Spis treści

1 Wprowadzenie

Teoria szeregowania zadań zajmuje się problemami polegającymi na przydzieleniu pewnych zadań do dostępnych maszyn w taki sposób, aby pewne kryterium było optymalizowane. Będziemy się zajmować deterministycznymi problemami, czyli takimi, w których wszystkie dane są znane z góry.

2 Postać problemu

Standardowo, problem jest skonstruowany z następujących składowych:

- zadania $\mathcal{J} = \{J_1, \ldots, J_n\}$
- maszyny $\mathcal{P} = \{P_1, \ldots, P_m\}$
- zasoby $\mathcal{R} = \{R_1, \ldots, R_s\}$ dostępnych w $m_1, \ldots, m_s$ jednostkach

2.1 Maszyny

W problemie w zależności od wykonywanego zadania i maszyn mogą występować różne ograniczenia i różnice między maszynami. Jeśli mamy do czynienia z kilkoma maszynami równoległymi to te maszyny mogą być:

- P - identycznościowe czyli z jednakową szybkością
- Q - jednorodne czyli z różną szybkością między maszynami
- R - dowolne czyli z różniącą się szybkością między zadaniami i maszynami

Jeśli mamy do czynienia z maszynami dedykowanymi, gdzie każde zadanie składa się z operacji wykonywanych na różnych maszynach to maszyny mogą być:

- F - system przepływowy czyli każde zadanie przechodzi przez maszyny w tej samej kolejności
- O - system otwarty czyli kolejność wykonywania operacji jest dowolna
- J - system gniazdowy czyli każde zadanie ma ustaloną własną kolejność przechodzenia przez maszyny

2.2 Zadania

Zadanie J opisują następujące atrybuty:

- p_j - czas wykonania zadania J_j
- r_j - czas przygotowania zadania J_j
- d_j - pożądany czas zakończenia zadania J_j
- w_j - waga zadania J_j

2.3 Parametry zadań

Zbiór zadań $\mathcal{J}$ jako całość opisują ograniczenia kolejnościowe (acykliczny graf skierowany), oraz podzielność czyli czy zadania można przerywać i wznawiać.

2.4 Uszeregowanie

Uszeregowaniem nazywamy przypisanie każdemu zadaniu maszyny i zasobów w czasie. Koniecznym jest aby następujące warunki były spełnione:

- w każdej chwili maszyna wykonuje tylko jedno zadanie
- w każdej chwili każde zadanie jest wykonywane przez jedną maszynę
- Każde zadanie jest wykonywane w całości
- Spełnione są ograniczenia kolejnościowe
- Jeśli zadania są podzielne to są one przerywane skończoną ilość razy

2.4.1 Parametry uszeregowania

- moment rozpoczęcia S_j
- moment zakończenia C_j
- czas przepływu $F_j = C_j - S_j$
- opóźnienie $L_j = C_j - d_j$
- spóźnienie $T_j = \max(0, L_j)$
- przyspieszenie $E_j = \max(0, d_j - C_j)$
- liczbę spóźnionych zadań $U_j = |\{i : C_i > d_i\}|$

2.4.2 Kryteria optymalizacji

Typowo w szeregowaniu optymalizujemy jakąś funkcję składającą się z parametrów uszeregowania. Przykładowe funkcje to: $C_{max} = \max(C_j)$, $C_{sum} = \sum C_j$ czy $T_{sum} = \sum T_j$.

2.5 Notacja Trójpolowa

$$\alpha|\beta|\gamma$$

gdzie α określa ograniczenia maszyn, β określa ograniczenia zadań, a γ określa kryterium optymalizacji.

3 Problemy na jednej maszynie

Poniżej są opisane raczej trywialne problemy z minimalnymi utrudnieniami.

3.1 $1||C_{max}$

Jest jedna maszyna i n zadań z czasami przetwarzania p_j. Każde uszeregowanie bez przestojów jest optymalne. $C_{max} = \sum_{j=1}^{n} C_j$.

3.2 $1|r_j|C_{max}$

Jest jedna maszyna i n zadań z czasami przetwarzania p_j i czasami, od których są dostępne r_j. Tutaj możliwe, że przestoje są nieuniknione. Aby rozwiązać ten problem, sortujemy zadania po r, po czym kolejno je szeregujemy.

3.3 $1||\sum C_j$

Jest jedna maszyna i n zadań z czasami przetwarzania p_j. Chcemy minimalizować sumę ich czasów zakończeń, więc lepiej najpierw wykonać najkrótsze zadania. Nazywamy to SPT.

3.4 $1||\sum w_j C_j$

Jest jedna maszyna i n zadań z czasami przetwarzania p_j i wagami w_j. Chcemy minimalizować ważoną sumę ich zakończeń, co za tym idzie najwyżej ważone zadania chcemy wykonywać jako pierwsze. Aby to osiągnąć wystarczy posortować zadania po $\frac{p_j}{w_j}$. Nazywamy to WSPT (reguła Smitha) i jest to ogólniejszy przypadek SPT ($w_j = 1$).

3.5 $1|pmtn, r_j|\sum C_j$

Jest jedna maszyna i n zadań z czasami przetwarzania p_j, z czasami gotowości r_j, które **są podzielne**. Tutaj ponownie stosujemy algorytm SPT, ale w momencie, gdy skończymy zadanie, lub zadanie stanie się dostępne, zmieniamy wykonywane zadanie na to o najkrótszym czasie przetwarzania. Nazywamy to SRPT.

3.6 Problemy kolejnościowe

W momencie dodania ograniczeń kolejnościowych, z reguły opisanych grafem kolejności, złożoność problemu znacznie rośnie. Załóżmy, że graf jest reprezentowany przez listę sąsiedztwa. Rozróżniamy następujące rodzaje ograniczeń kolejnościowych:

- *prec*: dowolne
- *chains*: zadania są w liniach
- *in-tree*: zadania są w drzewie, gdzie rodzice muszą być wykonane przed dziećmi
- *out-tree*: zadania są w drzewie, gdzie dzieci muszą być wykonane przed rodzicem

3.6.1 $1|\textbf{prec}|C_{max}$

W tym problemie mamy jedną maszynę i n zadań nieprzerywalnych z czasami przetwarzania p_j oraz dowolnymi ograniczeniami kolejnościowymi.

$$C_{max} = \sum_{j=1}^{n} p_j$$

Problemem nie jest optymalizacja czasu wykonania, albowiem dowolne szeregowanie dot. tego problemu rozwiązuje też problem bez ograniczeń kolejnościowych. Koniecznym jest sortowanie topologiczne grafu zależności.

W tym problemie wystarczy DFS, którego jedno przejście po grafie stworzy ścieżkę która będzie optymalnym szeregowaniem

3.6.2 $1|\textbf{prec}|\sum C_j$

Próba rozwiązania tego problemu tylko przy pomocy DFS nie wystarczy. Konieczny jest algorytm Kahna, który sortuje topologicznie graf. Algorytm Kahna jest bardzo podobny do DFS, ale zamiast używania stosu, używamy kolejki priorytetowej, czyli najpierw zwiedzamy te wierzchołki o najkrótszym czasie przetwarzania. Można sobie wyobrazić pracę algorytmu Kahna jako usuwanie węzłów z grafu, które nie mają żadnych następników, najpierw tego, które mają najmniejszy czas przetwarzania.

3.6.3 1|prec|f_{max}

W tym problemie mamy jedną maszynę i n zadań nieprzerywalnych z czasami przetwarzania p_j. Wystarczy rozważać szeregowania bez przestojów. Problem ten rozwiązuje algorytm Lawler'a, w którym budujemy uszeregowanie od końca. W każdym kroku algorytmu rozważamy te zadania, bez uszeregowanych następników, i wybieramy te które w danym momencie (odliczając czas od końca $p = \sum p_j$) generuje najmniejszy koszt. Wybrane zadanie trafia na początek uszeregowania.

3.6.4 1|out-tree|$\sum w_j C_j$

W tym problemie mamy jedną maszynę i n zadań nieprzerywalnych z czasami przetwarzania p_j oraz wagami. Graf kolejnościowy jest podany jako out-tree, czyli korzeń nie ma poprzedników. Trik aby rozwiązać ten problem, to łączyć kolejno zadania o najmniejszej wartości kryterium ($\frac{w_j}{p_j}$) z swoimi poprzednikami, pamiętając w poprzednikach o kolejności łączenia.

3.6.5 1|in-tree|$\sum w_j C_j$

Graf kolejnościowy jest podany jako in-tree, czyli korzeń ma poprzedników, zgodnie z tym ile ma dzieci. Ten problem mapuje się 1-1 do problemu poprzedniego, wystarczy tylko odwrócić krawędzie i nadać zadaniom wagi przeciwne.

3.6.6 Trudne jednomaszynowe problemy szeregowania zadań

- **1|prec|**$\sum w_j C_j$
- **1|**r_j**|**$\sum C_j \rightarrow$ **1|**r_j**|**$\sum w_j C_j$
- **1|pmtn,** r_j**|**$\sum w_j C_j$

3.7 Problemy z opóźnieniem

W tych problemach, optymalizujemy opóźnienie, czyli $C_j - d_j$.

3.7.1 1||L_{max}

Mamy jedną maszynę i n zadań nieprzerywalnych z czasami przetwarzania p_j i czasami oczekiwanymi d_j. Aby rozwiązać ten problem, wystarczy zastosować algorytm EDD (regułę Jacksona), w którym wybieramy najpierw zadanie o najwcześniejszym czasie oczekiwania. EDD to szczególny przypadek algorytmu lawlera, gdzie $f = L_j(t) = t - d_j$.

3.7.2 1|prec|L_{max}

Mamy jedną maszynę i n zadań nieprzerywalnych oraz graf kolejności. Ten problem rozwiązuje algorytm analogiczny do problemu 1|prec|C_{max}, czyli sortujemy topologicznie, ale tym raze po d_j a nie p_j.

3.7.3 1|r_j|L_{max}

To jest problem silnie NP-trudny, trudność wynika z niepodzielności zadań. Warto wspomnieć, że ten problem nie jest NP-trudny, jeśli $r_j = 1$, i wtedy rozwiązuje go EDD.

3.7.4 1|pmtn,r_j|L_{max}

W odróżnieniu od poprzedniego problemu, tutaj zadania są podzielne. Aby rozwiązać ten problem wystarczy zastosować regułę EDD, pamiętając o podzielności zadań (algorytm Horna). Zawsze jak zadanie kończy wykonywanie, lub nowe zadanie jest dostępne, to wybieramy zadanie o najmniejszym d_j.

3.7.5 1|prec,$r_j, p_j = 1$|L_{max}

W każdej całkowitej chwili czasu t, wybieramy zadanie o najmniejszym d_j, bez niewykonanych poprzedników.

3.7.6 1|prec,pmtn,r_j|L_{max}

W każdym momencie decyzji (zakończenia zadania lub pojawienia się nowego zadania) dostępne zadanie o najmniejszym d_j bez niewykonanych poprzedników.

3.8 1||$\sum U_j$

Tutaj optymalizujemy ilość spóźnionych zadań. W pewnym sensie nie interesuje nas ile jesteśmy spóźnieni, ale czy jesteśmy spóźnieni. Czyli mamy n zadań o czasie przetwarzania p_j i czasie oczekiwania d_j. Aby rozwiązać ten problem, wybieramy niespóźnione zadania w porządku EDD, potem robimy te spóźnione w dowolnym porządku. Czyli dobierasz do zbioru zadania, jeśli dodanie kolejnego zadania spowoduje, że zadanie jest spóźnione to usuń z zbioru zadanie o największym czasie wykonania.

3.9 1||$\sum T_j$

Problem ten jest słabo NP-trudny, generalizacja z wagami ($\sum w_j T_j$) jest silnie NP-trudna. Wersja z przerywaniem też jest silnie NP-trudna. Z kolei $p_j = 1$ znacznie ułatwia obydwa problemy i powoduje że należą do klasy P.

3.10 1||$\sum w_j E_j$

Problem ten jest równoważny z problemem $1||\sum w_j T_j$. $E_j = \max(0, L_j - d_j)$, co oznacza, że dla równoważnego problemu i $C = \sum_{i=1}^{n} p_i$ czasy oczekiwania $d'_j = C - d_j + p_j$. Optymalne uszeregowanie to odwrotność optymalnego uszeregowania S'.

4 Problemy wielu maszyn

Problemy wielu maszyn mają okropną tendencję bycia NP-trudnymi. Problem $P_2||C_{max}$ jest NP-trudny. Problemy wielu maszyn często aproksymują algorytmy listowe, w których do n wolnych maszyn przyporządkowujemy kolejno zadania z *jakiejś* kolejki. Im większa jest różnica między najdłuższym a najkrótszym czasem wykonania zadania, tym bardziej mylne rozwiązania dają algorytmy listowe.

4.1 P||C_{max}

Jest to problem silnie NP-trudny. Ten problem aproksymuje algorytm LPT, czyli sortujemy zadania nierosnąco wobec p_j i przypisujemy je kolejno do maszyn. Jest to algorytm listowy, w którym zadania w liście są posortowane nierosnąco wobec p_j. Współczynnik aproksymacji LPT dla tego problemu wynosi $\frac{4}{3} - \frac{1}{3m}$.

4.2 P|prec|C_{max}

Ten problem może aproksymować LPT, z modyfikacją w której jeśli zadanie nie ma spełnionych wymagań to nie może być zdjęte z kolejki. Niestety, ten problem jest szczególnie podatny na anomalie szeregowania listowego, gdzie wejścia "łatwiejsze"mogą dawać gorsze rezultaty niż "trudniejsze".

4.3 P|in-tree,$p_j = 1$|C_{max}

Mamy m identycznych maszyn równoległych Poziom zadania to jego odległość od korzenia. Problem ten może rozwiązać algorytm Hu, gdzie w każdej chwili t, wybieramy m zadań z najwyższymi poziomami. Poziom zadania to jego odległość od korzenia.

4.4 P|pmtn|C_{max}

Mamy m identycznych maszyn równoległych i n podzielnych zadań.

$$C^*_{max} = \max\{\frac{1}{m}\sum_{j=1}^{n} p_j, \max_{j=1}^{n} p_j\}$$

Optymalne uszeregowanie konstruuje algorytm McNaughton'a. Zadania są przypisywane do kolejnych maszyn w przedziale $[0, C^*_{max})$. Jeśli zadanie miałoby się kończyć w czasie $> C^*_{max}$, to jego "overflow"trafia na następną maszynę.

4.5 $\mathbf{P}|p_j = 1, r_j|L_{max}$

Jako, że zadania są równej długości, to jest to zdecydowanie prostszy problem. Kolejno wybieramy m zadań do wykonania w "turze", najpierw wykonując te, które mają najwcześniejsze terminy zakończenia.

4.6 $\mathbf{P}|\textbf{in-tree},p_j = 1|L_{max}$

Mamy m identycznych maszyn równoległych. Aby rozwiązać ten problem, wystarczy zmodyfkować $d_i = \min\{d_i, d_{i-1}\}$, a następnie szeregować zgodnie z EDD.

4.7 $\mathbf{P}|p_j = 1|\sum w_j U_j$

Problem sprowadza się do wybrania zbioru zadań, które zostaną wykonane na czas. Zadania nie spóźnione można przydzielić do maszyn LPT z porządkiem EDD. Czyli najpierw sortujemy zadania według d_j, potem dodajemy do zbioru zadania zgodnie z tym porządkiem. Jeśli dodanie zadania powoduje przekroczenie limitu czasu, oraz to zadanie ma wagę wiekszą niż najmniejsza waga zadania w zbiorze, to usuwamy zadanie o najmniejszej wadze.

4.8 Maszyny nie-identyczne

Maszyny niekoniecznie muszą być identyczne, ale jednorodne. Mogą być różne w pojemnościach przetwarzania. Prędkość oznaczamy v_i. Czas wykonywania zadania j na maszynie i wynosi $\frac{p_j}{v_i}$.

4.8.1 $\mathbf{Q}||\sum C_j$

Zadania sortujemy i wykonujemy z porządkiem SPT. Jeśli $t_j = \frac{k}{v_i}$, to zadanie J_j należy uszeregować na maszynie M_i jako k-te od końca. Niech $w_i = \frac{1}{v_i} : i < m$. Następnie dla każdego zadania znajdujemy najmniejsze w_i w liście, umieszczamy zadanie na początek uszeregowania i-tej maszyny, po czym $w_i += \frac{1}{v_i}$.

4.8.2 $\mathbf{Q}|\textbf{pmtn}|\sum C_j$

Najktótsze zadanie chcemy wykonywać na maszynie o najwyższej prędkości. W momencie wykonania zadania, przerzucamy drugie najkrótsze zadanie na tą maszynę, itd.

5 Maszyny dowolne

Czas wykonywania zadania j na maszynie i wynosi p_{ij}. Czasy wykonywania zadań są zapisane w macierzy $n \times m$. Problem R|pmtn|$C_{max} \in P$, i rozwiązuje go algorytm na znajdowanie optymalnego przepływu w sieci. Problem R|$|\sum C_j \in P$, ponieważ można go sprowadzić do problemu przydziału zadań. Problem R|pmtn|$\sum C_j \in NP$.

6 Programowanie dynamiczne

Programowanie dynamiczne, to technika w której algorytm rozwiązuje problem rekurencyjnie na podstawie ogromnej tablicy w której wyniki tymczasowe są przechowywane. Programowanie dynamiczne jest idealne, w momencie kiedy problemy można podzielić na podproblemy, idealnie powtarzające się. Największym problemem z programowaniem dynamicznym jest złożoność pamięciowa.

6.1 Problem plecakowy

Mamy n przedmiotów o wagach w_i i cenach c_i. Mamy też daną maksymalny łączny rozmiar B. Maksymalna łączna cena podzbioru przedmiotów o numerach nieprzekraczających i mieszczących się w plecaku o pojemności j:

$$A[i,j] = \begin{cases} 0 & i = 0 \vee j = 0 \\ A[i-1,j] & i,j > 0 \wedge w_i > j_j \\ \max\{A[i-1,j], A[i-1,j-w_i] + c_i\} & i,j > 0 \wedge w_i \leq j \end{cases}$$

Tworzymy tablicę A o rozmiarach $n+1 \times B+1$ i zwracamy $A[n,B]$. Aby stworzyć rozwiązanie, dla $j = B$, oraz $i = n$, jeśli $A[i,j] \neq A[i-1,j]$ wkładamy przedmiot i do plecaka, $j- = w_i$ oraz $i- = 1$.

6.2 P2$||C_{max}$

Mamy problem plecakowy, gdzie $w_j = c_j = p_j$, oraz $B = \lfloor \sum_{j=1}^{n} \frac{p_j}{2} \rfloor$.

6.3 Pm$||C_{max}$

Wykorzystujemy tablicę $m+1$ wymiarową. W komórce $A[j, t_1, \ldots, t_m]$ zapisujemy informację czy zadania o numerach nieprzekraczających j można przypisać w taki sposób, że czas działania maszyny $P_i = t_i$.

1. $A[0, t_1, \ldots, t_m] = false$
2. $A[0, 0, \ldots, 0] = true$
3. $A[1..n, t_1, \ldots, t_m] = \bigvee_{i=1}^{n} A[j-1, t_1, \ldots, t_{i-1}, t_i - p_j, t_{i+1}, \ldots, t_m]$
4. return $\min\{\max\{t_1, \ldots, t_m\} | A[n, t_1, \ldots, t_m] = true\}$

7 Podział i ograniczenia

Zwiedzamy drzewo rozwiązań problemu. Koniecznym jest określenie strategii wybierania gałęzi, oraz strategii ograniczenia aby uniknąć zwiedzania poddrzew, które nie mogą zawierać rozwiązania. Ogólnie ta metoda jest gorsza od programowania dynamicznego, ale potrafi osiągać lepszą złożoność pamięciową, ponieważ nie wymaga przechowywania wszystkich podproblemów. Dodatkowo nie zawsze można stosować programowanie dynamiczne.

8 System otwarty

Jeśli maszyny nie są jednorodne, to mogą być podzielone na grupy maszyn, które wykonują różne zadania. Każde zadanie J_j składa się z ciągu operacji $O_{1j}, \ldots O_{mj}$. Operację O_{ij} wykonuje maszyna P_i z czasem p_{ij}. W systemie otwartym zadania mogą być przypisane do dowolnej maszyny, która jest wolna. Nie ma kolejności w jakiej zadania mają być przypisane do maszyn.

8.1 O2$||C_{max}$

Mamy 2 maszynynowy system otwarty. To znaczy, że każde zadanie składa się z dwóch operacji, bez określonej kolejności. Problem ten rozwiązuje LAPT, gdzie za każdym razem gdy maszyna jest wolna, wybieramy do niej dostępne zadanie o najdłuższym czasie wykonania na alternatywnej maszynie.

$$C_{max} = \max\{\max_{j=1}^{n}\{p_{1j} + p_{2j}\}, \sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}\}$$

8.2 Inne problemy

Wielomianowe:

- O|pmtn|C_{max}
- O|r_j,pmtn|L_{max}

NP-trudne:

- O||C_{max}
- O2||$\sum C_j$
- O2|r_j|C_{max}
- O2||L_{max}

9 Flow Shop

System przepływowy to system podobny do systemu otwartego, ale z dodatkową warunkiem, że zadania muszą być przeprowadzone w określonej kolejności. Z reguły zgodnie z numerowaniem maszyn.

9.1 Szeregowanie permutacyjne

Dla każdego problemu w systemie przepływowym, optymalne jest uszeregowanie, w którym kolejność jest taka sama na dwóch pierwszych maszynach. Jeśli zadania są wykonywane na wszystkich maszynach w porządku $J_1, J_2, \ldots, J_n$, to operacja O_{ij} zaczyna się w momencie, kiedy zakończyły się operacje $O_{i,j-1}$ i $O_{i-1,j}$. Zatem:

$$C_{ij} = \begin{cases} 0 & \text{jeśli } i = 1 \text{ lub } j = 1 \\ \max\{C_{i-1,j}, C_{i,j-1}\} + p_{ij} & \text{w przeciwnym wypadku} \end{cases}$$

9.2 F2||C_{max}

Mamy 2 maszyny, gdzie najpierw zadanie musi trafić na pierwszą maszynę, a potem na drugą. Problem ten rozwiązuje algorytm Johnsona. Dzielimy zadania na dwa podzbiory: $\mathcal{A} = \{J_j \in J : p_{1j} \leq p_{2j}\}$ oraz $\mathcal{B} = \{J_j \in J : p_{1j} > p_{2j}\}$. Wykonaj najpierw zadania ze zbioru $\mathcal{A}$, w porządku niemalejących p_{1j} a potem zadania ze zbioru $\mathcal{B}$ w porządku niemalejących p_{2j}.

9.3 F3||C_{max}

Ten problem jest NP-trudny. Jeśli maszyna P_2 jest zdominowana to optymalne rozwiązanie tego problemu to algorytm Johnsona, gdzie $p'_{1j} = p_{1j} + p_{2j}$ oraz $p'_{2j} = p_{2j} + p_{3j}$. Maszyna P_2 jest zdominowana jeśli $\min p_{1j} \geq \max p_{2j} \vee \min p_{3j} \geq \max p_{2j} \vee \min\{p_{1j}, p_{3j}\} \geq p_{2j}$.

9.4 F||C_{max}

Jako, że ten problem jest NP-trudny, z reguły w kontekście tego problemu mówi się o heurystykach. W kontekście tego problemu z reguły używa się heurystyki NEH. Najpierw sortujemy zadania według nierosnących wartości $\sum_{i=1}^{m} p_{ij}$. Następnie, dla każdego zadania, rozważamy zbiór powstały z dodania tego zadania do najlepszego dotychczasowego rozwiązania na każdym miejscu. Wynikiem jest najlepsze rozwiązanie.

10 System gniazdowy

Jest to uogólnienie systemu przepływowego. Mamy n zadań i m maszyn. Zadanie J_j składa się z ciągu n_j operacji $O_{n_j j}$. Operację O_{ij} należy wykonać na maszynie P_{ij}. Żadne dwie operacje tego samego zadania nie mogą być wykonywane równocześnie. W odróżnieniu od systemu przepływowego, zadania mogą być wykonywane w dowolnej kolejności na każdej maszynie.

10.1 $J_2|n_j \leq 2|C_{max}$

Dzielimy zbiór zadań na 4 podzbiory:

- I_1 - do wykowania w całości na P_1
- I_2 - do wykowania w całości na P_2
- $I_{1,2}$ - do wykowania najpierw na P_1, a potem na P_2
- $I_{2,1}$ - do wykowania najpierw na P_2, a potem na P_1

Następnie rozwiązujemy $F_2||C_{max}$ dla $I_{1,2}$ oraz $I_{2,1}$ otrzymując szeregowania $R_{1,2}$ i $R_{2,1}$. Na maszynie P_1 szeregujemy: $R_{1,2}$, I_1, potem $R_{2,1}$ i potem podobnie dla maszyny P_2.

10.2 Inne problemy

Silnie NP-trudne problemy:

- $J_2|chains, p_{ij} = 1|C_{max}$
- $J_2|p_{ij} \in \{1, 2\}|C_{max}$
- $J_3|p_{ij} = 1|C_{max}$
- $J_2|pmtn|C_{max}$ lub $\sum C_j$

11 Metaheurystyki

Polegają na przeszukiwaniu przestrzeni rozwiązań problemów. Nie dają żadnych gwarancji optymalności, ale z reguły charakteryzują się stabilnym czasem wykonania.

11.1 Przeszukiwanie lokalne

Wybieramy rozwiązanie początkowe. Następnie znajdujemy sąsiedztwo tego rozwiązania i wybieramy najlepsze rozwiązanie z sąsiedztwa. Powtarzamy proces aż do osiągnięcia lokalnego minimum. Z reguły sąsiedztwo implikuje *jedną* zmianę.

11.1.1 Wielokrotne przeszukiwanie lokalne

Wielokrotnie wykonujemy przeszukiwanie lokalne, idealnie w różnych miejscach przestrzeni rozwiązań. Można to osiągnąć wybierając losowo rozwiązanie początkowe.

11.1.2 Iteracyjne przeszukiwanie lokalne

Zaczynamy od przeszukania lokalnego. Wynik tego przeszukiwania jest **perturbowany** i wykorzystywany do przeszukiwania lokalnego. Jeśli osiągnięty wynik jest lepszy niż poprzedni, to zapisujemy go jako najlepszy wynik. Jeśli nie, to powtarzamy proces aż do osiągnięcia maksymalnej liczby iteracji. Perturbacja implikuje *kilka* zmian.

11.1.3 Przeszukiwanie zmiennego sąsiedztwa

Algorytm ten zakłada przeszukanie całego sąsiedztwa, po czym zmianę sąsiedztwa. W pewnym sensie, znalezienie sąsiedniego sąsiedztwa i przeszukania go.

11.2 Wyrzarzanie

Idea jest taka, że skaczemy między różnymi częściami przestrzeni rozwiązań. Wraz z trwaniem algorytmu, częstotliwość i amplituda skoków zmniejsza się, podobnie jak temperatura podczas wyrzarzania stali.

Zaczynamy w pewnym stanie i energii. Nowy stan powstaje poprzez perturbację poprzedniego stanu, jeśli jego energia jest mniejsza niż poprzedni stan, to zapisujemy go jako najlepszy wynik. Jeśli nie, to powtarzamy proces aż do osiągnięcia maksymalnej liczby iteracji. Perturbacja implikuje *kilka* zmian. Jeśli zmiana zwiększa energię, to jest akceptowana z prawdopodobieństwem:

$$P(j) = e^{-\frac{\Delta E}{kT}} \quad k \approx 1.38 \cdot 10^{-23} \frac{J}{K}$$

Z czasem spada temperatura oraz ilość wykonywanych perturbacji. Prędkość spadku tych wartości określa schemat chłodzenia i jest zależny od parametrów algorytmu.

11.3 Tabu search

Przeszukujemy przestrzeń rozwiązań jak drzewo. Wraz z trwaniem algorytmu, wykonane zmiany są zapamiętywane w blackliście. Blacklista ma pewną skończoną długość. Jeśli blacklista jest pełna, to usuwamy najstarszy element. Krok musi być zgodny z blacklistą, ale zgodnie z pewnym kryterium "aspiracji"możemy ją zignorować.