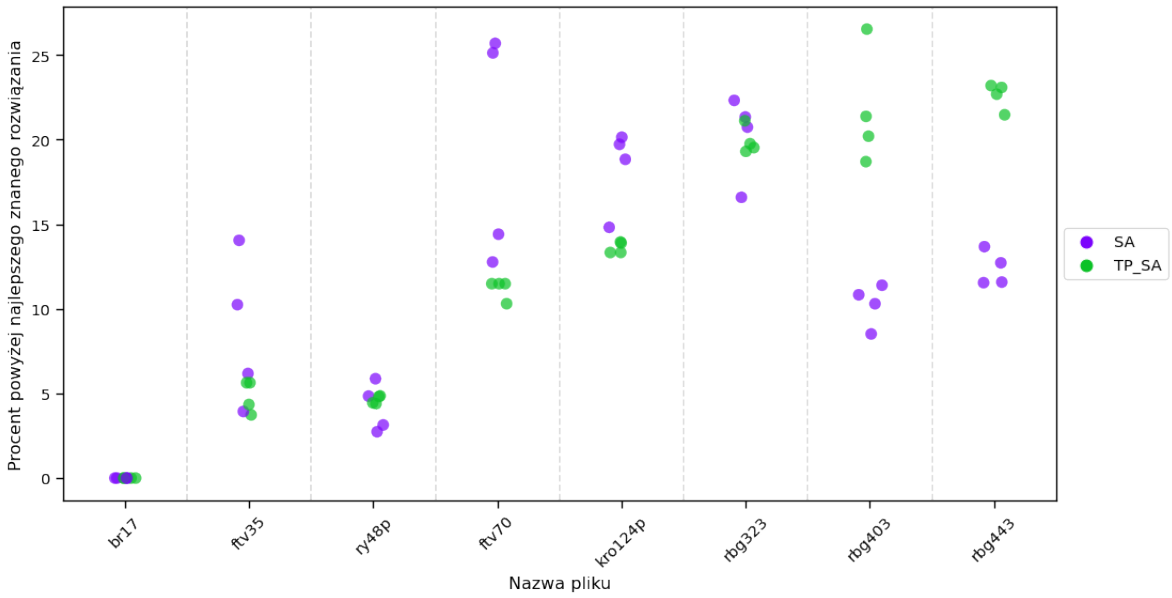
Wybierając losowo parametry nie osiągaliśmy tak dobrych wyników. Bardzo pomocna okazała się funkcja, która wskazała nam te najbardziej optymalne wielkości. Mimo, że nie dla wszystkich problemów otrzymaliśmy te same rezultaty, udało nam się wybrać dwa najbardziej uniwersalne.

Działanie algorytmu silnie zależy od początkowego wyboru ścieżki. Wybory są dokonywane losowo- między innymi to, czy zaakceptowany zostanie gorszy wynik czy też nie. Dlatego też można zauważyć duże różnice pomiędzy rozwiązaniami.

W związku z tym każdy zestaw parametrów testowaliśmy wiele razy, aby uniknąć wyciągania wniosków z losowych wyników (co można zauważyć na wykresach w Głównych wynikach 4).

6 Porównanie z innym zespołem

Zaimplementowany algorytm symulowanego wyżarzania porównany zostanie do algorytmu łączącego symulowane wyżarzanie z parallel tempering, który działał na zasadzie wielowątkowego wywoływania parallel tempering, używającego Metropolis Transition, następnie Replica Transition, a następnie wywoływane było symulowane wyżarzanie. Cały proces powtarzał się, w zależności od tego, czy liczba miast była większa niż 300, odpowiednio dwa lub sześć razy. Działanie algorytmów porównywane było na podstawie, ze względu na ograniczone zasoby czasowe, czterech wywołań każdego z nich na ośmiu różnych problemach. Dokładne nazwy plików opisujących dany problem znajdują się na wykresie poniżej. Sprawdzane jest jak w danym wywołaniu algorytm blisko był od najlepszego znanego rozwiązania. TP_SA oznacza symulowane wyżarzanie z parallel tempering, SA symulowane wyżarzanie. Problemy zostały wybrane losowo o różnych rozmiarach.



Obrazek 19: Procent ponad najlepsze znane rozwiązanie dla każdego problemu i algorytmu

W przypadku 17 miast obydwa algorytmy poradziły sobie równie dobrze, osiągając najlepsze znane rozwiązanie za każdym razem. W przypadku 35, 70 i 124 miast algorytm TP_SA charakteryzował się niewiele lepszymi wynikami. W przypadku dużej ilości miast, tutaj 403 i 443, algorytm SA konsekwentnie wypadał lepiej, nie przekraczając wyniku piętnastu procent ponad najlepsze znane rozwiązanie. Ponadto, każdy wynik dla SA był zbliżony, co świadczy również o dobrej stabilności przy takich problemach. W pozostałych przypadkach algorytmy osiągają zbliżone wyniki. Zaletą TP_SA jest stabilność wyników na niemalże każdym problemie.
Warto jednak pamiętać, iż TP_SA wymaga większych zasobów obliczeniowych, ze względu na wykorzystywanie wielowątkowości.

7 Dalsze możliwości rozwoju

Dalsze prace rozwojowe rozpoczną się od uodpornienia algorytmu na wpadanie w minima lokalne. Ponadto aktualnie algorytm zaprojektowany jest pod zwracanie wyniku w nie dłużej niż pięć minut. Przyszła wersja mogła by działać przez dowolny, zadany przez użytkownika, czas i wykorzystywać go w jak najlepszy sposób. Poniżej przedstawione zostało kilka rozwiązań, które mogą pomóc osiągnąć opisane cele.

Najprostszą modyfikacją może być zmienienie warunku stopu. Mianowicie dopóki czas nie przekroczy żądanej wartości w przypadku osiągnięcia granicznej temperatury program zapamiętuje najlepsze rozwiązanie i wykonuje wyżarzanie na nowo, możliwie z mniejszą temperaturą początkową, lub szybszym chłodzeniem.

Zamiast wykonywać wyżarzanie kilka razy z rzędu można by wykonywać je, używając wielowątkowości lub wieloprocesowości, kilka razy jednocześnie. To rozwiązanie jednak jest bardziej wymagające sprzętowo, więc nie sprawdzi się najlepiej na słabszych komputerach. Oczywiście, takie równoległe wywołania również można wykonać kilkukrotnie, gdy czas na to pozwala.

Ostatnim pomysłem jest dynamiczny dobór parametrów, w zależności od aktualnej temperatury lub pozostałego czasu do końca, jak funkcja chłodzenia, jej parametry, czy funkcja prawdopodobieństwa. W przypadku kolejnego wywołania wyżarzania także temperatury początkowej. Na przykład, w późnych fazach działania, gdy potencjalnie algorytm znajduje się blisko minimum, można obniżyć tempo chłodzenia, w celu dokładniejszej eksploracji. Jeśli algorytm wywoływany po raz kolejny, może startować z niższą temperaturą i szybszym chłodzeniem. Ponadto, aktualna wersja w każdej iteracji zamienia dwa losowo wybrane miasta miejscami szukając lepszego rozwiązania. Tutaj również początkowo, na przykład, mogła by być zamieniana większa liczba miast dla szybszej eksploracji, zaś później zmiany były by mniejsze, aby szukać w obszarze podejrzanym o bycie okolicą minimum. Możliwe było by również użycie Sequentially Modified Cost Function, poruszone w [2].

Powyższe propozycje mogły by również być łączone, jednakże niektóre z nich mogą mieć istotny wpływ na szybkość obliczeń lub dokładność w szukaniu rozwiązań, tak więc wprowadzenie zmian musi być poprzedzone testami, a także dobierane pod zasoby czasowe i sprzętowe potencjalnych użytkowników.

8 Bibliografia

Literatura

[1] Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statistical science*, 8(1):10–15, 1993.

[2] Jiayin Chen and Hendra I Nurdin. Generalized simulated annealing with sequentially modified cost function for combinatorial optimization problems. In *2019 Australian & New Zealand Control Conference (ANZCC)*, pages 48–53. IEEE, 2019.

[3] Wojciech Śmigielski. Heurystyki i algorytmy aproksymacyjne w asymetrycznym problemie komiwojażera. 2019.

[4] Shi-hua Zhan, Juan Lin, Ze-jun Zhang, and Yi-wen Zhong. List-based simulated annealing algorithm for traveling salesman problem. *Computational intelligence and neuroscience*, 2016, 2016.

[5] Github: https://github.com/ssafiejko/Travelling-salesman-problem