Zaawansowane metody optymalizacji

Alicja Augustyniak

14.06.2025

Spis treści

1 Wybrane problemy

1.1 Problem Flow Shop $F2, h_{T2} \mid$ no wait $\mid L_{\max}$

Flow Shop to jedna z klasycznych klas problemów szeregowania zadań, w której każde zadanie przetwarzane jest na wszystkich maszynach w tej samej, z góry ustalonej kolejności. W najogólniejszym przypadku, Flow Shop składa się z m maszyn oraz n zadań, przy czym każde zadanie J_j składa się z m kolejnych operacji, z których każda wykonywana jest na innej maszynie w ściśle określonej kolejności.

W wybranym przypadku $F2$, proces przebiega na dwóch maszynach M_1 i M_2, a każde zadanie J_j składa się z dwóch operacji, na pierwszej i drugiej maszynie.

Każde zadanie ma przypisany termin wykonania d_j. Nie istnieją przerwy pomiędzy zadaniami na maszynach (warunek no wait) co oznacza, że operacja na maszynie M_2 musi rozpocząć się natychmiast, po zakończeniu operacji na maszynie M_2.

Dodatkowo, parametr h_{T2} uwzględnione są również okresy niedostępności na maszynie M_2. Po zakończeniu zadania, wyznaczany jest czas zakończenia zadania C_j a następnie opóźnienie:

$$L_j \;=\; C_j - d_j$$

Celem jest minimalizacja maksymalnego opóźnienia:

$$L_{\max} = \max_{j=1,\ldots,n} L_j$$

1.1.1 Złożoność obliczeniowa problemu

Złożoność problemów typu Flow Shop rośnie wykładniczo i należy do klasy problemów NP-trudnych.

1.2 Problem Open Shop $O2$, $h_{2k} \mid LE \mid L_{\max}$

Problem Open Shop stanowi jedną z podstawowych klas zagadnień szeregowania zadań na maszynach dedykowanych. W jego ogólnej postaci rozpatrujemy układ składający się z m maszyn oraz n zadań, w którym każde zadanie J_j składa się z m operacji, po jednej na każdą maszynę. Kluczową cechą Open Shop jest to, że kolejność wykonywania tych operacji nie jest z góry ustalona: każda operacja może być wykonana w dowolnym momencie, o ile zarówno maszyna, jak i samo zadanie są dostępne, a jedynym warunkiem jest brak nakładania się operacji tej samej maszyny ani dwóch operacji jednego zadania.

W rozważanym specjalnym przypadku $O2$ znajdują się dokładnie dwie maszyny, oznaczane M_1 i M_2, oraz n zadań $J_1, J_2, \ldots, J_n$. Zadanie J_j wymaga wykonania najpierw operacji na maszynie M_1 o czasie p_{j1}, a następnie operacji na maszynie M_2 o czasie p_{j2}, lub odwrotnie, kolejność wybieramy dowolnie, celem zminimalizowania wskaźnika jakości.

Dodatkowo uwzględniony jest efekt uczenia (LE, ang. *learning effect*), który modeluje przyspieszenie (lub spowolnienie) wykonania kolejnych operacji na tej samej maszynie w zależności od już zrealizowanej kolejności zadań.

Każdemu zadaniu J_j przypisany jest termin d_j. Po zakończeniu obu jego operacji wyznaczany czas zakończenia C_j oraz opóźnienie:

$$L_j \;=\; C_j - d_j$$

Celem jest minimalizacja maksymalnego opóźnienia:

$$L_{\max} \;=\; \max_{j=1,\ldots,n} L_j.$$

1.2.1 Złożoność obliczeniowa problemu

Problem Open Shop z dwoma maszynami i dodatkowymi warunkami h_{2k} (okresy niedostępności na maszynie 2), efektem uczenia (LE) oraz celem minimalizacji opóźnień $L_{\max}$ należy do klasy NP-trudnych.

- **Klasa NP-trudności**
  Nie istnieje znany algorytm wielomianowy rozwiązujący dokładnie ten problem dla dowolnej liczby zadań n.
- **Eksplozja kombinatoryczna**
  Przestrzeń wszystkich możliwych harmonogramów (permutacji n zadań na dwóch maszynach) ma rozmiar rzędu $n!$. Już dla $n \approx 15$ liczba permutacji jest astronomiczna, co uniemożliwia pełne przeszukanie.
- **Metody rozwiązania**

– *Algorytmy dokładne* (branch&bound, MIP) działają w czasie wykładniczym, praktycznie tylko dla małych n.
– *Metaheurystyki* (Tabu Search, Simulated Annealing, Algorytmy genetyczne) oferują kompromis: brak gwarancji optimalności, ale dobre wyniki w czasie wielomianowym.

W praktyce dla $n > 15$–20 stosuje się metaheurystyki, by uzyskać *dobry* wynik $L_{\max}$ w akceptowalnym czasie.

Tabela 1: Porównanie Open Shop i Flow Shop

Cecha	**Open Shop**	**Flow Shop**
Kolejność operacji	Dowolna	Stała (np. M1 → M2)
Elastyczność	Wysoka	Niska
Złożoność	NP-trudna	NP-trudna
Typowe zastosowania	Naprawy, usługi	Produkcja masowa

2 Programowanie matematyczne

2.1 Flow Shop: $F2, h_{T2} \mid$ no wait $\mid L_{\max}$

2.1.1 Definicja problemu

Problem Flow Shop z:

- 2 maszynami (Maszyna 1. i Maszyna 2.),
- okresami niedostępności tylko na Maszynie 2. (h_{2T}),
- kryterium minimalizacji maksymalnego opóźnienia ($L_{\max}$).

2.1.2 Oznaczenia

- n – liczba zadań,
- $i \in \{1, 2\}$ – indeks maszyny,
- $j \in \{1, 2, \ldots, n\}$ – indeks zadania,
- $l \in \{1, 2, \ldots, n\}$ – pozycja w kolejności wykonywania zadań,
- h_{2k} – długość k-tego okresu niedostępności na Maszynie 2,
- s_{k2} – moment rozpoczęcia k-tego okresu niedostępności na Maszynie 2,
- t_{li} – moment rozpoczęcia zadania na pozycji l na maszynie i,
- p_{ji} – pierwotny czas wykonania zadania j na maszynie i,
- $p^i_{jl} = p_{ji} \cdot l^a$ – rzeczywisty czas wykonania zadania j na pozycji l na maszynie i,
- $z^i_{jl} \in \{0, 1\}$ – zmienna binarna, oznaczająca czy zadanie j jest na pozycji l na maszynie i,
- d_j – termin zakończenia zadania j,
- C_j – czas zakończenia zadania j,
- $L_j = C_j - d_j$ – opóźnienie zadania j,
- $L_{\max} = \max_j\{L_j, 0\}$ – maksymalne opóźnienie (funkcja celu).

2.1.3 Ograniczenia

Minimalizujemy L_{max}, przy ograniczeniach:

1. $\sum_{i=1}^{n} z_{ij} = 1 \qquad \forall j = 1, \ldots, n \qquad (1)$
2. $\sum_{j=1}^{n} z_{ij} = 1 \qquad \forall i = 1, \ldots, n \qquad (2)$
3. $C_j = t_{L_j} + \sum_{i=1}^{n} z_{ij}(p_i + q_i) \qquad \forall j = 1, \ldots, n \qquad (3)$
4. $L_j = C_j - d_j \qquad \forall j = 1, \ldots, n \qquad (4)$
5. $L_{\max} \geq L_j \qquad \forall j = 1, \ldots, n \qquad (5)$
6. $t_{L_j} \geq t_{L_i} + \sum_{k=1}^{n} (p_k + q_k) z_{ik} z_{jk} \qquad \forall i \neq j,\ i, j = 1, \ldots, n \qquad (6)$
7. $t_{L_l} + \sum_{j=1}^{n} (p_j + q_j) z_{jl} z_{jk} > t_{L_k} \qquad \forall k, l = 1, \ldots, n,\ k \neq l \qquad (7)$
8. $t_i \geq 0 \qquad (8)$
9. $t_l + \sum_{j'=1}^{n} p_{j'} z_{j'l} \leq S_k \quad \text{lub} \quad t_l \geq S_k + h_k \qquad \forall l = 1, \ldots, n,\ k = 1, \ldots, K \qquad (9)$
10. $S_k - (s_k + h_k) \leq \tau \qquad \forall k = 1, \ldots, K - 1 \qquad (10)$

2.1.4 Funkcja celu

Minimalizacja maksymalnego opóźnienia:

$$\min L_{\max}$$

2.2 Open Shop: $O2, h_{2k} \mid LE \mid L_{\max}$

2.2.1 Definicja problemu

Problem Open Shop z:

- 2 maszynami (Maszyna 1. i Maszyna 2.),
- okresami niedostępności tylko na Maszynie 2. (h_{2k}),
- efektem uczenia (LE),
- kryterium minimalizacji maksymalnego opóźnienia ($L_{\max}$).

2.2.2 Oznaczenia

$$n - \text{liczba zadań} \tag{11}$$
$$i \in \{1, 2\} - \text{indeks maszyny} \tag{12}$$
$$j \in \{1, 2, \ldots, n\} - \text{indeks zadania} \tag{13}$$
$$l \in \{1, 2, \ldots, n\} - \text{pozycja w kolejności wykonywania} \tag{14}$$
$$K_2 - \text{liczba okresów niedostępności na Maszynie 2} \tag{15}$$
$$h_{2k} - \text{długość } k\text{-tego okresu niedostępności} \tag{16}$$
$$s_{k2} - \text{moment rozpoczęcia } k\text{-tego okresu niedostępności} \tag{17}$$
$$t_{li} - \text{moment rozpoczęcia zadania na pozycji } l \text{ na maszynie } i \tag{18}$$
$$p_{ji} - \text{pierwotny czas wykonania zadania } j \text{ na maszynie } i \tag{19}$$
$$a \leq 0 - \text{współczynnik uczenia} \tag{20}$$
$$p_{jli} = p_{ji} \cdot l^a - \text{rzeczywisty czas wykonania} \tag{21}$$
$$z_{jli} \in \{0, 1\} - \text{zmienna binarna} \tag{22}$$
$$d_j - \text{termin zakończenia zadania } j \tag{23}$$
$$C_j - \text{czas zakończenia zadania } j \tag{24}$$
$$L_j = C_j - d_j - \text{opóźnienie zadania } j \tag{25}$$
$$L_{\max} = \max\{L_j\} - \text{maksymalne opóźnienie} \tag{26}$$

2.2.3 Ograniczenia

1. Każde zadanie wykonane dokładnie raz na każdej maszynie:

$$\sum_{l=1}^{n} z_{jli} = 1 \quad \forall j = 1, \ldots, n; \quad i = 1, 2$$

2. Każda pozycja zajęta przez co najwyżej jedno zadanie:

$$\sum_{j=1}^{n} z_{jli} = 1 \quad \forall l = 1, \ldots, n; \quad i = 1, 2$$

3. Zmienne binarne:

$$z_{jli} \in \{0, 1\} \quad \forall j, l, i$$

4. Maszyna dostępna od $t = 0$:

$$t_{1i} \geq 0 \quad \forall i = 1, 2$$

5. Efekt uczenia:

$$a \leq 0$$

6. Liczba okresów:

$$K_2 \geq 0$$

7. Długość okresów:

$$h_{2k} > 0 \quad \forall k = 1, \ldots, K_2$$

8. Brak kolizji zadań z okresami niedostępności:

$$t_{l2} + \sum_{j=1}^{n} p_{jl2} z_{jl2} \leq s_{k2} \quad \vee \quad t_{l2} \geq s_{k2} + h_{2k} \quad \forall l, k$$

9. Kolejność zadań na maszynach:

$$t_{l+1,i} \geq t_{li} + \sum_{j=1}^{n} p_{jli} z_{jli} \quad \forall l, i$$

10. Kolejność okresów niedostępności:

$$s_{k+1,2} \geq s_{k2} + h_{2k} \quad \forall k$$

11. Czas zakończenia zadań:

$$C_j \geq t_{li} + \sum_{j'=1}^{n} p_{j'li} z_{j'li} \quad \forall j, l, i \text{ gdzie } z_{jli} = 1$$

12. Maksymalne opóźnienie:

$$L_{\max} = \max\{C_j - d_j\}$$

2.2.4 Funkcja celu

Minimalizacja maksymalnego opóźnienia:

$$\min L_{\max}$$

3 Zaimplementowane algorytmy dla problemu $F2, h_{T2} \mid$ **no-wait** $\mid L_{\max}$

3.1 Algorytm heurystyczny

W celu sprostania zadanemu problemowi podjęto się implementacji algorytmu symulowanego wyżarzania (Simulated Annealing, SA). Zastosowano algorytm w czterech różnych wariantach: wersja podstawowa z losową permutacją startową; wersja z modyfikacją chłodzenia - okresowym podgrzewaniem i detekcją stagnacji; wersja z deterministyczną permutacją startową (wybrano Earliest Due Date); wersja z deterministyczną permutacją startową (EDD) oraz modyfikacją chłodzenia i detekcją stagnacji.

3.1.1 Inicjalizacja

Algorytm rozpoczyna od wygenerowania losowej permutacji zadań. Dla tej początkowej kolejności obliczana jest wartość funkcji celu $L_{\max}$. Następnie ustawiana jest początkowa temperatura $T_0 = 1000$, która kontroluje prawdopodobieństwo akceptacji gorszych rozwiązań w trakcie procesu optymalizacji.

3.1.2 Główna pętla optymalizacyjna

W każdej iteracji algorytm:

1. Generuje nowe rozwiązanie sąsiednie poprzez zamianę miejscami dwóch losowo wybranych zadań w aktualnej permutacji.
2. Oblicza wartość $L_{\max}$ dla nowego rozwiązania.
3. Decyduje o akceptacji nowego rozwiązania na podstawie:
   - natychmiastowej akceptacji, jeśli nowe rozwiązanie jest lepsze ($L_{\max}^{\text{nowe}} < L_{\max}^{\text{aktualne}}$);
   - akceptacji z prawdopodobieństwem $P = \exp\left(\frac{L_{\max}^{\text{aktualne}} - L_{\max}^{\text{nowe}}}{T}\right)$ w przeciwnym przypadku.
4. Obniża temperaturę zgodnie ze wzorem: $T \leftarrow \alpha T$, gdzie $\alpha = 0.998$

3.1.3 Zakończenie

Po wykonaniu zdefiniowanej liczby iteracji (domyślnie 100 000) algorytm zwraca najlepsze znalezione rozwiązanie.

3.1.4 Obliczanie funkcji celu $L_{\max}$

Funkcja celu $L_{\max}$ obliczana jest w następujący sposób:
- dla każdego zadania w aktualnej kolejności wyznaczany jest czas rozpoczęcia na pierwszej maszynie (S_j^1) i drugiej maszynie (S_j^2);
- uwzględniane są ograniczenia technologiczne;
- zadanie może rozpocząć się na drugiej maszynie dopiero po zakończeniu na pierwszej;
- sprawdzane są kolizje z okresami niedostępności drugiej maszyny;
- dla każdego zadania obliczane jest opóźnienie $L_j = C_j - d_j$;
- $L_{\max}$ to maksymalna wartość spośród wszystkich L_j (lub zero, jeśli żadne zadanie nie jest spóźnione).

3.2 Okresowe podgrzewanie (reheating)

W celu zapobieganiu utknięciu w lokalnych minimach, wprowadzono mechanizm okresowego resetowania temperatury. Co 20% całkowitej liczby iteracji temperatura jest przywracana do wartości początkowej T_0. Dodatkowy reset następuje, gdy przez połowę tego okresu nie nastąpiła żadna poprawa rozwiązania. Mechanizm ten pozwala na okresowe zwiększenie prawdopodobieństwa akceptacji gorszych rozwiązań, co ułatwia ucieczkę z lokalnych minimów

3.2.1 Deterministyczna permutacja startowa (EDD)

W tej modyfikacji przed rozpoczęciem optymalizacji zadania są sortowane według rosnących terminów wykonania (Earliest Due Date). Daje to punkt startowy bliższy optymalnemu rozwiązaniu i jest szczególnie skuteczne dla instancji, gdzie kolejność EDD jest zbliżona do optymalnej.

3.2.2 Wnioski

Wersja podstawowa jest najprostsza w implementacji i wystarczająca dla małych problemów, ale może utknąć w lokalnych minimach dla bardziej złożonych instancji.
Wersja z podgrzewaniem wykazuje lepszą skuteczność dla dużych problemów (np. $N \geq 15$), dzięki możliwości ucieczki z lokalnych minimów.
Wersja z EDD może przyspieszyć zbieżność, ale jej skuteczność zależy od struktury terminów wykonania zadań, w niektórych przypadkach może ograniczać eksplorację przestrzeni rozwiązań.

3.2.3 Złożoność obliczeniowa

Dla I iteracji algorytmu, całkowita złożoność wynosi:

$$\mathcal{O}(I \cdot n \cdot k) \tag{27}$$

- Liczba iteracji I
- Liczba zadań n
- Liczba przerw k

3.2.4 Czynniki wpływające na wydajność

Tabela 2: Wpływ parametrów na złożoność

Parametr	Wpływ na złożoność
Liczba zadań (n)	Liniowy wzrost $\mathcal{O}(n)$
Liczba przerw (k)	Liniowy wzrost $\mathcal{O}(k)$
Odstęp między przerwami (τ)	Odwrotnie proporcjonalny do k

3.2.5 Porównanie zaimplementowanych algorytmów

Opracowane warianty przetestowano na podanych poniżej parametrach, a uzyskane wyniki umieszczono w tabelach. Wninoski i obserwacje podano pod każdą tabelą.

Tabela 3: Zakresy i wartości domyślne parametrów testów

Parametr	**Zakres testów**	**Wartość domyślna**
Liczba zadań N	5, 10, 15, 20, 25	10
Długość okresu h	1, 3, 5, 7, 9	3
Odstęp między przerwami τ	10, 20, 30, 40, 50	20
Maks. długość zadania (range)	3, 5, 10, 15, 20	5
Maks. iteracji (maxIter)	–	100 000
Temperatura początkowa (temp_0)	–	1000
Czynnik chłodzenia (α)	–	0,998
Liczba powtórzeń (repeats)	–	5

Uwaga: Algorytmy testowano również dla 10 000 i 54 000 iteracji oraz współczynników chłodzenia 0,995 i 0,98, jednak najlepsze wyniki uzyskano dla wartości zapisanych w tabeli.

Tabela 4: Wartość Lmax w zależności od N

N	SA	SA_EDD_reheating	SA_reheating	SA_EDD
5	**7,6**	7,8	**7,2**	**7,6**
10	**10,4**	10,8	14	**10**
15	18,6	**14,4**	**14**	14,8
20	17,2	19,4	**16**	**15,2**
25	21,6	30,4	**20**	**19,4**

Po przeprowadzonych obliczeniach i analizie wyników dla każego z wariantów można zaobserwować kluczowe prawidłowości. SA dobrze radzi sobie z małymi rozmiarami N, natomiast dla większych instancji bardzo skutczne okazuje się zastosowanie modyfikacji chłodzenia lub heurystyki EDD. Warto jednak zauważyć, że algorytm z obiema modyfikacjami daje najgorsze wyniki dla większych instancji (większe od 15). Jeśli startowy układ EDD jest daleko od globalnego optimum (co przy dużym N i wielu okresach przerw jest bardzo prawdopodobne), SA-EDD-reheating nie ma wystarczająco silnego bodźca, by opuścić ten region. Każda propozycja daleko odbiegająca od EDD jest szybko odrzucana lub przyjmowana tylko rzadko i reset temperatury nie zmienia faktu, że większość iteracji krąży wokół lokalnych minimów wynikających z EDD. Podgrzewanie cofa tylko proces ochładzania i dopuszcza większą liczbę gorszych przetasowań, ale nadal operuje w obrębie sąsiedztwa pozycji wyjściowej. Jeśli wartość parametru N liczba zadań jest duży, lepsze okazuje się rozpoczynać od losowej (lub wielokrotnie różnych) permutacji w połączeniu z mechanizmem pogrzewania, zamiast „uprzednio" narzucać harmonogram EDD.

Tabela 5: Wartość Lmax w zależności od h

h	SA	SA_EDD_reheating	SA_reheating	SA_EDD
1	11,2	**10,2**	**9,6**	12,2
3	**10,6**	12	11,8	**10,4**
5	**11,8**	15,8	**13,8**	16
7	19	23,2	**17,4**	**15,6**
9	22,8	23	**18,8**	**19,4**

Dla małych wartości h użycie SA bez modyfikacji w zupełności wystarcza. Dla większych wartości h widoczne są dużo lepsze wyniki dla algorytmów wraz z modyfikacjami, najlepiej radzi sobie algorytm z podgrzewaniem. Gdy okresy niedostępności są długie, w harmonogramie łatwo powstają przedziały czasu, w których maszyny stoją bezczynnie, a zadania gromadzą się lub są przesuwane tak, że ostateczne opóźnienia znacznie rosną. W klasycznym SA, w miarę jak temperatura opada, prawdopodobieństwo zaakceptowania ruchu pogarszającego jakość rozwiązania maleje, więc algorytm szybko utrzyma się na jednym z takich lokalnych minimów i już go nie opuści. Mechanizm cyklicznego podnoszenia temperatury z powrotem do wartości początkowej pozwala w takich momentach zwiększyć losowość akceptowanych ruchów. Gdy temperatura rośnie, ponownie dopuszczamy znaczny odsetek przetasowań nawet o wyższej wartości $L_{\max}$. Dzięki temu SA może zaakceptować ruchy, które z pozoru prowadzą do gorszego wyniku, ale w rzeczywistości przenoszą go poza dotychczasowe „zagłębienie" w krajobrazie funkcji celu.

Tabela 6: Wartość Lmax w zależności od τ

τ	SA	SA_EDD_reheating	SA_reheating	SA_EDD
10	**15**	**15,4**	17,6	19,4
20	**12,4**	15	14,6	**13,4**
30	**8,4**	9,2	**9**	12,4
40	**9,4**	**8**	10,6	12,2
50	**7,6**	10,6	**7**	**7,6**

Przy testowanych wartościach czyste SA (losowa permutacja oraz klasyczne chłodzenie) jest w zupełności wystarczające, daje niskie wartości $L_{\max}$ i pracuje wyjątkowo szybko. Być może w aplikacjach z bardzo częstymi przerwami można rozważyć modyfikacje np. podgrzewanie, by pomóc SA wydostać się z głębokich minimów.

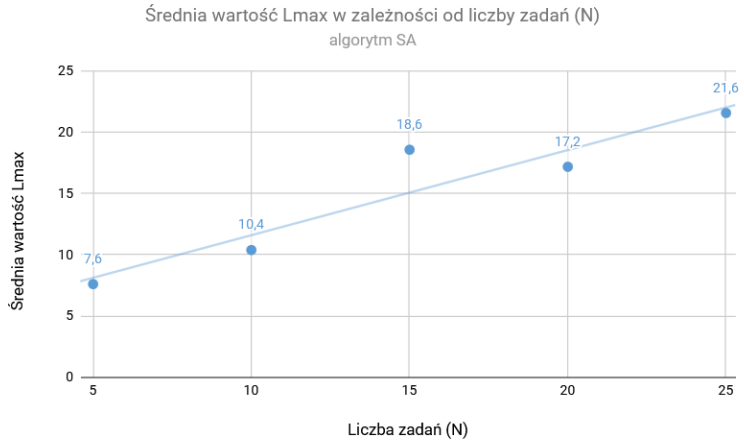
Tabela 7: Wartość Lmax w zależności od range

range	SA	SA_EDD_reheating	SA_reheating	SA_EDD
3	8,6	**7,6**	**6**	8
5	15,2	**10,2**	11,2	**9,8**
10	31,4	**25,2**	**26,4**	27,6
15	46	**34**	**34,2**	35
20	305,8	**229,4**	**304,4**	364,4

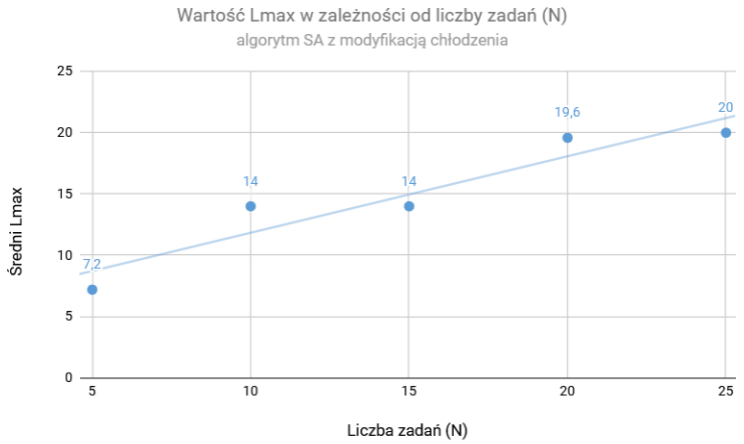
EDD pozwala na uszeregowanie zadań według rosnącego terminu wykonania, co minimalizuje największe przekroczenia na starcie, szczególnie gdy długości zadań są bardzo zróżnicowane (duże wartości range). Dzięki temu algorytm startuje bliżej optymalnego rozwiązania i nie marnuje cennych pierwszych iteracji na „naprawianie" rażących opóźnień. Okresowe przywracanie pierwotnej temperatury pomaga SA wyjść z lokalnych minimów, które stają się głębsze, gdy zadania są dłuższe i częściej blokują maszynę. Wariant SA-reheating (bez EDD) radzi sobie lepiej od czystego SA tylko przy średnich wartościach range (np. 10-15), ale przy dużym range=20 nie nadąża za SA-EDD-reheating i błędnie „zapada" w lokalnym minimum niewystarczająco dobrym (304,4 vs. 229,4). Sama permutacja EDD jest dobra na start, ale brak możliwości ucieczki z lokalnych minimów oznacza, że po wstępnym ułożeniu zadań algorytm szybko zatrzymuje się wokół pierwszego napotkanego dobrego, lecz suboptymalnego harmonogramu.

3.2.6 Analiza przeprowadzonych obliczeń

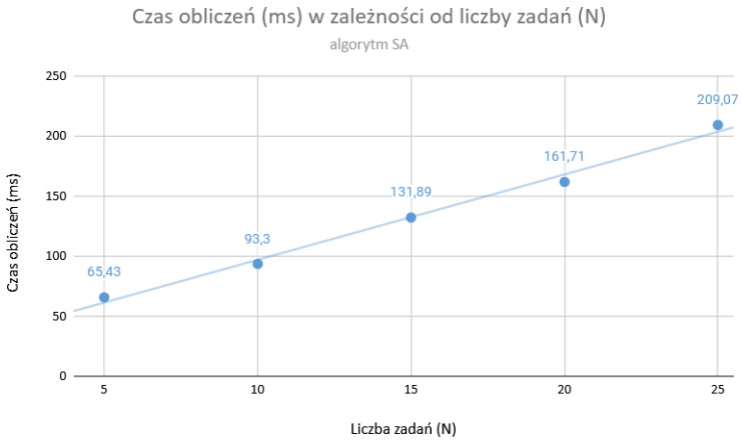
Do szczegółowej analizy wybrano algorytm SA oraz algorytm SA z mechanizmem podgrzewania i zapobiegania stagnacji.



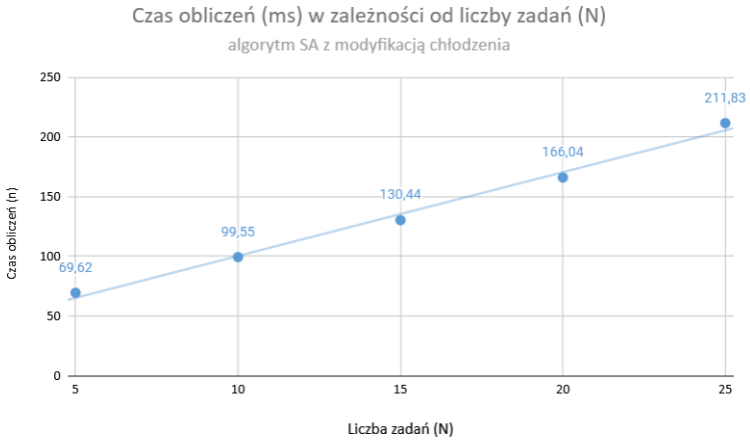
Rysunek 1: Średnia wartość $L_{\max}$ w zależności od liczby zadań N



Rysunek 2: Średnia wartość $L_{\max}$ w zależności od liczby zadań N (modyfikacja chłodzenia)

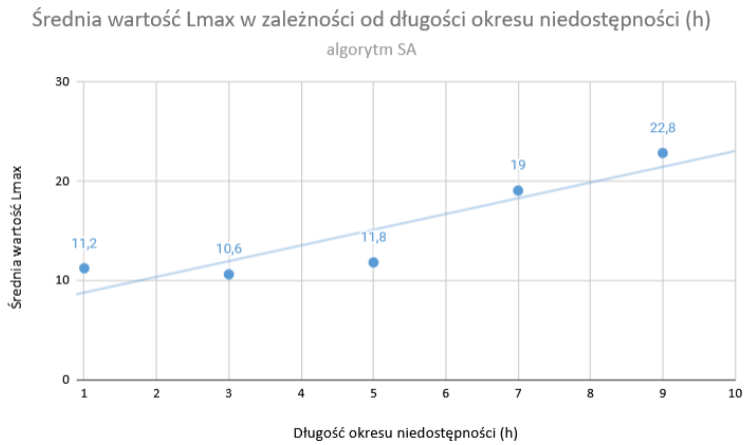


Rysunek 3: Czas obliczeń (ms) w zależności od liczby zadań N

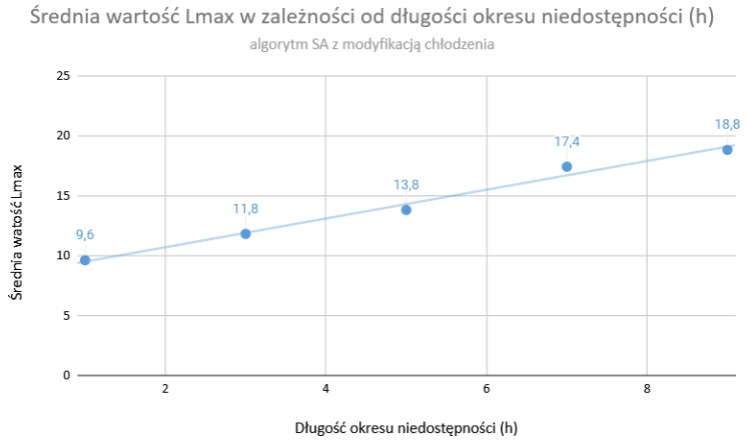


Rysunek 4: Czas obliczeń (ms) w zależności od liczby zadań N (modyfikacja chłodzenia)

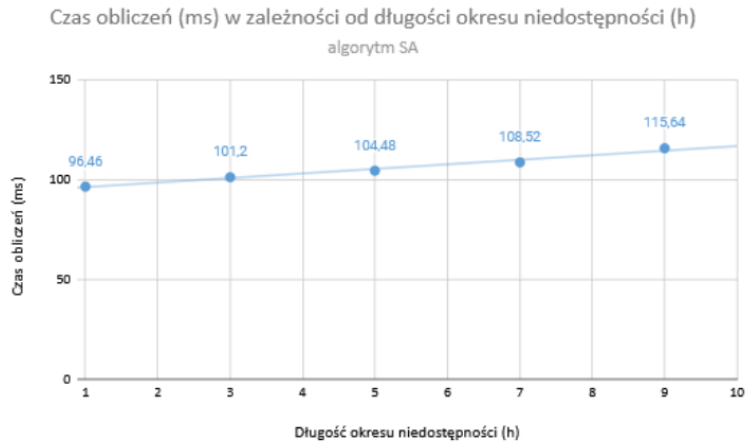
Średnie $L_{\max}$ w obu przypadkach rośnie wraz ze wzrostem liczby zadań (Rys. 1 oraz Rys. 2). Dla małych instancji (N=5-10) różnice między wersjami są minimalne, co sugeruje, że prostsza wersja SA jest wystarczająca dla małych problemów. Przy N>15 wersja z modyfikacją chłodzenia wykazuje lepszą skalowalność, utrzymując niższe wartości $L_{\max}$. Wynika to z mechanizmu podgrzewania, który pomaga uciec z lokalnych minimów przy większej przestrzeni przeszukiwań. Czas obliczeń w obu przypadkach rośnie niemal liniowo wraz ze wzrostem liczby zadań (Rys. 3 oraz Rys. 4). W przypadku wersji z modyfikacją chłodzenia czasy obliczeń są minimalnie większe, co wynika z dodatkowych obliczeń związanych z detekcją stagnacji i podgrzewaniem. Heurystyka na tym etapie nie ma problemu z realizacją zadanego problemu w błyskawicznym czasie, a otrzymywane rezultaty są zgodne ze spodziewanymi.



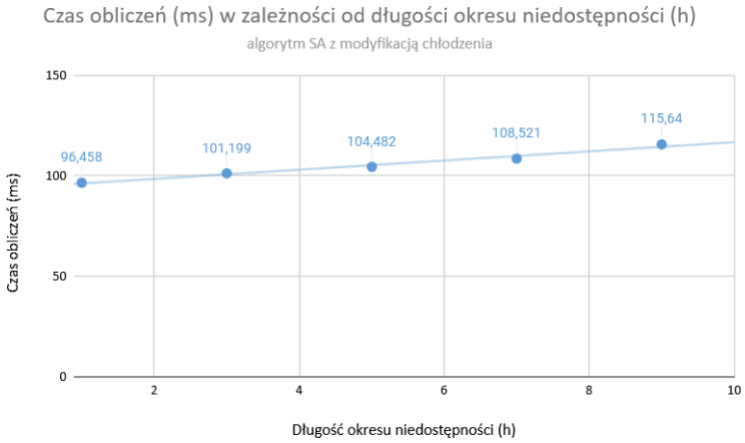
Rysunek 5: Średnia wartość $L_{\max}$ w zależności od długości okresu niedostępności h



Rysunek 6: Średnia wartość $L_{\max}$ w zależności od długości okresu niedostępności h (modyfikacja chłodzenia)

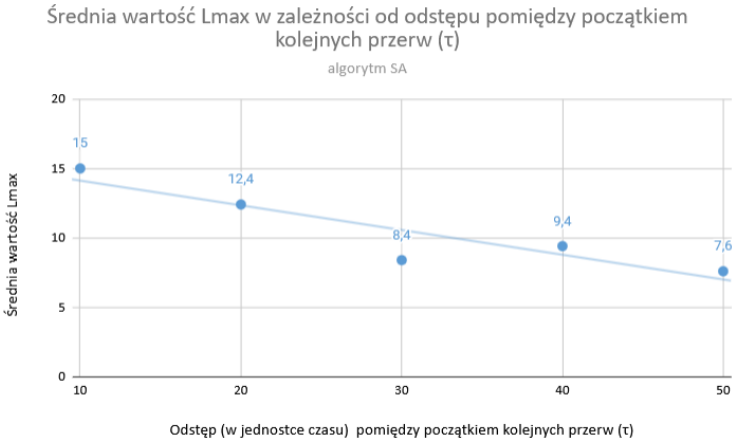


Rysunek 7: Czas obliczeń (ms) w zależności od długości okresu niedostępności h

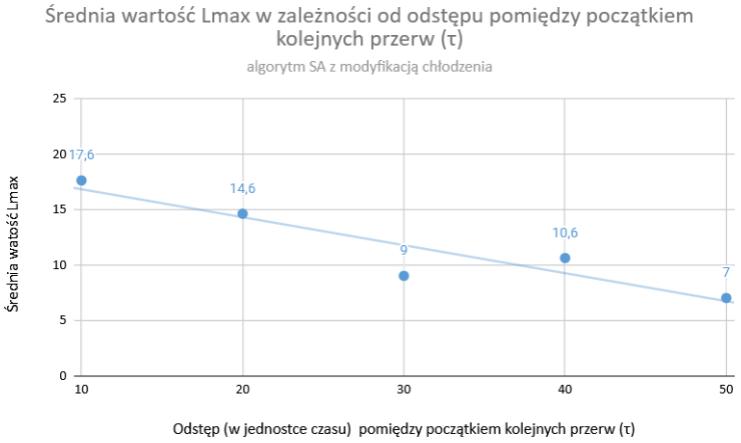


Rysunek 8: Czas obliczeń (ms) w zależności od długości okresu niedostępności h (modyfikacja chłodzenia)

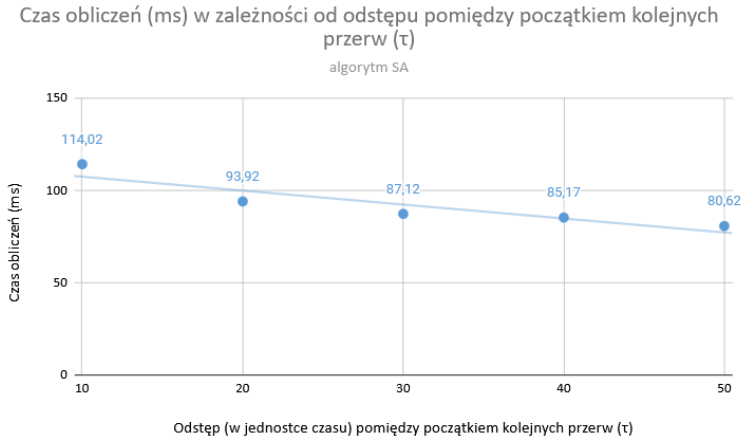
Dłuższe okresy niedostępności przekładają się na wzrost $L_{\max}$, przerwy zmuszają do odraczania zadań. Wersja z modyfikacją (Rys. 6) radzi sobie lepiej przy h=7-9, redukując Lmax w porównaniu do podstawowej wersji (Rys.!6). Sugeruje to, że mechanizm podgrzewania pomaga w adaptacji do trudniejszych warunków harmonogramowania. Dłuższe okresy niedostępności zwiększają czas obliczeń, jednak nadal są one bardzo małe - cały czas operujemy na milisekundach (Rys. 7 oraz Rys. 8).



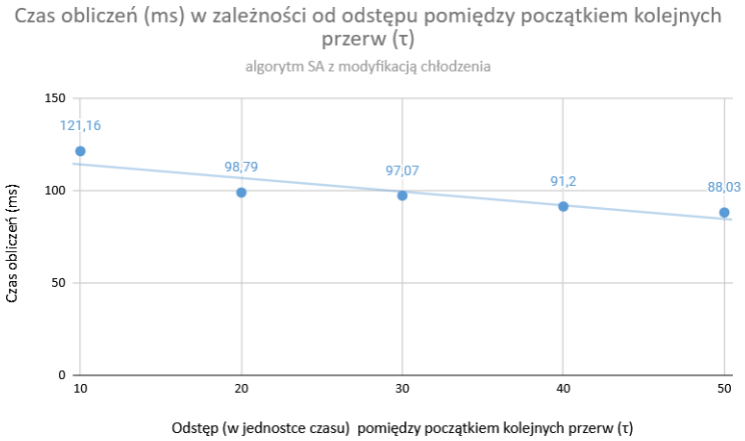
Rysunek 9: Średnia wartość $L_{\max}$ w zależności od odstępu między początkami kolejnych przerw τ



Rysunek 10: Średnia wartość $L_{\max}$ w zależności od odstępu między początkami kolejnych przerw τ (modyfikacja chłodzenia)

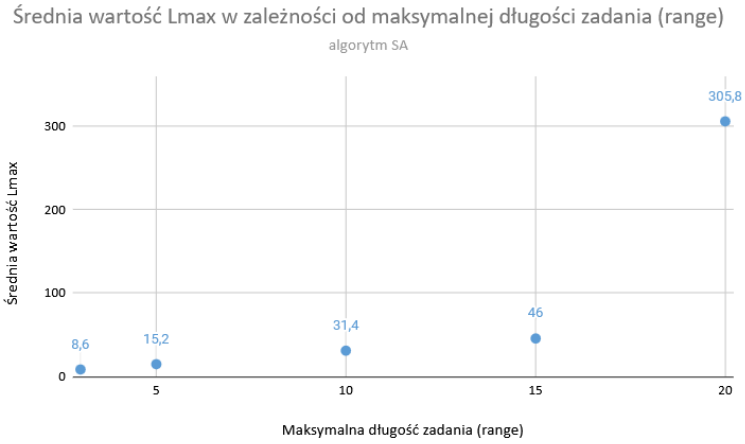


Rysunek 11: Czas obliczeń (ms) w zależności od odstępu między początkiem kolejnych przerw τ

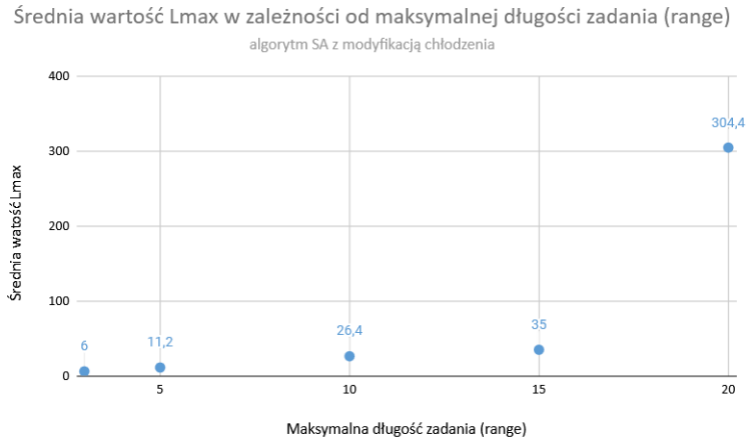


Rysunek 12: Czas obliczeń (ms) w zależności od odstępu między początkami kolejnych przerw τ (modyfikacja chłodzenia)

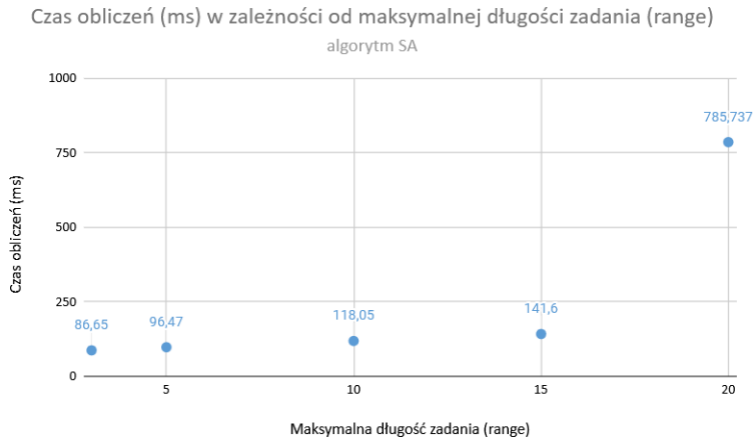
Czyste SA (losowa permutacja oraz klasyczne chłodzenie) jest w zupełności wystarczające, daje niskie wartości $L_{\max}$, a uzyskane czasy obliczeń są niższe od wariantu z modyfikacją chłodzenia. Zwiększenie odstępu τ między przerwami powoduje nieznaczne skrócenie czasu, mniej przerw należy wstawić.



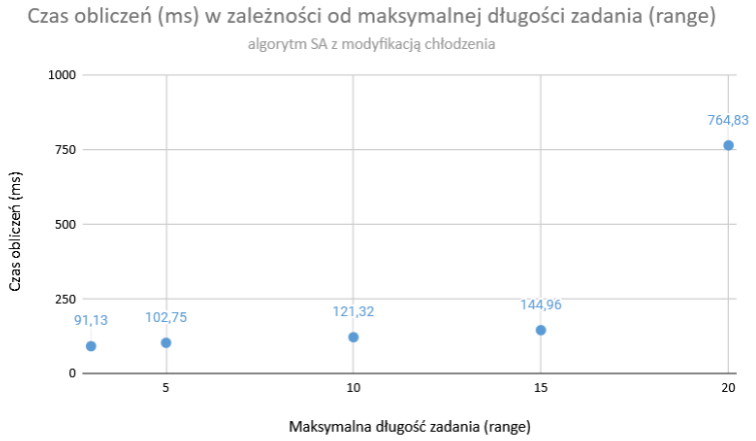
Rysunek 13: Średnia wartość $L_{\max}$ w zależności od maksymalnej długości zadania (range)



Rysunek 14: Średnia wartość $L_{\max}$ w zależności od maksymalnej długości zadania (range) (modyfikacja chłodzenia)



Rysunek 15: Czas obliczeń (ms) w zależności od maksymalnej długości zadania (range)



Rysunek 16: Czas obliczeń (ms) w zależności od maksymalnej długości zadania (range) (modyfikacja chłodzenia)

W obu wariantach zwiększenie maksymalnej długości zadania do 20 bardzo gwałtownie pogorszyło jakość rozwiązania, również czas obliczeń znacznie się wydłużył.

Modyfikacja chłodzenia poprawia jakość rozwiązań (szczególnie dla dużych N i range), ale kosztem dłuższych obliczeń. Dla małych problemów (N mniejsze od 15, range mniejsze od 10) wystarczy podstawowy SA. Przy

dużych zadaniach (range > 15) lub wielu przerwach (h > 5) warto zastosować wersję z podgrzewaniem. W celu dalszej optymalizacji można przetestować kombinację z innymi heurystykami.

4 Podsumowanie

W przedstawionym sprawozdaniu omówiono podejście do problemu szeregowania zadań w układzie przepływowym na dwóch maszynach z warunkiem „no–wait" oraz okresowymi przerwami serwisowymi na drugiej maszynie, oznaczanym jako $F_2\ hT_2 \mid$ no–wait $\mid L_{\max}$. Problem sformalizowano za pomocą programowania matematycznego, wprowadzając zmienne decyzyjne opisujące kolejność zadań i momenty rozpoczęcia na obu maszynach, a także ograniczenia dotyczące jednokrotnego przetworzenia każdego zadania na każdej maszynie, braku oczekiwania pomiędzy maszynami oraz uwzględnienia cyklicznych przerw serwisowych. Kryterium optymalizacji stanowi minimalizacja maksymalnego opóźnienia względem terminów wykonania zadań.

Pierwszą z zaprezentowanych metod była metoda dokładna typu „podział i ograniczenia" (Branch & Bound). Algorytm ten gwarantuje znalezienie rozwiązania optymalnego, jednak jego złożoność rośnie wykładniczo wraz ze wzrostem liczby zadań, co czyni go praktycznym jedynie dla niewielkich instancji (najczęściej dla $n \leq 10$).

Wyniki eksperymentów pokazują, że złożoność obliczeniowa rośnie wraz z liczbą zadań, co jest naturalne ze względu na eksplorację przestrzeni permutacji. Długość okresów niedostępności oraz maksymalny czas między nimi wpływają na elastyczność harmonogramu, co przekłada się na wartość funkcji celu. Szczególnie istotny jest wzrost maksymalnej długości operacji, który znacząco utrudnia znalezienie optymalnego rozwiązania, zwiększając opóźnienia. Mimo to, dzięki mechanizmowi odcinania gałęzi (pruning), algorytm efektywnie redukuje przestrzeń poszukiwań, co pozwala na znalezienie dobrych rozwiązań w rozsądnym czasie nawet dla instancji o długich zadaniach.

Drugą klasę rozwiązań stanowią algorytmy metaheurystyczne oparte na symulowanym wyżarzaniu (Simulated Annealing, SA). Wykonano cztery warianty tej metaheurystyki. W podstawowej wersji permutacja startowa jest losowa, a temperatura chłodzona wykładniczo za pomocą współczynnika $\alpha = 0{,}998$. Drugi wariant wprowadza mechanizm okresowego podnoszenia temperatury („reheating") wraz z detekcją stagnacji, dzięki czemu algorytm zyskuje zdolność wydostawania się z głębokich minimów lokalnych. Trzeci wariant wykorzystuje deterministyczny start zgodny z regułą Earliest Due Date (EDD), co pozwala szybko wyeliminować największe początkowe przekroczenia terminów. Wreszcie czwarty wariant łączy permutację EDD z mechanizmem reheating, łącząc zalety obu poprzednich strategii inicjalizacji i chłodzenia.

Na podstawie wyników eksperymentów wykazano, że dla małych instancji ($n \leq 10$) wszystkie warianty SA osiągają wartości $L_{\max}$ zbliżone do optymalnych, a permutacja EDD pomaga zwłaszcza przy silnie zróżnicowanych terminach. Wraz ze wzrostem liczby zadań i skali instancji najlepszą jakość rozwiązań, przy utrzymaniu wielomianowego czasu działania $O(\text{iteracji} \times n)$, zapewnia wersja SA z mechanizmem pogrzewania, gdyż tradycyjne wyżarzanie bez resetów temperatury zbyt szybko zastaje się w lokalnych minimach, zwłaszcza gdy długie przerwy generują rozległe płaskie fragmenty krajobrazu funkcji celu. Permutacja EDD bez reheatingu potrafi być z kolei wąskim gardłem przy dużych instancjach, ograniczając eksplorację przestrzeni rozwiązań do sąsiedztwa wstępnego układu.

Ostatecznie porównanie z metodą dokładną wykazało, że choć Branch & Bound daje gwarancję optymalności, to metaheurystyki symulowanego wyżarzania stanowią praktyczną alternatywę dla problemów średniej i dużej skali. W szczególności SA z reheatingiem dostarcza rozwiązania bliskie optymalnym w akceptowalnym czasie dla dużych instancji, z którymi algorytm dokładny nie jest sobie w stanie poradzić.