Znajdowanie optymalnej trasy podczas zakupów spożywczych

Jan Rajczyk, Mateusz Słuszniak, Maksymilian Wojnar

Maj 2022

Spis treści

1 Cel projektu

Celem projektu jest stworzenie modelu pozwalającego wyznaczyć optymalną trasę pomiędzy sklepami posiadającymi zadany asortyment, tak aby skompletować zadaną listę produktów.

2 Model matematyczny

2.1 Struktury danych

Mamy zadane następujące struktury danych:

- P - zbiór wszystkich rodzajów produktów (każdy sklep posiada nieskończoną liczbę egzemplarzy danego produktu),
- $L = \{p_1, p_2, p_3, \dots p_m\} \subset P$ - zbiór rodzajów produktów do kupienia (lista zakupów); produkty są niepodzielne, co sprawia, że dany produkt kupujemy tylko w jednym sklepie,
- N - liczba sklepów,
- x_0, y_0 - współrzędne punktu początkowego,

Własności sklepów:

- $S_i = \{p_j, p_k, p_l, \dots\} \subset P$ - zbiór rodzajów produktów będących w asortymencie i-tego sklepu,
- x_i, y_i - współrzędne i-tego sklepu,
- q_i - "współczynnik kolejki", tj. czas kupowania wszystkich produktów w danym sklepie.

2.2 Postać rozwiązania

- $\pi = [b_1, b_2, \dots, b_k]$ - uporządkowana lista zbiorów produktów do kupienia w kolejnych $k \in \{1, 2, \dots, N\}$ sklepach,
- b_j – zbiór produktów kupionych w j-tym sklepie.

2.3 Postać funkcji celu

Chcemy dokonać minimalizacji czasu spędzonego w drodze oraz w sklepach:

$$f(\pi) = \sum_{i=0}^{k-1} w_{i,i+1} \cdot d_{i,i+1} + \sum_{i=1}^{k} q_k,$$

gdzie

- $k \in \{1, 2, \dots\}$ - indeksy kolejnych sklepów,
- $k = 0$ - punkt początkowy,
- $d_{i,i+1} = \sqrt{(x_{\pi_i} - x_{\pi_{i+1}})^2 + (y_{\pi_i} - y_{\pi_{i+1}})^2}$ - odległości między sklepami,
- $w_{i,i+1}$ - wagi dróg (czynnik skalujący drogę, który oznacza różnicę w czasie podróży drogą szybką oraz wolną),
- q_k - "współczynniki kolejki- liczba określająca jak długo powinny trwać zakupy w danym sklepie.

2.4 Warunki ograniczające

1. $L \subset (\bigcup_{i=0}^{k} b_i)$ - lista zakupów L jest podzbiorem sumy produktów ze wszystkich odwiedzonych k sklepów.
2. Zadajemy punkt początkowy (x_0, y_0).
3. Trasa jest cyklem.

3 Opis algorytmu

3.1 Parametry algorytmu

Dla rozwiązania problemu wykorzystaliśmy algorytm pszczeli z następującymi parametrami:

- ns – liczba rozwiązań początkowych,
- nb – liczba rozwiązań dobrych,
- ne – liczba rozwiązań elitarnych,
- nrb – liczba „zwiadowców" przeszukujących sąsiedztwa rozwiązania dla rozwiązania dobrego,
- nre – liczba „zwiadowców" przeszukujących sąsiedztwa rozwiązania dla rozwiązania elitarnego,
- d – maksymalna odległość definiująca rozmiar sąsiedztwa - w naszym przypadku jest to odległość Levenshteina pomiędzy dwoma ciągami oznaczającymi rozwiązania,
- T_0 – temperatura początkowa w schemacie symulowanego wyżarzania,
- $a \in (0, 1)$ – współczynnik spadku temperatury,
- $Iters_{impr}$ – maksymalna liczba iteracji bez poprawy najlepszego wyniku (warunek stopu),
- $Iters_{max}$ – maksymalna liczba iteracji algorytmu (warunek stopu).

3.2 Rozszerzenia

W rozwiązaniu zastosowaliśmy elementy symulowanego wyżarzania. Podczas przeszukiwania sąsiedztwa, nowa pszczoła może otrzymać lepszy wynik $f(\pi')$, niż dotychchczasowa wartość rozwiązania $f(\pi)$ – w takim wypadku zawsze zachowujemy lepsze rozwiązanie. Jeśli jednak propozycja danej pszczoły będzie miała wyższy koszt $f(\pi')$, wówczas przyjmujemy ją z pewnym prawdopodobieństwem p_t, które jest zależne od temperatury T_n w danej iteracji n:

$$\begin{cases} p_t = e^{(f(\pi) - f(\pi'))/T_n} \\ T_{n+1} = a \cdot T_n \end{cases} \tag{1}$$

3.3 Generowanie rozwiązań początkowych

Do wygenerowania losowych dopuszczalnych rozwiązań, stosujemy prosty Algorytm 1 z heurystyką dotyczącą pierwotnego wyboru kolejności sklepów:

Algorithm 1 Generator rozwiązań początkowych

$\pi \leftarrow$ pusta lista
$i \leftarrow 0$
while zbiór L nie jest pusty **do**
    $item \leftarrow$ losowy element z listy zakupów L
    $shop \leftarrow$ losowy sklep z listy sklepów, który posiada dany produkt
    $b_i \leftarrow$ wszystkie produkty z listy zakupów, które można kupić w sklepie *shop* (zachłanny wybór)
    $\pi \leftarrow \pi + b_i$
    $L \leftarrow L \setminus b_i$
    $i \leftarrow i + 1$
end while
posortuj listę sklepów π kątowo względem punktu początkowego (x_0, y_0)

3.4 Generowanie sąsiednich rozwiązań

Nasz algorytm podaje rozwiązania w formacie listy sklepów oraz produktów, które w nich kupujemy. Zdefiniowaliśmy odległość pomiędzy dwoma rozwiązaniami jako odległość napisów w metryce Levenshteina. Napisem jest uporządkowana lista identyfikatorów sklepów, które odwiedzamy. Algorytm generowania sąsiednich rozwiązań zapewnia, że odległość w zadanej metryce jest mniejsza lub równa parametrowi d, a wygenerowane rozwiązanie jest dopuszczalne.

Algorithm 2 Generator sąsiednich rozwiązań

do
    $solution \leftarrow$ rozwiązanie, dla którego generujemy sąsiadów
    $d \leftarrow$ maksymalna wielkość sąsiedztwa
    while $d > 0$ **do**
        $rand \leftarrow$ losowa liczba ze zbioru $\{0, 1, 2, 3\}$
        if $rand = 0$ **then**
            wstaw nowy sklep w losowe miejsce $solution$
            $d \leftarrow d - 1$
        else if $rand = 1$ **then**
            usuń losowy sklep z $solution$
            $d \leftarrow d - 1$
        else if $rand = 2$ **then**
            zamień losowy sklep z $solution$ na inny
            $d \leftarrow d - 1$
        else if $rand = 3$ oraz $d \geq 2$ **then**
            zamień miejscami dwa losowe sklepy w $solution$
            $d \leftarrow d - 2$
        end if
    end while
while wygenerowane rozwiązanie $solution$ nie jest dopuszczalne

3.5 Pseudokod algorytmu

Algorithm 3 Algorytm pszczeli

$population \leftarrow$ wygeneruj populację początkową ns pszczół Algorytmem 1
$population \leftarrow nb$ najlepszych rozwiązań z $population$
$best \leftarrow$ najlepsze rozwiązanie z $population$
$no\_improvement \leftarrow 0$
for i in $1 \ldots Iters_{max}$ **do**
    $new\_solutions \leftarrow nre$ sąsiednich rozwiązań dla każdej z najlepszych ne pszczół z $population$
    $new\_solutions \leftarrow new\_solutions \cup nbe$ sąsiednich rozwiązań dla każdej z pozostałych najlepszych $nb - ne$ pszczół z $population$
    $new\_solutions \leftarrow new\_solutions \cup (ns - nb)$ rozwiązań wygenerowane Algorytmem 1
    $new\_best \leftarrow$ najlepsze rozwiązanie z $new\_solutions$
    if koszt $new\_best \geq$ koszt $best$ **then**
        $no\_improvement \leftarrow no\_improvement + 1$
    else
        $no\_improvement \leftarrow 0$
        $best \leftarrow new\_best$
    end if
    if $no\_improvement = Iters_{impr}$ **then**
        break
    end if
    $population \leftarrow nb$ najlepszych rozwiązań z $new\_solutions$
end for
return $best$, całkowitą liczbę iteracji oraz numer iteracji, w której otrzymano rozwiązanie $best$

4 Aplikacja

4.1 Format danych wejściowych

Dane wejściowe do algorytmu są przekazywane w formie słownika. Słownik ten zawiera:

- Listę produktów, które mają być zakupione
- Słownik zawierający współrzędne początkowe x_0 oraz y_0
- Lista słowników zawierająca informacje o sklepach. Każdy sklep jest opisany przez jeden słownik. Każdy słownik zawiera id sklepu, współczynnik kolejki danego sklepu, współrzędne x oraz y sklepu, a także listę produktów, które można zakupić w sklepie
- Słownik słowników zawierający mapowanie (id pierwszego sklepu, id drugiego sklepu) $\rightarrow$ czynnik skalujący drogę między sklepami

Funkcja obliczająca rozwiązanie przyjmuje również parametry wymienione w punkcie 3.1.

4.2 Interfejs użytkownika

Interfejs użytkownika jest niezwykle prosty. Główną funkcjonalnością oferowaną przez program, jest funkcja `bees_algorithm` w pliku `bees_algorithm.py`. Służy ona do obliczania właściwego rozwiązania. Oprócz niej, użytkownik może skorzystać z predefiniowanego wrappera funkcji obliczającej wynik. Wrapper znajduje się w pliku `run.py` i przyjmuje na wejściu parametry algorytmu oraz ścieżkę do pliku JSON z danymi instancji (przykładowe uruchomienie: `./run.py --ns=100 --max_iters=1000 --filename="tests/data/normal2d.json"`). Poza tym, użytkownik może chcieć wygenerować własne testy. Mogą w tym pomóc funkcje zaimplementowane w pliku `test_generator.py`.

4.3 Format zwracanych danych

Główna funkcja zwraca liczbę faktycznie wykonanych iteracji, numer iteracji, w której osiągnięto najlepsze rozwiązanie oraz samo rozwiązanie. Rozwiązanie to słownik zawierający dwa pola. Pierwsze z nich to koszt rozwiązania, czyli liczba rzeczywista. Drugie z pól to lista sklepów, jakie należy odwiedzić, z uwzględnieniem kolejności oraz produktów, które kupujemy w kolejnych sklepach.

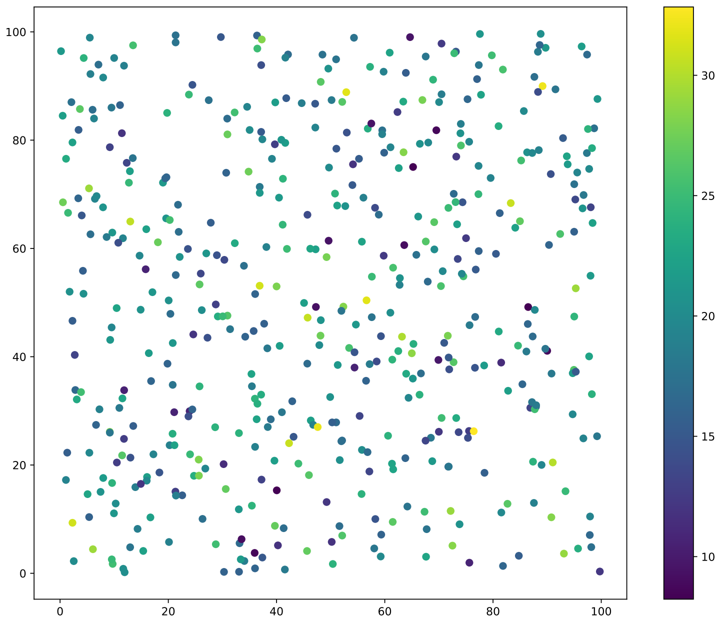
5 Analiza wybranych parametrów

5.1 Dane testowe

Przygotowaliśmy funkcje do generowania dużych losowych testów, aby sprawdzić, jak nasz algorytm sprawdza się w przypadku większych problemów. Poniżej zamieszczamy opis sposobu generowania danych oraz rysunki z przykładowymi danymi.

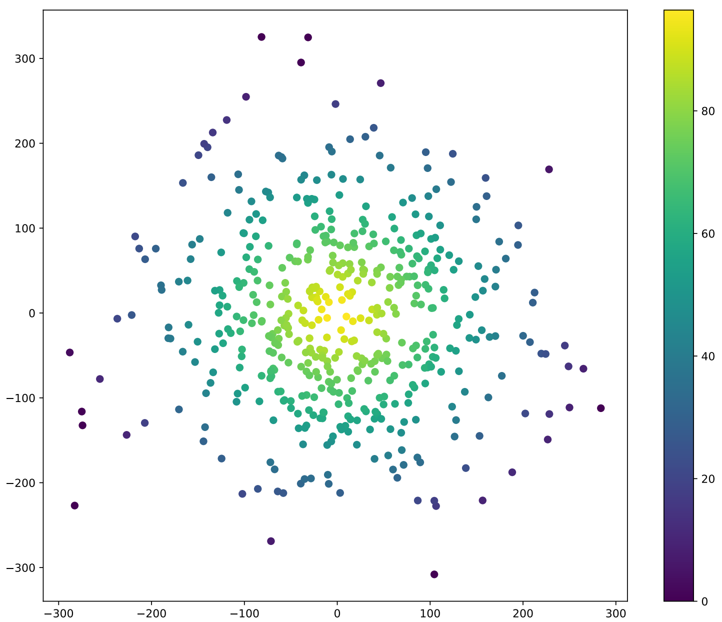
5.1.1 Random

- $x_0, y_0 \sim \mathcal{N}(50, 10)$
- $x_i, y_i \sim \mathcal{U}(0, 100)$
- $q_i \sim \mathcal{N}(20, 5)$
- $w_{i,j} \sim \mathcal{N}(1, 0.05)$
- $|S_i| \in \{1, 2, 3, 4\}$



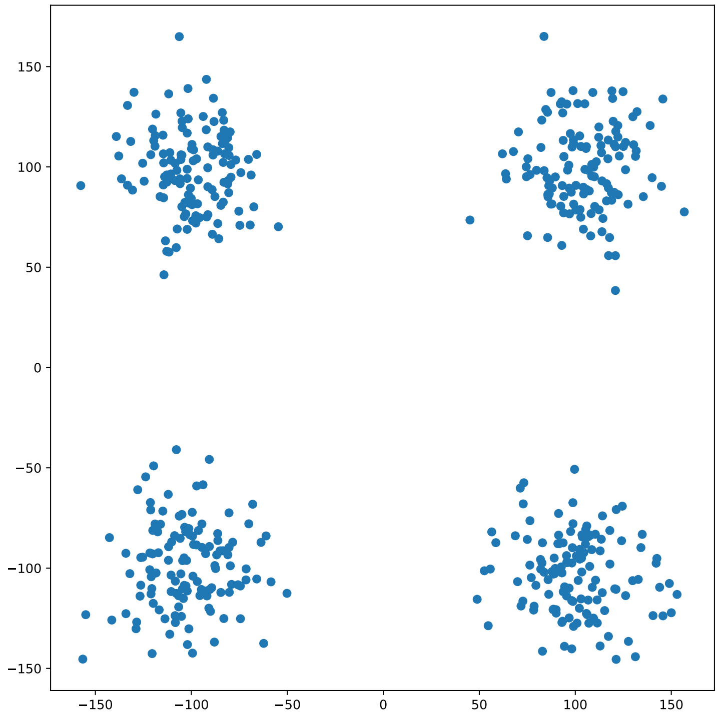
5.1.2 City

- $r = 100$
- $max_q = 100$
- $w_{scale} = 0.5$
- $x_0, y_0 \sim \mathcal{N}(0, r)$
- $x_i, y_i \sim \mathcal{N}(0, r)$
- $q_i = \max\{0, max_q \cdot (1 - \frac{\sqrt{x_i^2 + y_i^2}}{3r})\}$
- $w_{i,j} \sim \mathcal{N}(1, w_{scale})$
- $|S_i| \in \{1, 2, 3, 4\}$



5.1.3 Agglomeration

- $r = 20$
- $x_0 = 0, y_0 = 0$
- $x_i, y_i \sim \mathcal{N}(center, r)$
- $q_i = \text{const} = 0$
- $w_{i,j} \sim \mathcal{N}(1, 0.05)$
- $|S_i| \in \{1, 2, 3, 4\}$



5.2 Metodologia

Spośród zdefiniowanych w sekcji 3.1 parametrów, wybraliśmy trzy: *nre*, *nbe*, *d*. Każdy z wymienionych parametrów został przetestowany na pięciu różnych wartościach $(2, 4, 6, 8, 10)$. Sprawdziliśmy ponadto, jak zachowuje się algorytm z uwzględnieniem elementów symulowanego wyżarzania oraz bez schematu wyżarzania (wtedy zawsze zachowujemy lepsze rozwiązanie). Każdy test został przeprowadzony 5 razy, aby zapewnić większą wiarygodność uzyskanych wyników.

5.3 Opracowanie wyników

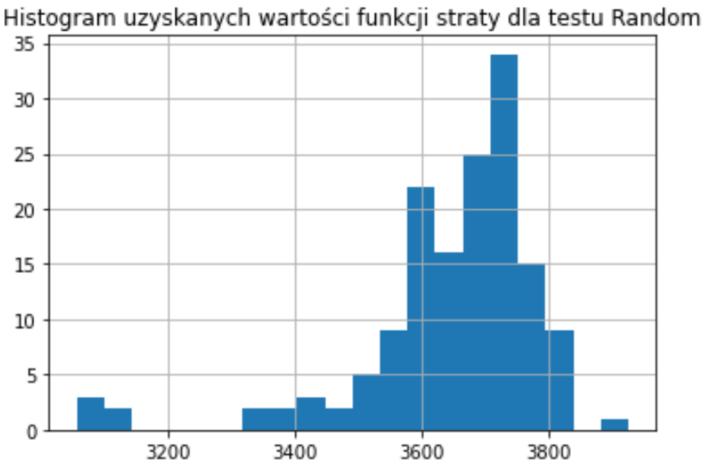
5.3.1 Random

W przypadku testu *random*, który zawierał sklepy rozłożone z rozkładem jednostajnym na kwadracie o wymiarach 100×100 oraz wagach kolejki i wagach krawędzi z rozkładu normalnego o ustalonych parametrach, wyniki były następujące. Zakres osiągniętych wyników ze względu na funkcje kosztu wahały się od 3000 do 3800. Najlepszy rezultat, który udało się uzyskać, to 3087.82 dla parametrów:

- $ns = 50$
- $ne = 20$
- $nb = 30$
- $nre = 5$

- $nrb = 3$
- $d = 2$
- $improve\_iters = 150$
- $max\_iters = 500$
- $temperature = 0$

Ilość pszczół rekrutowanych dla rozwiązań elitarnych w przypadku najlepszego wyniku jest większa od ilości pszczół dla rozwiązań dobrych (5 oraz 3). Może to oznaczać, że algorytm skupił się wówczas na przeszukiwaniu lokalnie najlepszego wyniku, a nie pozwalał sobie na bardziej dynamiczne przejścia pomiędzy rozwiązaniami w poszukiwaniu globalnego optimum. Współczynnik określający maksymalną odległość w metryce Levenshteina pomiędzy dwoma rozwiązaniami wynosił 2, co również pokazuje, że algorytm nie dopuszczał gwałtownych zmian. Temperatura była ustawiona na 0, więc mechanizm symulowanego wyżarzania nie miał w tym przypadku zastosowania. Zatem sądząc po wszystkich wskazaniach, otrzymaliśmy minimum lokalne funkcji, które nie jest globalne. Okazało się, że wpływ mechanizmów, które miały dostarczyć możliwość głębszego przeszukiwania przestrzeni rozwiązań, jest minimalny.



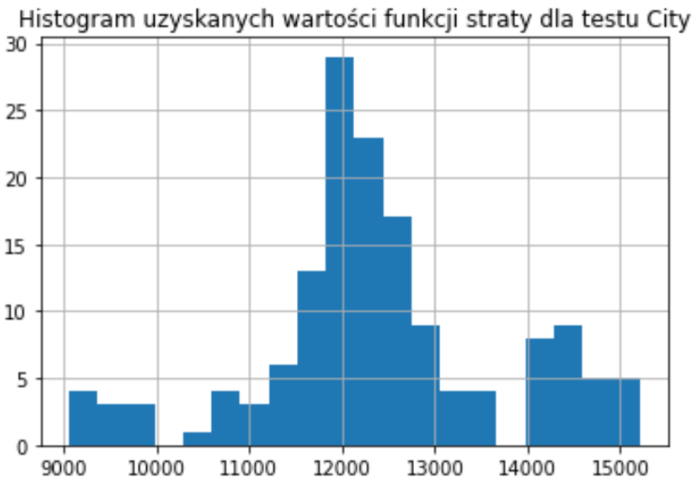
Widać, że znaczna ilość rozwiązań dla tego testu skupiła się w okolicach 3700, więc wyniki w okolicach 3100 są już bardzo dobre, a takie uzyskiwaliśmy niemal wyłącznie dla wskazanych wyżej parametrów.

5.3.2 City

Dla danych testowych *city*, wyniki przyjmowały wartości do 9 do nawet 15 tysięcy, co spowodowane było własnościami wybranej przez nas funkcji kosztu. Najmniejszy koszt równy 9059.78 uzyskaliśmy dla próby posiadającej następujące parametry:

- $ns = 50$
- $ne = 20$
- $nb = 30$
- $nre = 5$
- $nrb = 3$
- $d = 2$
- $improve\_iters = 150$
- $max\_iters = 500$
- $temperature = 0$

Jak możemy łatwo zauważyć, są one identyczne, jak parametry najlepszej próby w testach typu *random*. Widzimy, że tak jak powyżej, najlepszą „startegią” znajdowania wyniku okazała się być strategia skupiania się na poprawie wyników otrzymanych przez elitarne oraz dobrze pszczoły. Odległość Levenshteina równa 2 pozwala z jednej strony na jedynie bardzo delikatne zmiany rozwiązań początkowych, zaś z drugiej dopuszcza wszystkie możliwości manipulacji nimi. Co istotne i warte wspomnienia, parametry dla których udało się uzyskać najlepsze wyniki, są również parametrami, dla których niemalże wszystkie próby nie przekroczyły kosztu równego 10 tysięcy. Jest to szczególnie ciekawe w kontekście opisywanego histogramu – możemy powiedzieć bowiem, że próbki te stanowią pewną wyspę, mocno oddzieloną od pozostałych, gorszych rozwiązań.



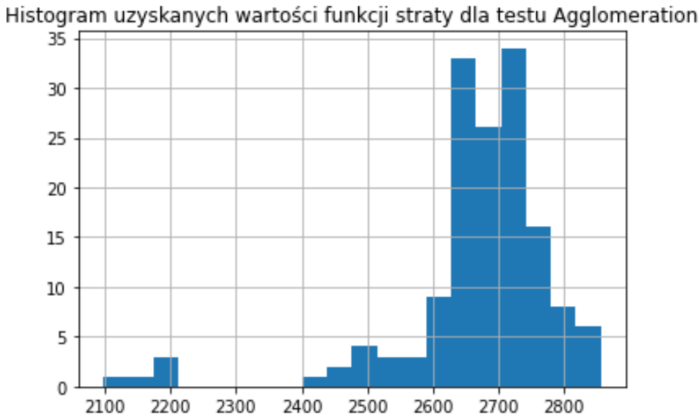
Podczas analizy histogramu, jeszcze jedna rzecz zwróciła naszą uwagę – jest to fakt, że dosyć spore skupisko danych powstało również przy wynikach najgorszych, co nie zdarzyło się dla testu *random*. Patrząc na próbki, które dawały te najgorsze wyniki, można zauważyć, że ich wspólnym mianownikiem zdają się być dwa parametry: po pierwsze dosyć duża odległość pomiędzy rozwiązaniami – przeważnie równa 6, choć zdarzały się i większe – równe chociażby 10, zaś po drugie duża część tych słabych rozwiązań miała również ustawiony parametr temperatury, tak aby realizować symulowane wyżarzanie. Widzimy, więc, że te dwie kwestie zdają się mieć negatywny wpływ na wyniki.

5.3.3 Agglomeration

Ponownie jak w powyższych sekcjach, najlepszy wynik dla testu *agglomeration* – równy 2097.28 – uzyskaliśmy dla zadanych parametrów:

- $ns = 50$
- $ne = 20$
- $nb = 30$
- $nre = 5$
- $nrb = 3$
- $d = 2$
- $improve\_iters = 150$
- $max\_iters = 500$
- $temperature = 0$

Widzimy więc, że taktyka operowania na znalezionych już rozwiązaniach przyniosła ponownie najbardziej obfite plony. W powyższych punktach opisaliśmy już, co może na takowy stan rzeczy wpływać. Warto jeszcze przyjrzeć się histogramowi, aby zobaczyć, czy w rozkładzie wyników znalazło się coś ciekawego:



To co może przykuwać naszą uwagę, to fakt bardzo dużego zgromadzenia wyników wokół wartości najgorszych z przedziału 2600-2750. W wielu z tych rozwiązań widzimy spore odległości w metryce Levenshteina (przeważnie równe 6-10), a także obecność symulowanego wyżarzania.

5.4 Wnioski ogólne

Jak pokazały eksperymenty, najlepsze wyniki uzyskujemy dla parametrów:

- $ns = 50$
- $ne = 20$
- $nb = 30$
- $nre = 5$
- $nrb = 3$
- $d = 2$
- $improve\_iters = 150$
- $max\_iters = 500$
- $temperature = 0$

niezależnie od testowanych danych. Płynie stąd prosty wniosek o tym, że najlepiej sprawdzają się w naszym problemie delikatne poprawki – bardzo kosmetyczna zmiana kolejności sklepów, bądź dodanie czy usunięcie jednego z nich, a także to, że bardzo duża ilość pszczółek jest zaprzęgana do poprawy istniejących już rozwiązań i tylko mały ich procent prowadzi poszukiwania globalnego optimum.

6 Bibliografia

- `http://beesalgorithmsite.altervista.org/BeesAlgorithm.htm`
- Materiały z kursu *Badania Operacyjne*
- `https://en.wikipedia.org/wiki/Bees_algorithm`