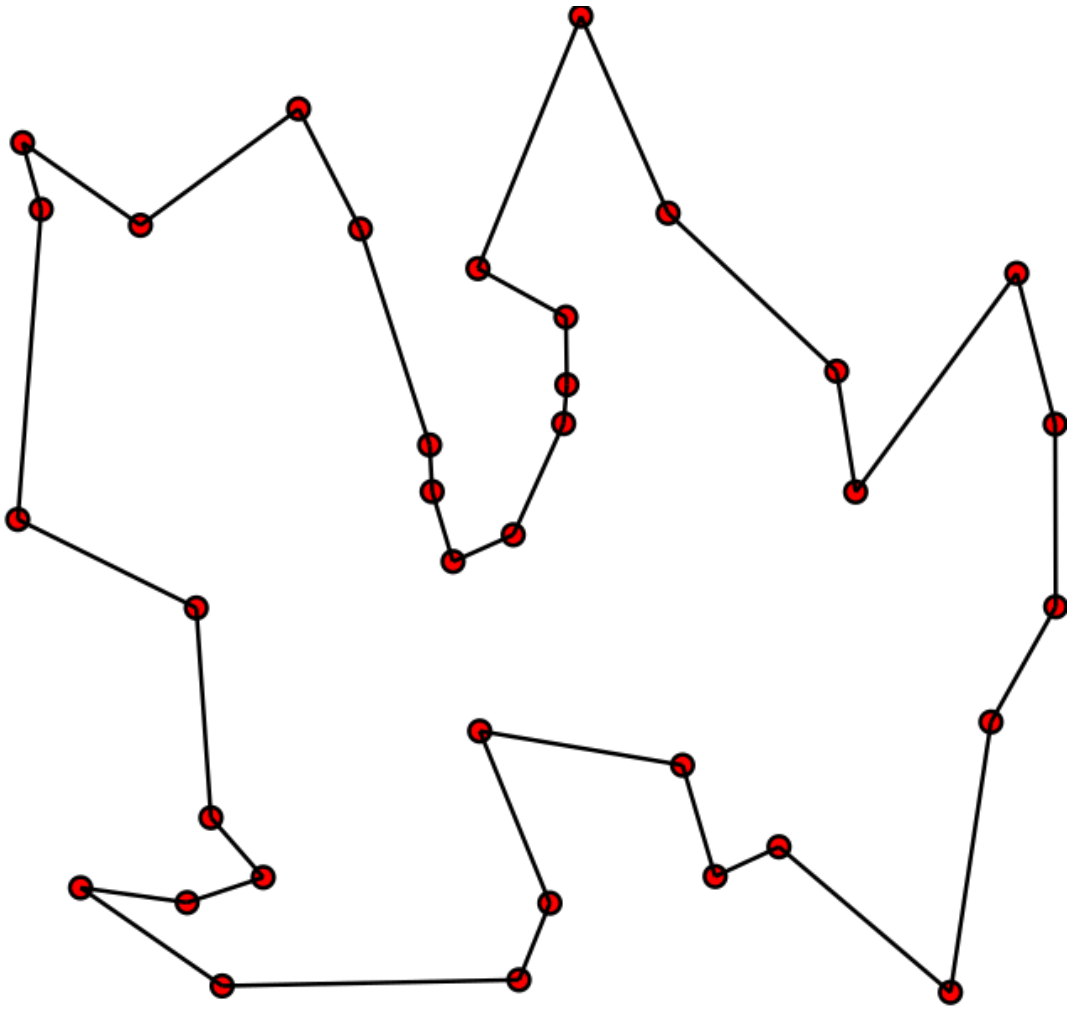
ALGORYTMY METAHEURYSTYCZNE

Problem komiwojażera

Sprawozdanie 1



Wiktoria Październiak i Sandra Szwed

Wrocław, 07.04.2022

WPROWADZENIE

Celem naszego zadania było zaimplementowanie algorytmów rozwiązujących Problem Komiwojażera (Travelling salesman problem), będącego problemem obliczeniowym polegającym na poszukiwaniu w grafie takiego cyklu, który zawiera wszystkie wierzchołki (każdy dokładnie raz) i ma jak najmniejszy koszt. Zaimplementowane przez nas algorytmy to: k-random, algorytm najbliższego sąsiada, 2-opt oraz powtarzalny algorytm najbliższego sąsiada.

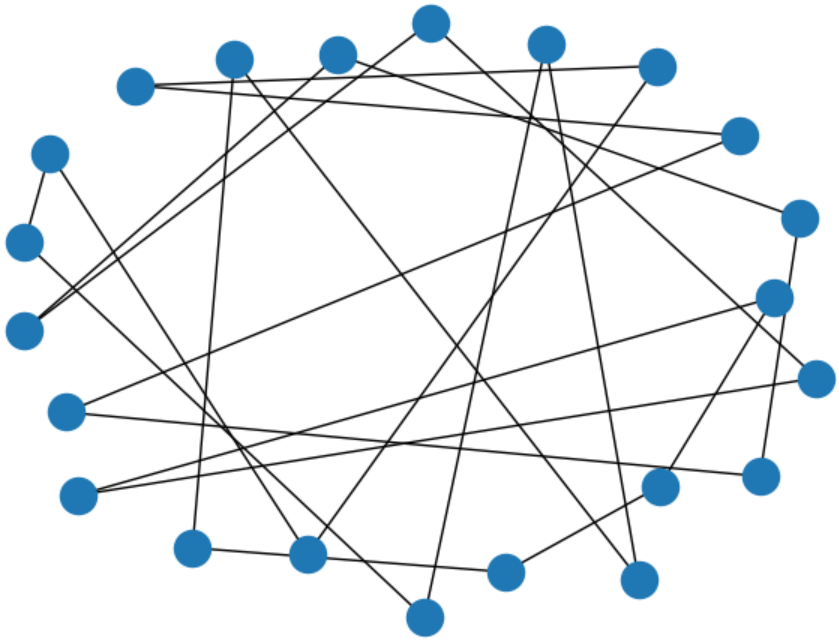
OSZACOWANA ZŁOŻONOŚĆ OBLICZENIOWA ZAIMPLEMENTOWANYCH PRZEZ NAS ALGORYTMÓW

W przypadku algorytmu najbliższego sąsiada szacowana przez nas złożoność obliczeniowa wynosi $O(n^2)$. Znajdowanie odległości między wierzchołkami generuje nam czynnik $O(n)$. Ponieważ operację tę wykonujemy dla każdego wierzchołka, (za każdym razem liczbę wierzchołków pomniejszamy o odwiedzone, jednak wciąż generuje to stały czynnik) asymptotyczne oszacowanie górne naszego algorytmu wynosi $O(n^2)$. W przypadku powtarzalnego algorytmu najbliższego sąsiada, skoro bazowa wersja tego algorytmu wykonywana jest n razy, możemy stwierdzić, że otrzymamy ostatecznie złożoność $O(n^3)$. Algorytm k-random natomiast ma złożoność $O(n)$ ponieważ k razy losujemy n elementów czyli jest to zależne od n.

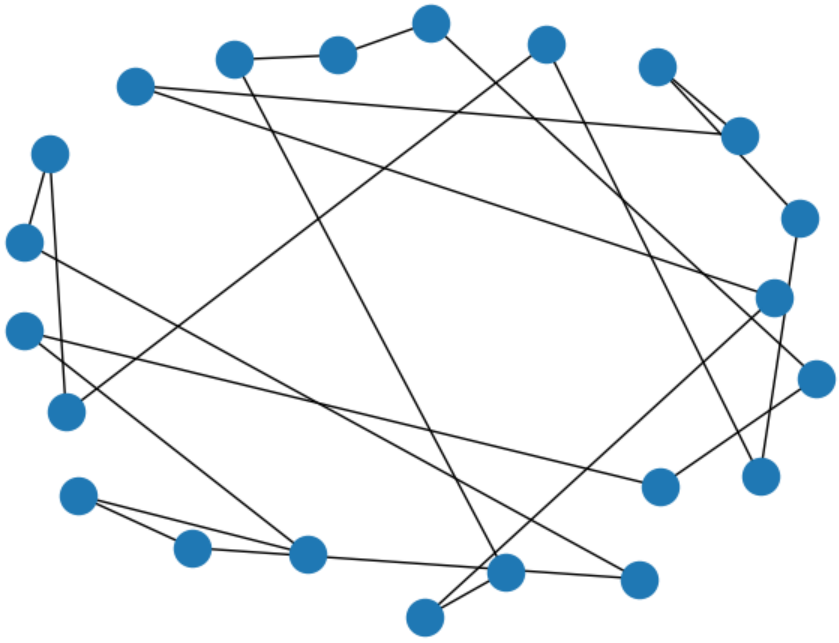
METODA WIZUALIZACJI INSTANCJI I ROZWIĄZANIA DLA KOMIWOJAŻERA EUKLIDESOWEGO

W celu wizualizacji trasy komiwojażera euklidesowego napisałyśmy funkcję generującą graficznie graf z zaznaczonymi wierzchołkami i krawędziami. Wizualizacja ta pomaga między innymi w potwierdzeniu naszych intuicji dotyczących algorytmu 2-opt, poprawiającego rozwiązanie przez „rozplątanie” charakterystycznych nieoptymalnych pętli pojawiających się na trasie komiwojażera. Poniżej przedstawiłyśmy kod źródłowy oraz wynik jego działania dla symetrycznej instancji ulysses22.tsp przed i po zastosowaniu algorytmu 2-opt. Porównując te dwa rysunki ze sobą można wyciągnąć wniosek, że zastosowanie algorytmu 2-opt znacznie uprościło trasę komiwojażera.

```
def print_tour(problem, tour_n=0, weights=False):
	g = nx.Graph()
	pos = None

	if problem.edge_weight_type == 'EUC_2D':
		for node in list(problem.get_nodes()):
			g.add_node(node, coord = problem.node_coords[node])
		pos = nx.get_node_attributes(g, 'coord')
	else:
		g.add_nodes_from(list(problem.get_nodes()))
		pos = nx.spring_layout(g, seed = 225)

	tour = problem.tours[tour_n]
	for i in range(0, len(tour) - 1):
		g.add_edge(tour[i], tour[i + 1],
			weight = problem.get_weight(tour[i], tour[i + 1]))
	g.add_edge(tour[-1], tour[0], weight = problem.get_weight(tour[-1],
tour[0]))

	nx.draw(g, pos)
	if weights:
		labels = nx.get_edge_attributes(g, 'weight')
		nx.draw_networkx_edge_labels(g, pos, edge_labels = labels)
	plt.show()
```



Trasa komiwojażera dla instancji ulysses22.tsp przed zastosowaniem algorytmu 2-opt



Trasa komiwojażera dla instancji ulysses22.tsp po zastosowaniu algorytmu 2-opt

OPIS EKSPERYMENTÓW

Podstawą ewaluacji naszych algorytmów były eksperymenty empiryczne, ponieważ trudno wyciągnąć o nich konstruktywne wnioski bez ich uprzedniego przetestowania.

1. Implementacja

   Nasze algorytmy implementowałyśmy w języku Python interpretowanym interpreterem python3. Podczas pisania kodu posługiwałyśmy się zintegrowanym środowiskiem programistycznym PyCharm.

2. Wymagania sprzętowe

   Sprzęt na którym wykonywałyśmy nasz eksperyment to MacBook Air, z 8-rdzeniowym procesorem M1, 8 GB pamięci RAM oraz Acer Nitro, z 6-rdzeniowym procesorem Intel Core i7-9750H, 16 GB pamięci RAM.

3. Instancje

   Nasze algorytmy badałyśmy zarówno na instancjach symetrycznych, jak i niesymetrycznych, pochodzących z biblioteki TSPLIB. Za punkt odniesienia efektywności naszych algorytmów przyjęłyśmy długość optymalnego cyklu Hamiltona dla danej instancji, dostępnej również w bibliotece TSPLIB. W naszym programie zaimplementowałyśmy metodę służącą do losowego generowania instancji:

   ```
   def random_solve(problem):
         nodes = list(problem.get_nodes())
         np.random.shuffle(nodes)
         problem.tours.append(nodes)

         return nodes
   ```

   W programie używałyśmy różnych form instancji. Były to między innymi forma euklidesowa, dla której przygotowałyśmy również metodę wizualizacji instancji i rozwiązania, oraz macierz odległości.

4. Metodologia przeprowadzanych eksperymentów i opis wyników.

Najprostszym podejściem do uzyskania obiektywnych porównań między algorytmami jest

zestawienie wartości ich funkcji celu dla danej instancji:

RODZAJ	INSTANCJA	NAZWA PLIKU	2-OPT	K-RANDOM	NN	RNN	OPTYMALNE
Symetryczna	Euklidesowa	gr96.tsp	58400	326067	70916	63945	55209
Symetryczna	Euklidesowa	att48.tsp	10730	43417	12861	12012	10628
Symetryczna	Euklidesowa	ulysses22.tsp	7294	13234	10586	8180	7013
Symetryczna	Euklidesowa	berlin52.tsp	8522	26517	8980	8181	7541
Symetryczna	Euklidesowa	st70.tsp	716	3433	830	796	675
Symetryczna	Euklidesowa	rd100.tsp	8366	51424	9938	9423	7910
Symetryczna	Macierzowa	bays29.tsp	2084	5301	2258	2134	2020
Symetryczna	Euklidesowa	a280.tsp	2851	32849	3157	2975	2579
Symetryczna	Euklidesowa	bayg29.tsp	1630	4428	2005	1935	1610
Symetryczna	Euklidesowa	brazil58.tsp	27072	108530	28429	27384	25395
Symetryczna	Euklidesowa	burma4.tsp	3371	5606	4048	3841	3323
Symetryczna	Euklidesowa	ch150.tsp	7003	50517	8191	7113	6528
Symetryczna	Macierzowa	dantzig42.tsp	758	2608	956	864	699
Niesymetryczna	Macierzowa	br17.atsp	39	162	94	56	39
Niesymetryczna	Macierzowa	ft70.atsp	53663	70785	42515	41815	38673
Niesymetryczna	Macierzowa	ftv33.atsp	1883	3784	1608	1590	1286
Niesymetryczna	Macierzowa	p43.atsp	5628	17329	5689	5684	5620
Niesymetryczna	Macierzowa	ry48p.atsp	15882	45058	16414	15575	14442
Niesymetryczna	Macierzowa	kro124p.atsp	57237	174290	44468	43316	36230
Niesymetryczna	Macierzowa	ftv64.atsp	4226	8100	2612	2202	1839
Niesymetryczna	Macierzowa	ftv38.atsp	2616	4483	1774	1759	1530

W naszym zestawieniu, w tabeli figurują wartości funkcji celu odpowiednio dla algorytmu: 2-opt, k-random, najbliższego sąsiada oraz powtarzalnego najbliższego sąsiada. Dla wyciągnięcia lepszych wniosków można posłużyć się również tabelą

zawierającą przyrosty względne dla poszczególnych algorytmów względem rozwiązania optymalnego:

RODZAJ	INSTANCJA	NAZWA PLIKU	2-OPT	K-RANDOM	NN	RNN	OPTYMALNE
Symetryczna	Euklidesowa	gr96.tsp	5,779	490,605	28,45	15,823	55209
Symetryczna	Euklidesowa	att48.tsp	0,959	308,515	21,01	13,022	10628
Symetryczna	Euklidesowa	ulysses22.tsp	4,007	88,706	50,948	16,640	7013
Symetryczna	Euklidesowa	berlin52.tsp	12,993	251,591	19,066	8,472	7542
Symetryczna	Euklidesowa	st70.tsp	6,074	408,592	22,962	17,925	675
Symetryczna	Euklidesowa	rd100.tsp	5,764	550,113	25,637	19,127	7910
Symetryczna	Macierzowa	bays29.tsp	3,168	162,425	11,782	5,64	2020
Symetryczna	Euklidesowa	a280.tsp	10,546	1173,71	22,411	15,354	2579
Symetryczna	Euklidesowa	bayg29.tsp	1,242	175,031	24,534	20,186	1610
Symetryczna	Euklidesowa	brazil58.tsp	6,603	327,367	11,947	7,832	25395
Symetryczna	Euklidesowa	burma4.tsp	1,444	68,702	21,817	15,58	3323
Symetryczna	Euklidesowa	ch150.tsp	7,276	673,851	25,474	8,9613	6528
Symetryczna	Macierzowa	dantzig42.tsp	8,440	273,104	36,766	23,605	699
Niesymetryczna	Macierzowa	br17.atsp	0	315,384	141,025	43,588	39
Niesymetryczna	Macierzowa	ft70.atsp	38,760	83,035	9,93	8,124	38673
Niesymetryczna	Macierzowa	ftv33.atsp	46,423	194,246	25,03	23,639	1286
Niesymetryczna	Macierzowa	p43.atsp	0,142	208,345	1,22	1,1387	5620
Niesymetryczna	Macierzowa	ry48p.atsp	9,970	211,992	13,63	7,845	14442
Niesymetryczna	Macierzowa	kro124p.atsp	57,9823	381,065	22,74	19,558	36230
Niesymetryczna	Macierzowa	ftv64.atsp	129,7988	340,457	42,03	19,73	1839
Niesymetryczna	Macierzowa	ftv38.atsp	70,9804	193,007	15,95	14,967	1530

Pierwsze wnioski jakie nasuwają się po pobieżnym spojrzeniu na wyniki są takie, że

algorytmy, które dobrze sprawdzają się dla instancji symetrycznych niekoniecznie dają już tak dobre rezultaty w przypadku instancji niesymetrycznych. Przykładem takiego algorytmu jest 2-opt, który sprawdzając się najlepiej dla instancji symetrycznych, często wypada gorzej w stosunku do algorytmu najbliższego sąsiada, jeżeli w grę wchodzą instancje niesymetryczne.

Zaimplementowane przez nas algorytmy można również porównywać parami. I tak w przypadku algorytmów k-random oraz najbliższego sąsiada przyjęłyśmy następującą metodologię działania:

Jako że nie jesteśmy w stanie wpłynąć na czas wykonania algorytmu najbliższego sąsiada, zmierzyłyśmy czas jego działania. Następnie, mając świadomość, że wykonując algorytm k-random możemy manipulować czasem jego wykonywania, dla k z zakresu od 1 do 100 mierzyłyśmy czas jego działania dla każdej iteracji i porównywałyśmy wartość bezwzględną różnicy uzyskanych czasów (najbliższego sąsiada i k-random). Najmniejszy z uzyskanych wyników dał nam możliwość w miarę obiektywnego porównywania algorytmów.

Wyniki zestawiłyśmy w poniższej tabeli:

RODZAJ	INSTANCJA	NAZWA PLIKU	K-RANDOM	K	NN	CZAS WYWOŁANIA
Symetryczna	Euklidesowa	gr96.tsp	32928	43	70916	0.0154027
Symetryczna	Euklidesowa	att48.tsp	45685	20	12861	0.0026762
Symetryczna	Euklidesowa	ulysses22.tsp	10868	10	10586	0.0009211
Symetryczna	Euklidesowa	berlin52.tsp	26047	22	8980	0.0029562
Symetryczna	Euklidesowa	st70.tsp	3303	29	830	0.0052274
Symetryczna	Euklidesowa	rd100.tsp	50869	42	9938	0.0104482
Symetryczna	Macierzowa	a280.tsp	31555	97	3157	0.0810104
Symetryczna	Euklidesowa	bays29.tsp	4323	9	2005	0.0009901
Symetryczna	Euklidesowa	bayg29.tsp	4323	9	2005	0.0009901
Symetryczna	Euklidesowa	pcb442.tsp	730025	98	61979	0.1991166
Symetryczna	Euklidesowa	bayg29.tsp	3800	9	2005	0.0009437

Symetryczna	Euklidesowa	brazil58.tsp	110921	18	28429	0.0037195
Symetryczna	Euklidesowa	burma4.tsp	6214	7	4048	0.0037195
Symetryczna	Euklidesowa	ch150.tsp	51389	64	8191	0.0236529
Symetryczna	Macierzowa	dantzig42.tsp	2778	13	956	0.0016892
Niesymetryczna	Macierzowa	br17.atsp	164	5	94	0.0002567
Niesymetryczna	Macierzowa	ft70.atsp	68940	21	42515	0.0033874
Niesymetryczna	Macierzowa	ftv33.atsp	3819	10	1608	0.0009036
Niesymetryczna	Macierzowa	p43.atsp	22519	13	5689	0.0013564
Niesymetryczna	Macierzowa	ry48p.atsp	47913	14	16414	0.00159645
Niesymetryczna	Macierzowa	ftv64.atsp	7962	20	2612	0.0029034
Niesymetryczna	Macierzowa	ftv38.atsp	4499	11	1774	0.0011945

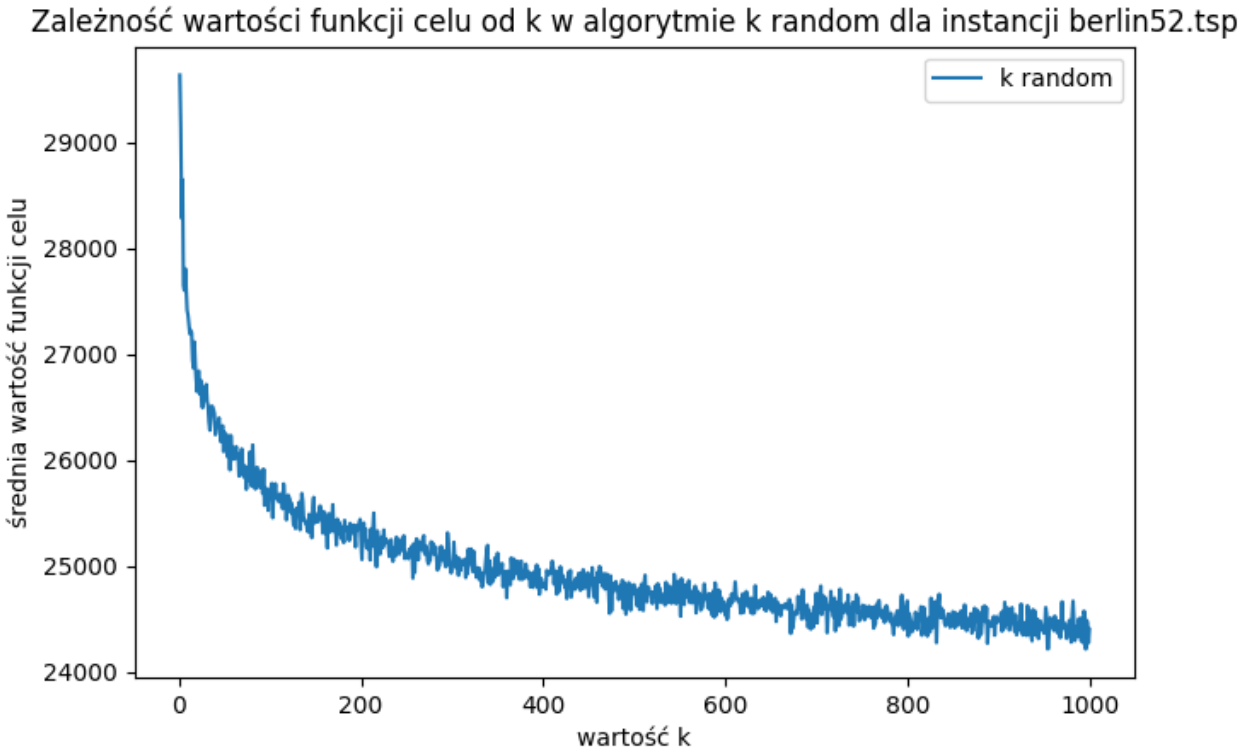
W naszych eksperymentach badałyśmy również wpływ wyboru k w algorytmie k-random na wartość funkcji celu.

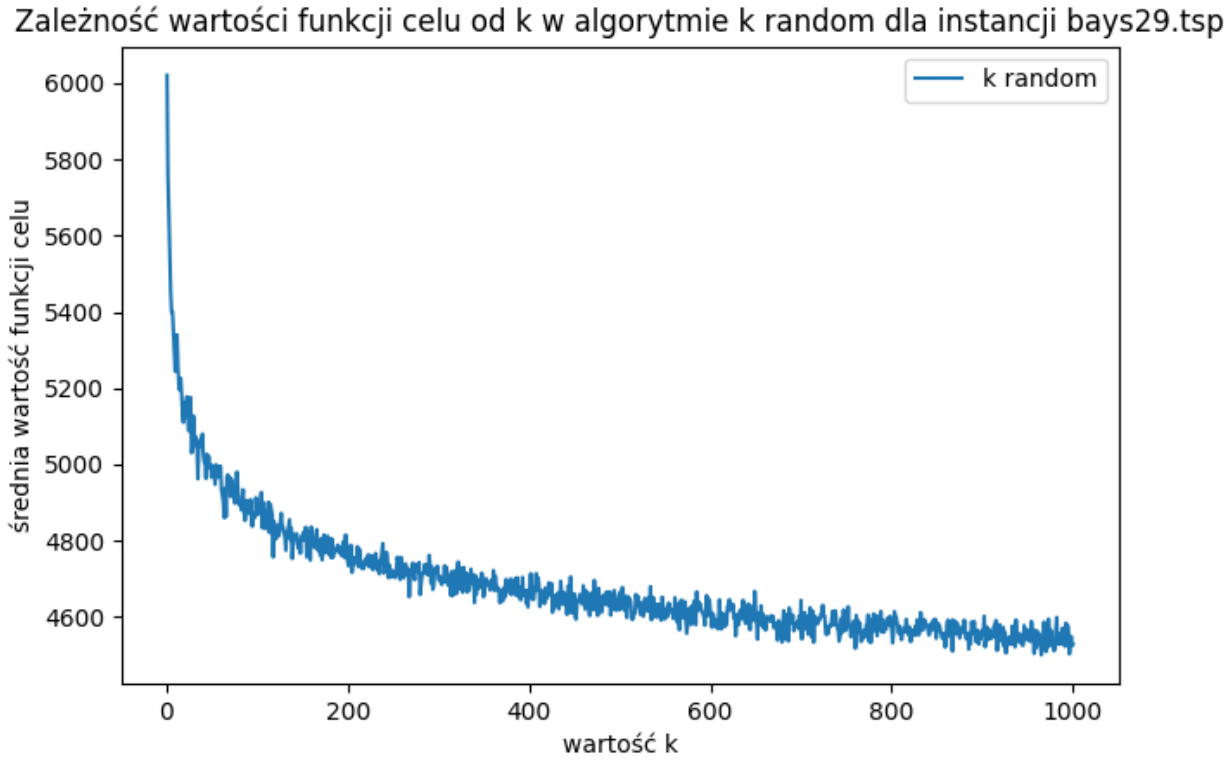
Ponieważ wiemy, że aby zredukować ryzyko otrzymania bardzo słabego rozwiązania w algorytmie k - random generuje się k losowych rozwiązań i jako wynik zwraca się najlepszego napotkanego kandydata. Zatem im większa próba tym większa szansa na uzyskanie dobrego rozwiązania o czym będą świadczyć coraz to mniejsze wartości funkcji celu uzyskiwane przez nas wraz ze wzrostem k.

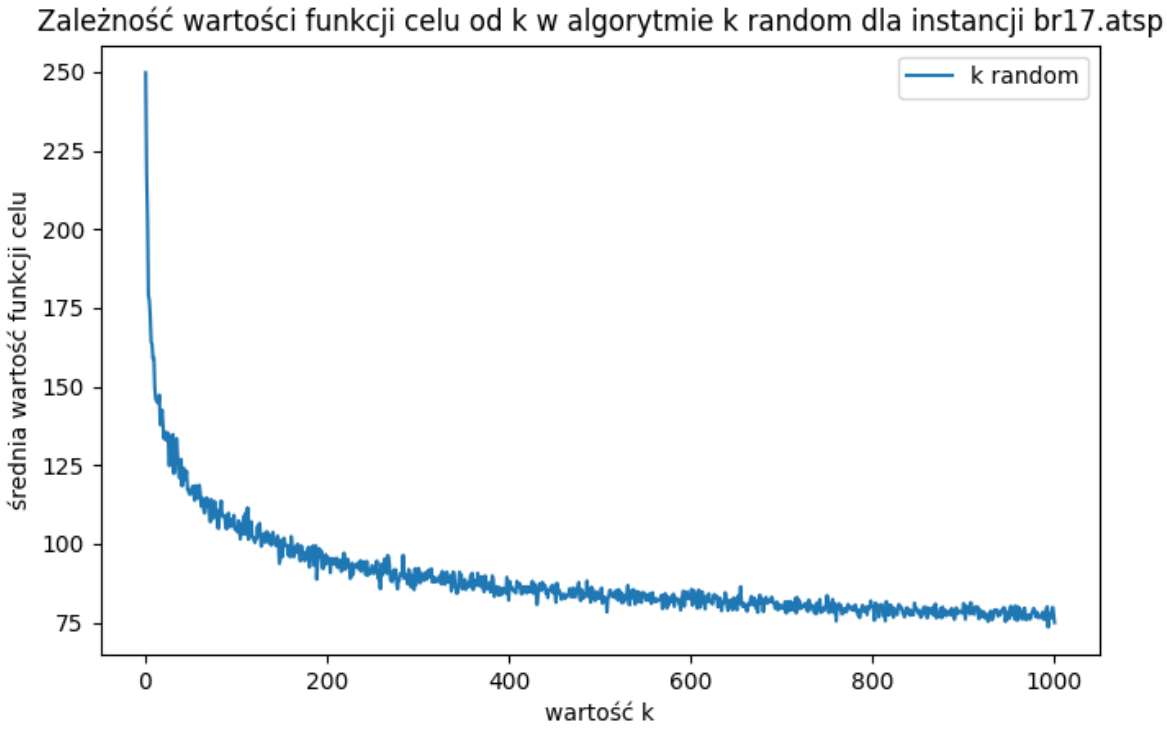
Badanie wpływu k na średnią wartość funkcji celu w algorytmie k-random przeprowadziłyśmy dla 3 różnych instancji, dwóch symetrycznych(euklidesowej i macierzowej) oraz jednej niesymetrycznej.

Przy generowaniu wykresów przyjęłyśmy następującą metodologię działania:

Dla każdej wartości k z zakresu od 1 do 1000, uruchamiałyśmy algorytm k-random 100 razy. Następnie brałyśmy średnią arytmetyczną uzyskanych wyników, którą następnie wykorzystałyśmy do wygenerowania wykresów zależności k od wartości funkcji celu w algorytmie k-random.


Zależność wartości funkcji celu od k w algorytmie k random dla instancji berlin52.tsp
k random
średnia wartość funkcji celu
wartość k
29000
28000
27000
26000
25000
24000
0
200
400
600
800
1000


Zależność wartości funkcji celu od k w algorytmie k random dla instancji bays29.tsp
k random
średnia wartość funkcji celu
wartość k
6000
5800
5600
5400
5200
5000
4800
4600
0
200
400
600
800
1000



WNIOSKI KOŃCOWE

Powyższe eksperymenty pozwalają nam wyciągnąć wnioski na temat efektywności zaimplementowanych przez nas algorytmów. W ogólnym zestawieniu, dla instancji symetrycznych najlepiej wypada algorytm 2-opt. Jeżeli chodzi o instancje niesymetryczne, jest to powtarzalna wersja algorytmu najbliższego sąsiada.

Porównując algorytmy k-random oraz algorytm najbliższego sąsiada, dając im taki sam budżet czasowy, algorytm najbliższego sąsiada wypada zdecydowanie lepiej w stosunku do algorytmu k-random. Generalnie algorytm k-random wypada najsłabiej na tle innych algorytmów, niezależnie od instancji na której był wykonywany.

Nasze eksperymenty pozwoliły nam również potwierdzić zasadność manipulowania parametrem k w algorytmie k-random. Analizując wykonane przez nas wykresy można wyciągnąć wniosek, że parametr ten pozwala sterować balansem pomiędzy czasem działania a statystyczną jakością wyników.

Link do repozytorium z kodem źródłowym:
https://github.com/wikcia/Algorytmy_metaheurystyczne