ROZWIĄZANIE PROBLEMU KOMIWOJAŻERA ALGORYTMEM ŁĄCZĄCYM SYMULOWANE WYŻARZANIE Z PARALLEL TEMPERING

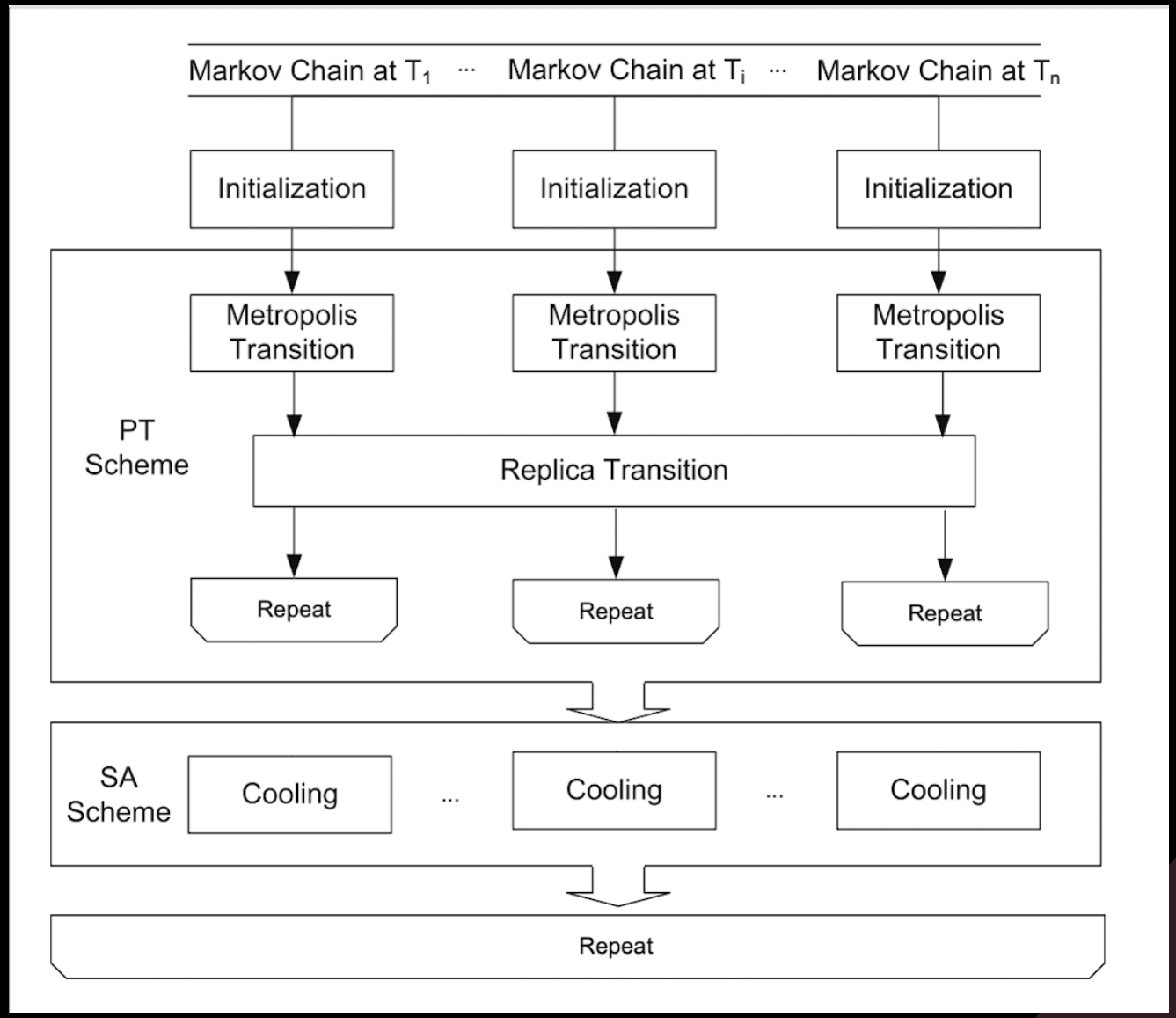
Łukasz Grabarski, Magdalena Jeczeń, Karolina Mączka, Mateusz Nizwantowski, Marta Szuwarska

Plan prezentacji

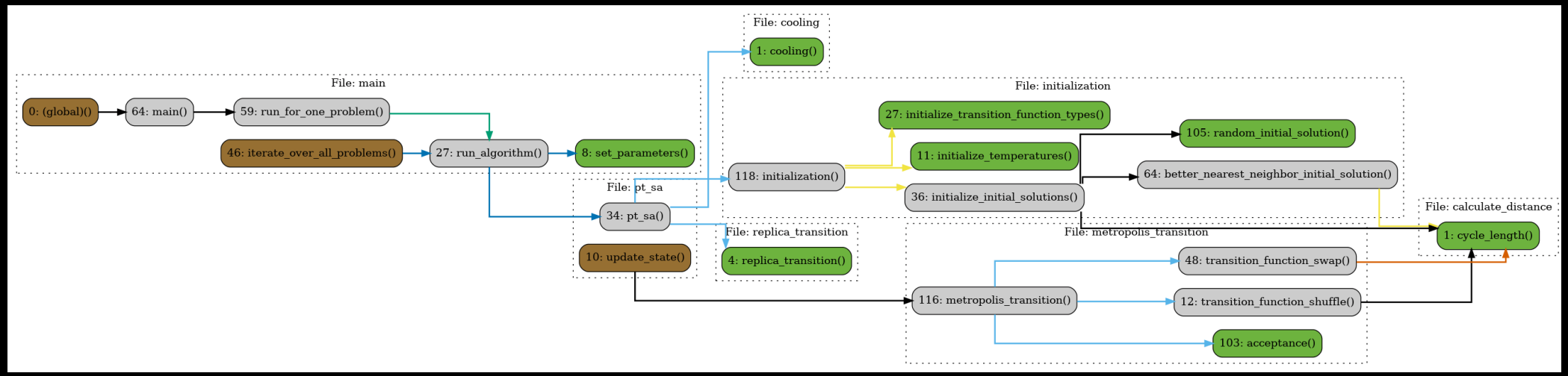
- Algorytm
- Modyfikacje algorytmu
- Parametry
- Testowanie parametrów
- Ostateczne wyniki

Algorytm

General idea



Nasza Implementacja



Modyfikacje

Zaczynanie od rozwiązań heurystycznych

```
1  function better_nearest_neighbor_initial_solution:
2    heuristic_solutions = []
3    for every city as starting_city do:
4      unvisited = list of cities
5      current_city = starting_city
6      path = [current_city]
7      unvisited.remove(current_city)
8      while unvisited:
9        nearest_neighbor = nearest city to current city
10       unvisited.remove(nearest_neighbor)
11       path.append(nearest_neighbor)
12       current_city = nearest_neighbor
13     heuristic_solutions.append(path)
14   heuristic_solutions.sort()
15   return heuristic_solutions[: math.floor(len(distance_matrix) * 0.1)]
```

Zaczynanie od rozwiązań heurystycznych

```
1  nearest_neighbor_solution =
2  better_nearest_neighbor_initial_solution(distance_matrix)
3  for n times:
4      if eps < probability_of_heuristic:
5        initial_solutions.append(random.choice(nearest_neighbor_solution))
```

Jak dużo dały rozwiązania heurystyczne?

Problem	PTSA	Małpki	**N**	Heurystyka
FTV38	1608	1779	1652	1759
FT70	40218	46864	40286	41815
RBG403	2957	4371	7019	3497

Dla wszystkich 17 problemów nasze rozwiązanie heurystyczne liczy się w 5.6 sekundy, produkując na ogół rozwiązania gorsze o 22.4% od optymalnego.

Implementacja dwóch typów metropolis transition

Obrazek 1, [1]

Shuffle transition

```
1  function transition_function_shuffle:
2    list_to_shuffle = []
3    l = length(solution)
4    start_index = random index
5    repeat
6    for i from 0 to transformation_length - 1:
7      list_to_shuffle[i] = solution[(start_index + i) mod l]
8
9    shuffle(list_to_shuffle)
10
11   new_solution = copy(solution)
12   for i from 0 to transformation_length - 1:
13     new_solution[(start_index + i) mod l] = list_to_shuffle[i]
14
15   new_solution_length = cycle_length(new_solution, distance_matrix)
16   return new_solution, new_solution_length
```

Swap transition

```
1  function transition_function_swap:
2    l = length(solution)
3
4    start_index_first_path = random index
5    start_index_second_path = random index
6
7    if (not first_path_can_go_right) and second_path_can_go_right:
8      start_index_first_path =
9      start_index_first_path - transformation_length + 1
10
11   else if first_path_can_go_right and (not second_path_can_go_right):
12     start_index_second_path =
13     start_index_second_path - transformation_length + 1
14
15   first_path = path from start_index_first_path
16   second_path = path from start_index_second_path
17
18   new_solution = solution.swap(first_path, second_path)
19   new_solution_length = cycle_length(new_solution, distance_matrix)
20
21   return new_solution, new_solution_length
```

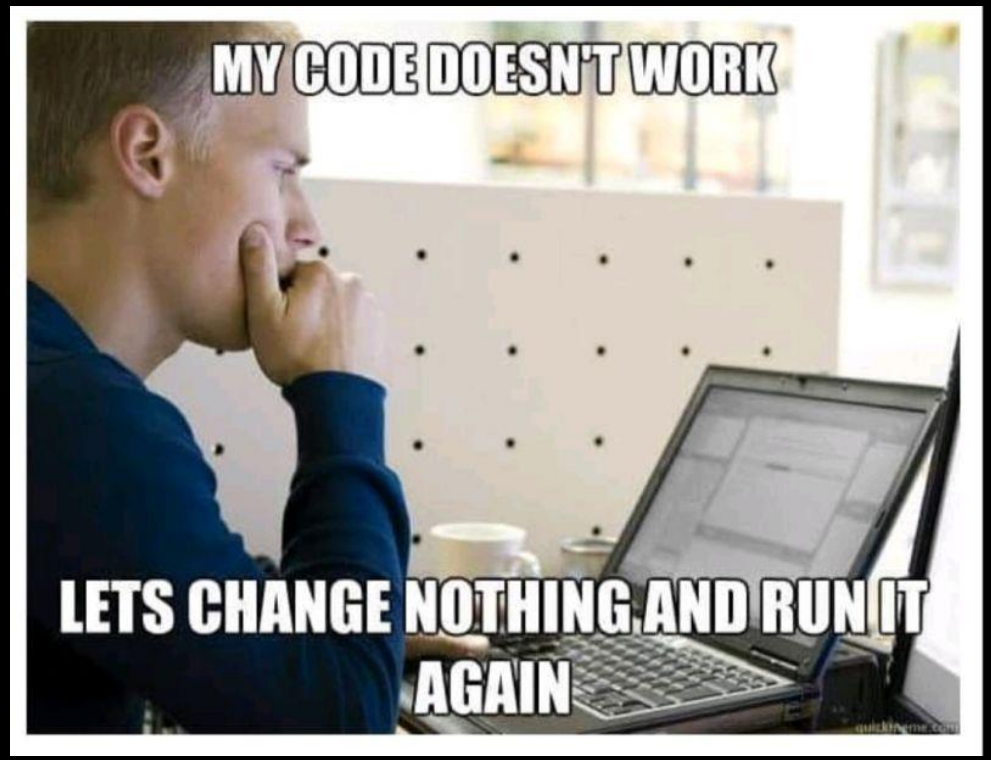
Przyśpieszenie metropolis transition za pomocą wielowątkowości

```
1  function update_state:
2    solution, solution_length = metropolis_transition()
3    with lock:
4      solutions[state], solutions_lengths[state] = solution, solution_length
5
6  function pt_sa:
7    ...
8    threads = []
9      lock = threading.Lock()
10     for state in range(n):
11       thread = new Thread(update_state)
12       thread.start()
13       threads.append(thread)
14     for thread in threads:
15       thread.join()
16   ...
```

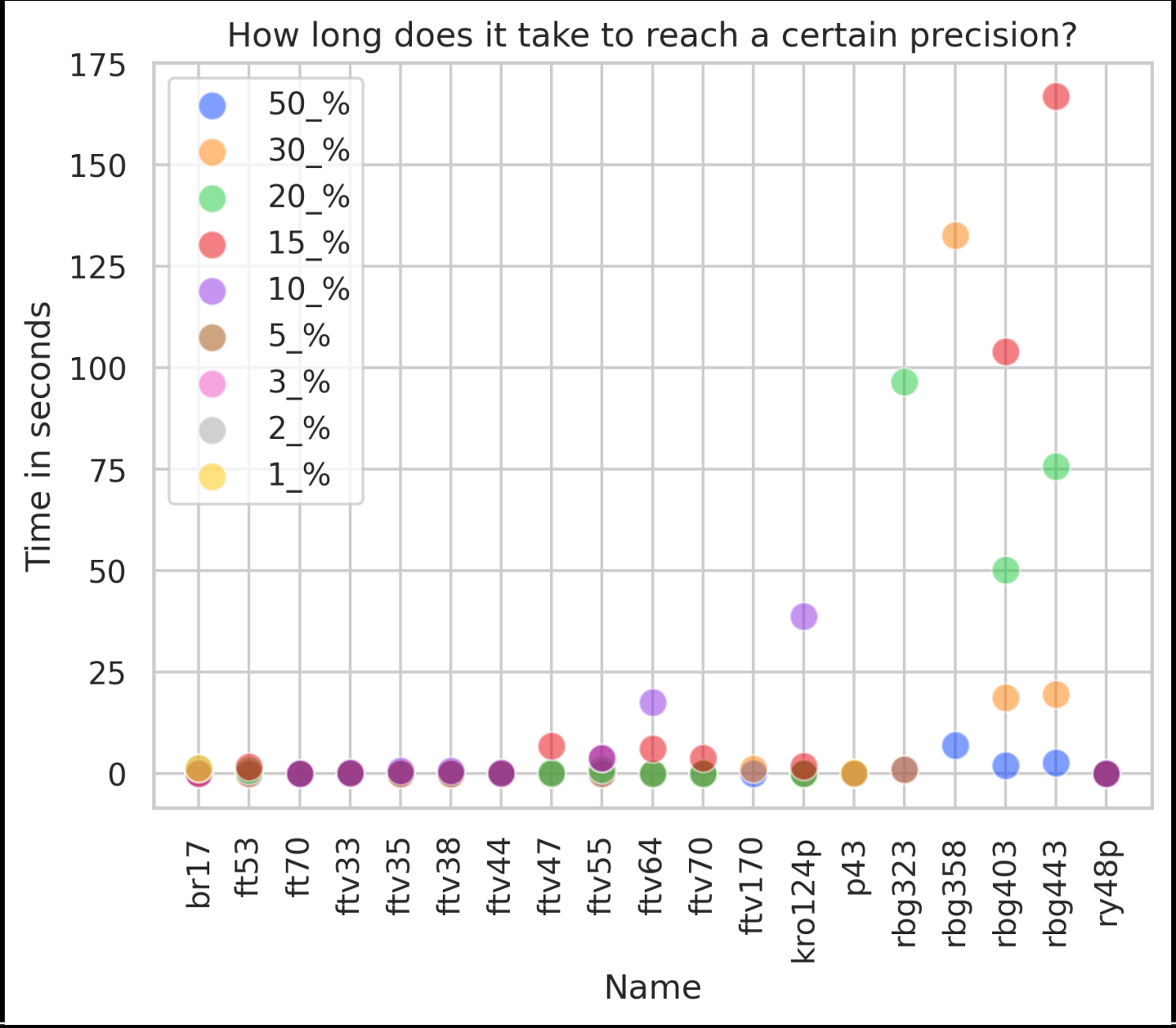
Pominięcie rozwiązań bliskich najlepszemu rozwiązaniu w replica transition

```
1  first_solution_close_to_best =
2  solutions_length[first_index] <= closeness * best_solution_length
3    second_solution_close_to_best =
4    solutions_length[second_index] <= closeness * best_solution_length
5
6    if first_solution_close_to_best or second_solution_close_to_best:
7      pass
8    else:
9      ...
```

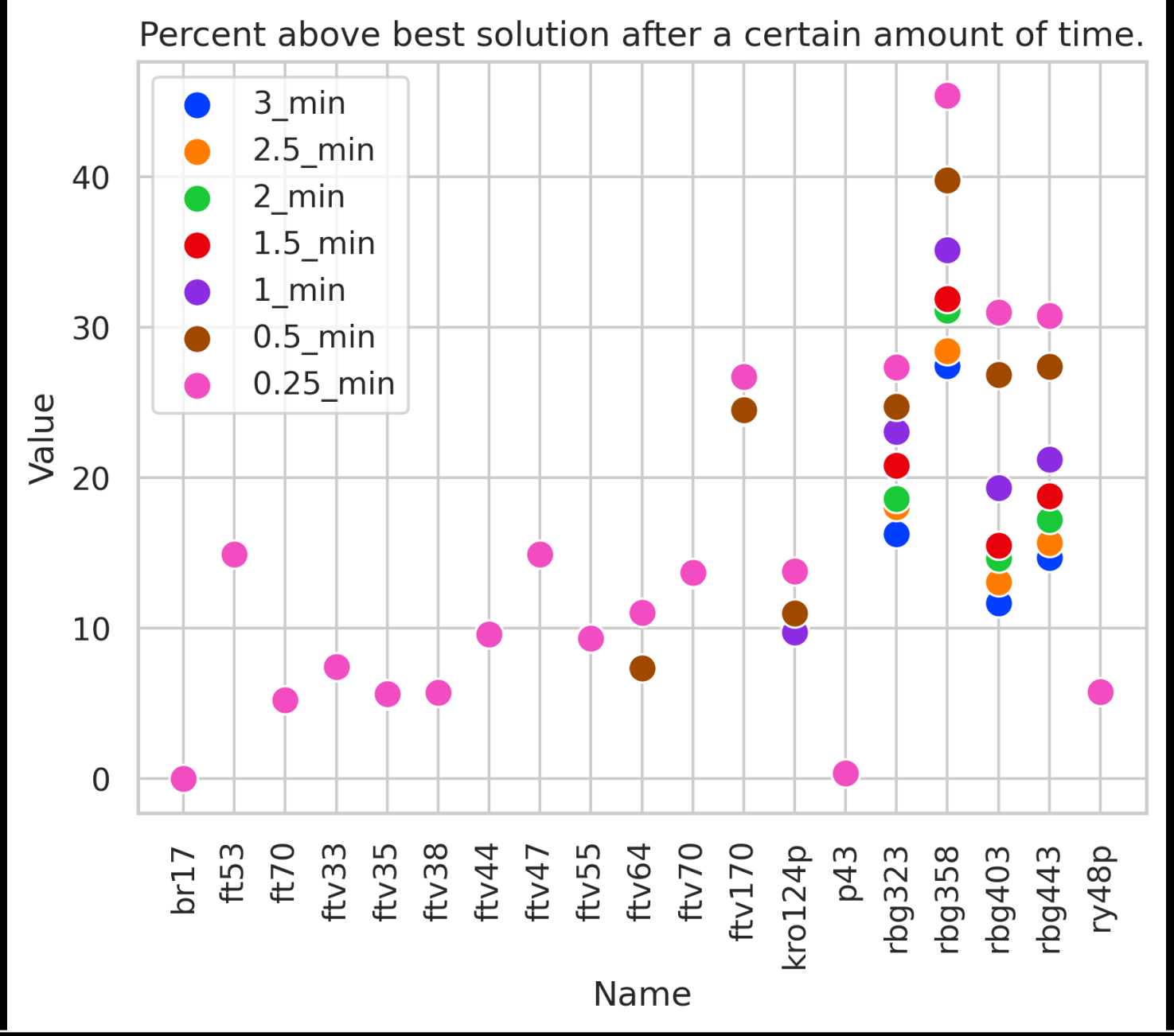
Wielokrotne uruchamianie algorytmu w ciągu 5 minut



Obrazek 2, [2]



Porównanie czasu potrzebnego do osiągnięcia danej precyzji



Porównanie precyzji osiągniętej po danej ilości czasu

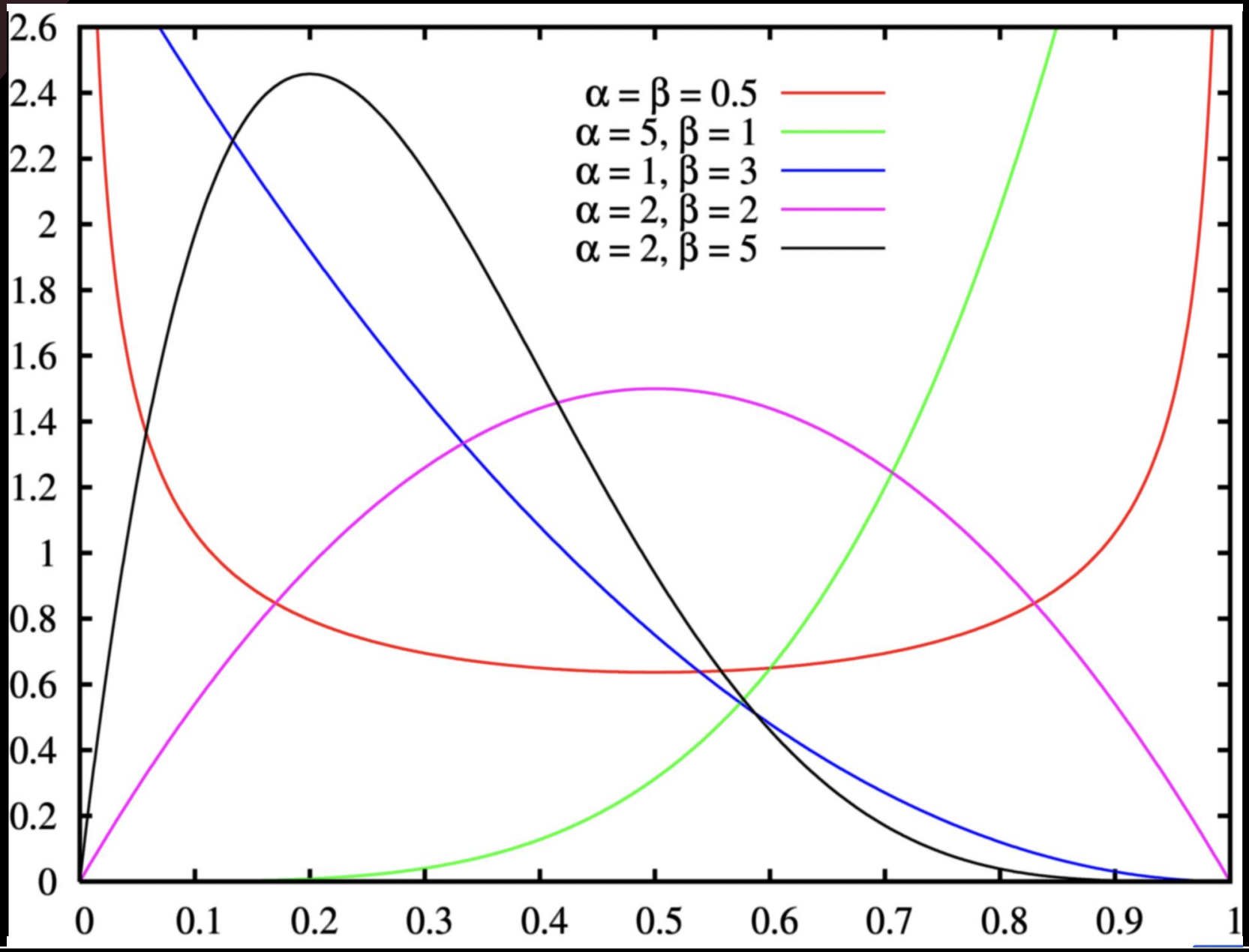
Wielokrotne uruchamianie algorytmu w ciągu 5 minut

```
1  function run_algorithm:
2    N = len(distance_matrix)
3    if N < 300:
4      num_of_runs = 5 min / 30 s
5    else:
6      num_of_runs = 5 min / 2.5 min
7    for num_of_runs times:
8      solution, solution_length = pt_sa(distance_matrix, **parameters)
9      if solution_length < best_solution_length:
10       best_solution, best_solution_length = solution, solution_length
11   return best_solution, best_solution_length
```

Parametry

Lista parametrów jakich używamy:

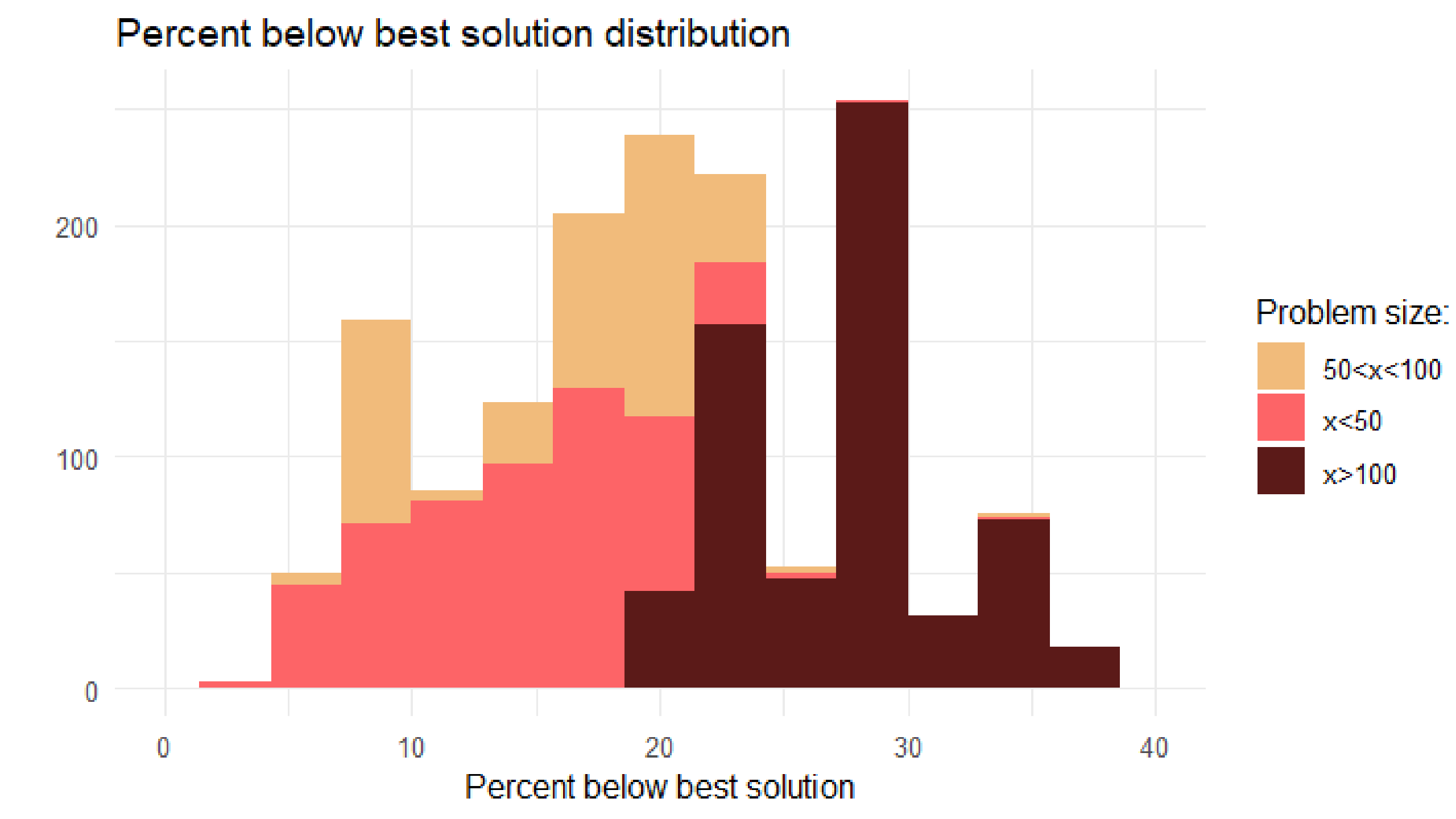
- n - liczba stanów
- min_temperature - minimalna temperatura,
- max_temperature - maksymalna temperatura,
- probability_of_shuffle - prawdopodobienstwo użycia shuffle transition
- probability_of_heuristic - prawdopodobienstwo użycia rozwiązania heurystycznego
- a - pierwszy parametr rozkładu beta
- b - drugi parametr rozkładu beta

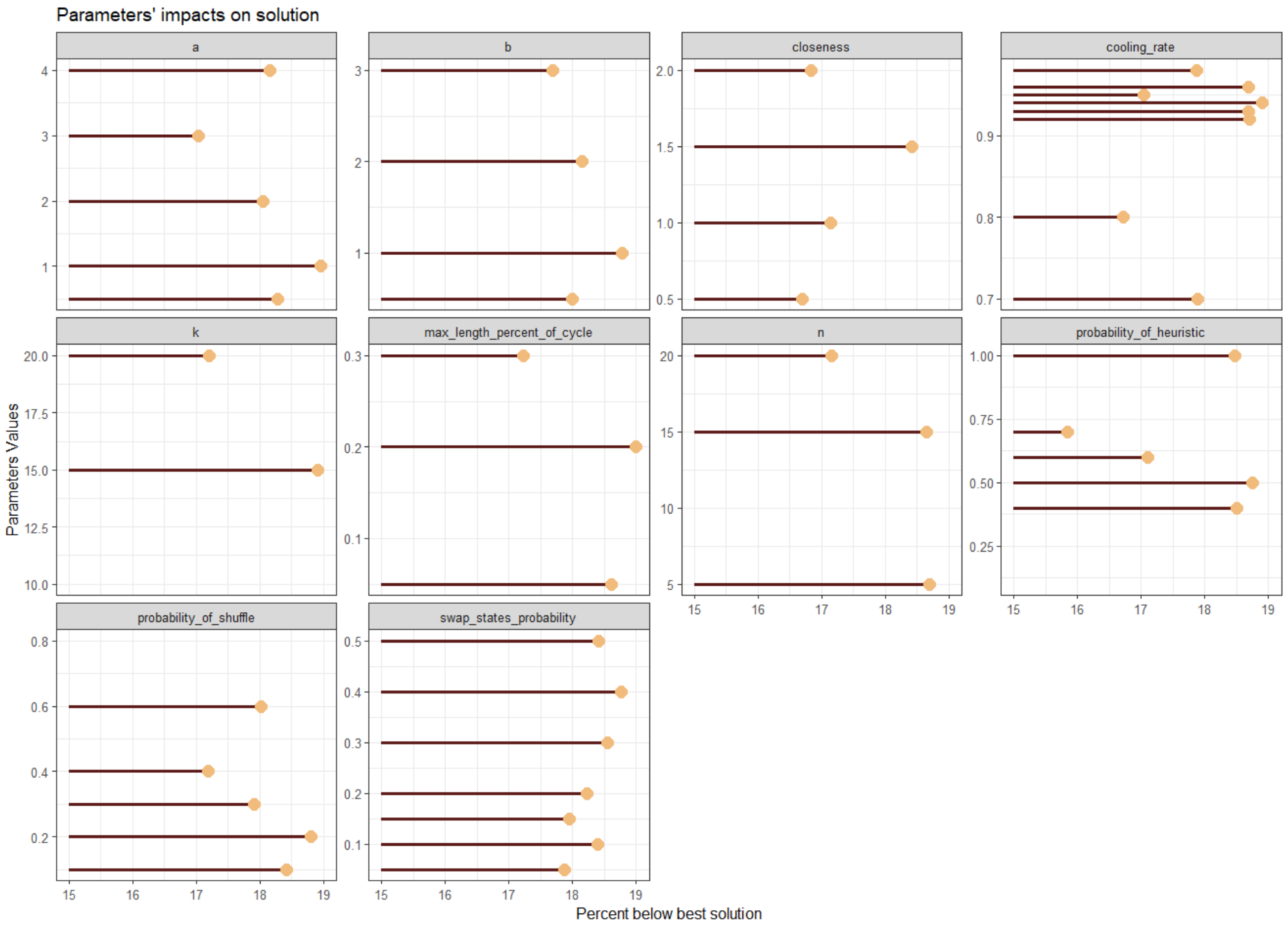


Rozkład beta dla różnych parametrów

- duration_of_execution_in_seconds - czas, przez jaki algorytm działa,
- k - liczba wykonań fazy PT zanim przystąpimy do fazy SA,
- max_length_percent_of_cycle - jak długie ścieżki zmieniamy w metropolis transitions,
- swap_states_probability - prawdopodobieństwo zamiany temperatur pomiędzy stanami,
- closeness - jak blisko trzeba być od najlepszego rozwiązania, żeby nie zmieniać temperatury w replica tranistion,
- cooling_rate - szybkość chłodzenia.

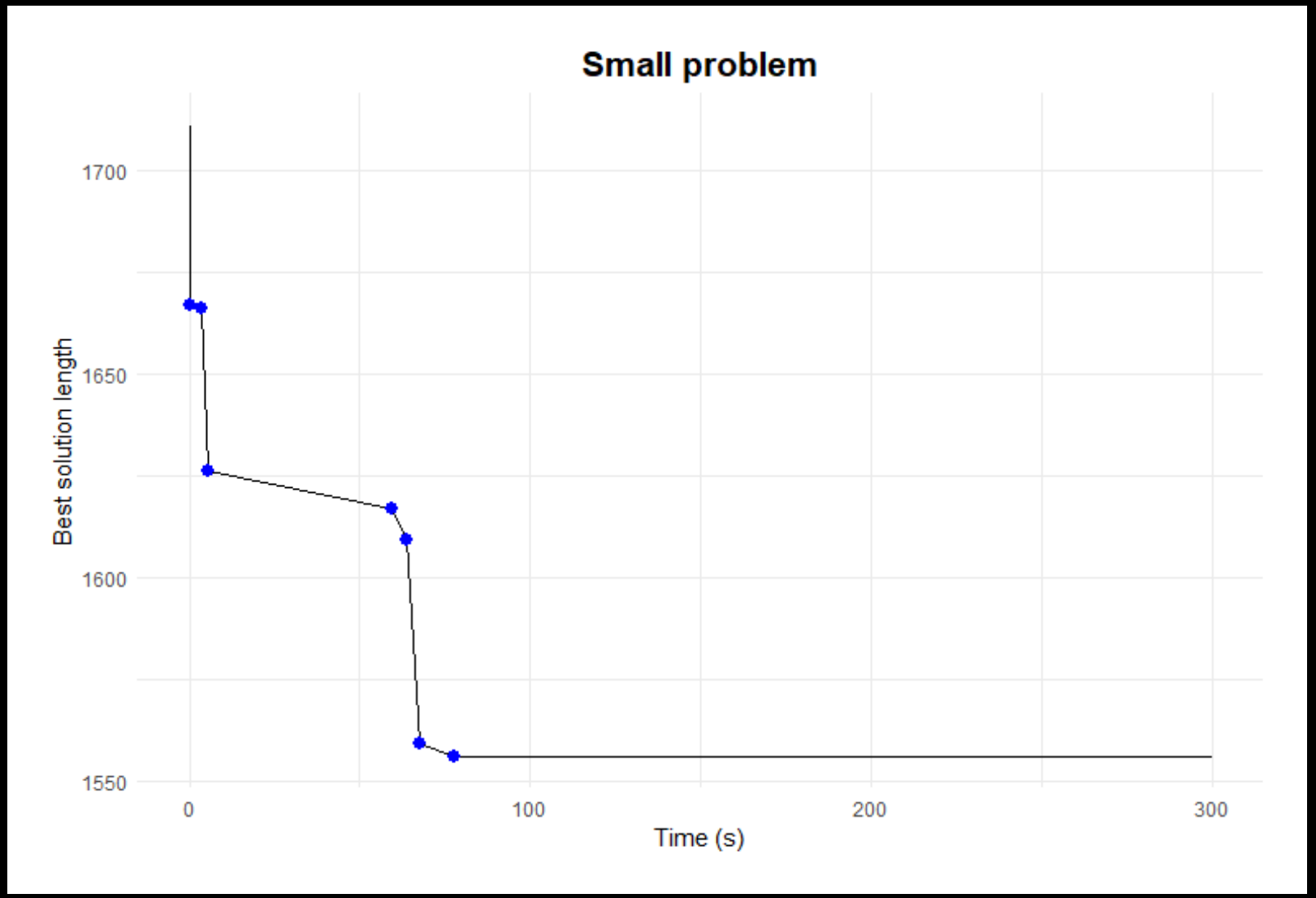
Testowanie parametrów

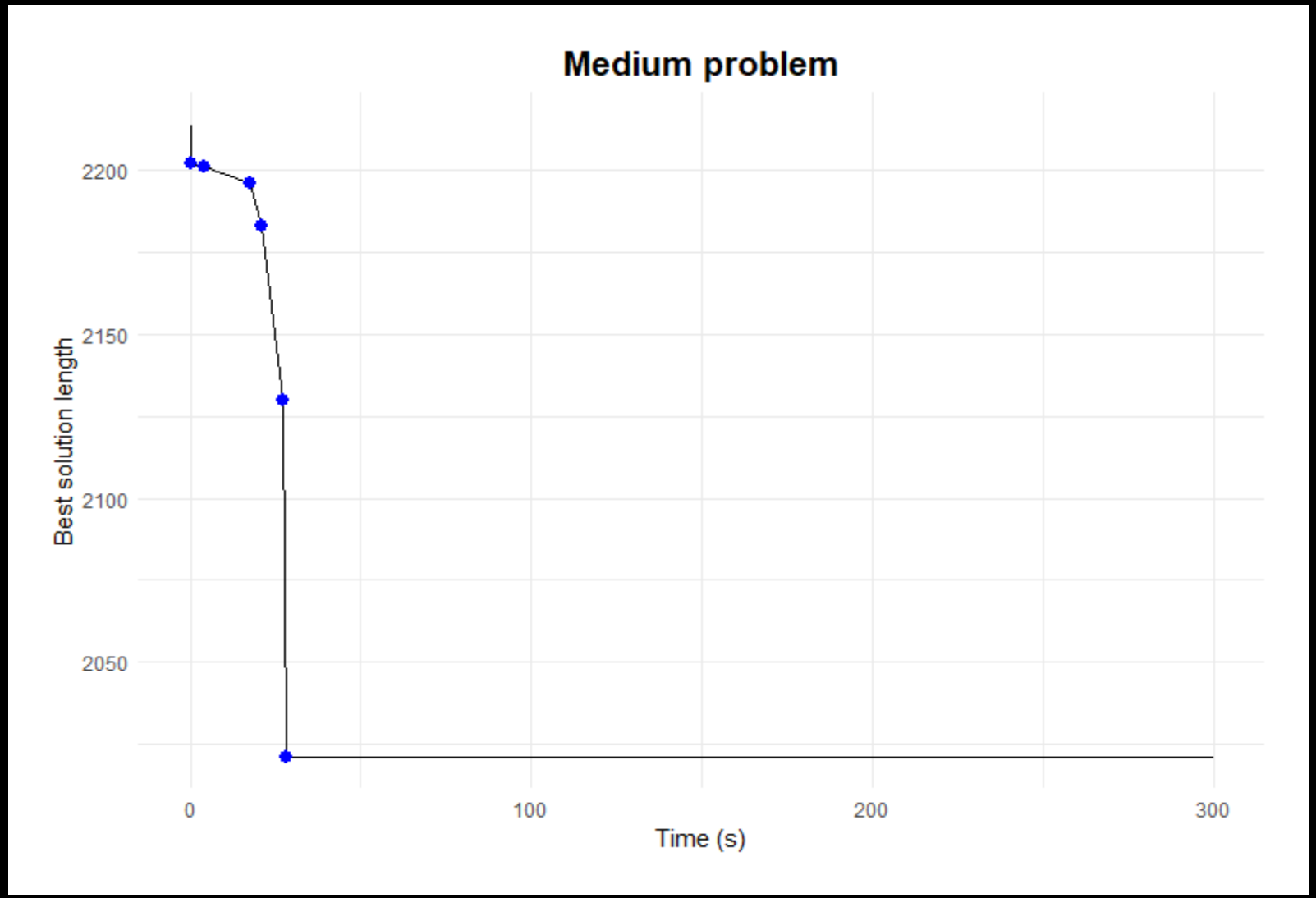

Percent below best solution distribution
200
100
0
0
10
20
30
40
Percent below best solution
Problem size:
50<x<100
x<50
x>100

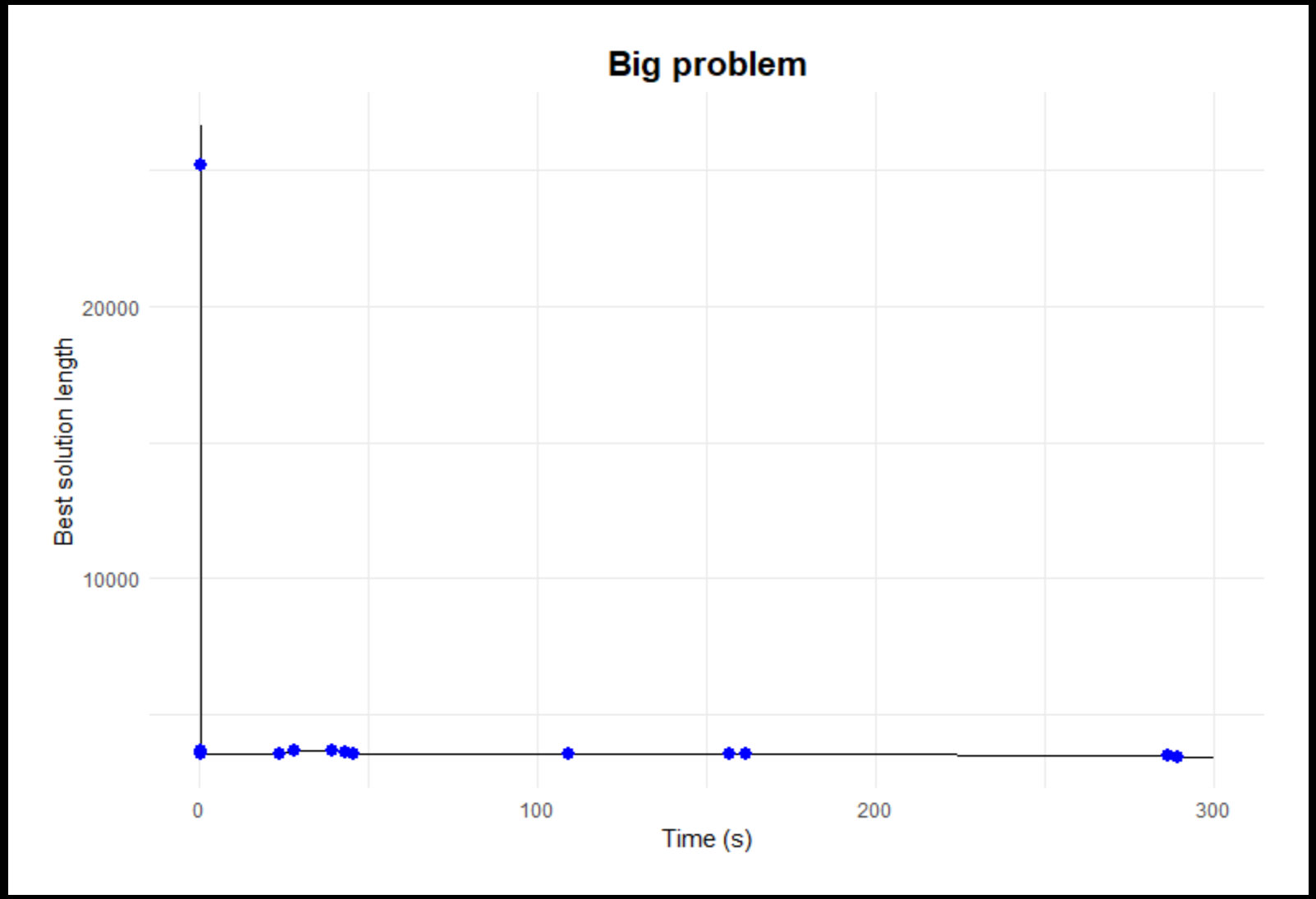

Parameters' impacts on solution
a
b
closeness
cooling_rate
k
max_length_percent_of_cycle
n
probability_of_heuristic
probability_of_shuffle
swap_states_probability
Parameters Values
Percent below best solution

Ostateczne wyniki

NAJLEPSZY WYNIK ZALEŻNIE OD CZASU PRZEPROWADZANIA ALGORYTMU (DLA CZASU CAŁKOWITEGO WYNOSZĄCEGO 5MIN)


Small problem
Best solution length
1700
1650
1600
1550
0
100
200
300
Time (s)


Medium problem
Best solution length
2200
2150
2100
2050
0
100
200
300
Time (s)


Big problem
Best solution length
20000
10000
0
100
200
300
Time (s)

Nasze wyniki w porównaniu z najlepszymi znanymi rozwiązaniami

	ftv35	br17	rbg323	rbg358	rbg443	rbg403	ft70	ftv170	ftv47	ftv70
Our solution length	1556	39	1582	1525	3274	2952	40542	3471	2005	2174
Best known solution length	1473	39	1326	1163	2720	2465	38673	2755	1776	1950

	p43	ry48p	ftv44	ftv64	ftv33	kro124p	ftv38	ftv55	ft53
Our solution length	5639	15173	1750	2058	1440	42010	1616	1788	7988
Best known solution length	5620	14422	1613	1839	1286	36230	1530	1608	6905

Bibliografia

- [1] https://www.undp.org/sites/g/files/zskgke326/files/migration/mk/transformation.png
- [2] https://media.licdn.com/dms/image/D4D22AQHPB41QZ3b0xA/feedshare-shrink_800/0/1684300278082?e=2147483647&v=beta&t=nTMhz3hRYp0bUTt7fKxyJtQtnmNjl-bQxSsBhA58TDU
- [3] https://www.sciencedirect.com/science/article/pii/S0096300309001362?via%3Dihub#fig1

Dziękujemy za uwagę