
Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki
Kierunek Informatyka

**Praca dyplomowa magisterska**

# Algorytm wyszukiwania z tabu do rozwiązywania problemu układania planu zajęć.

Autor: inż. Michał Szluzy
Kierujący pracą: prof. dr hab. inż. Zbigniew Czech

Gliwice, czerwiec 2017

**Streszczenie**

The abstract will go here....
W tym miejscu można umieścić abstrakt pracy. W przeciwnym wypadku należy usunąć/zakomentować ninijeszy fragment kodu.

# Spis treści

# Spis rysunków

# Spis tabel

# Rozdział 1

# Wprowadzenie.

Ala ma kota i kot ma ale

# Rozdział 2

# Problem układania planu zajęć

## 2.1 Sformułowanie problemu

Problem układania planu zajęć można sformułować następująco. Dana jest lista określonych wydarzeń, dostepnych okien czasowych i zasobów. Należy przyporządkować wydarzeniom okna czasowe oraz zasoby w taki sposób, by spełnić przyjęte założenia. W formułowanym problemie:

- wydarzeniami są poszczególne lekcje odbywające się w ciągu jednego tygodnia,
- dostępnymi oknami czasowymi są godziny w których mogą odbywać się zajęcia (np. od poniedziałku do piątku w godzinach między 8:00 a 18:00),
- zasoby to dostępne sale lekcyjne, nauczyciele prowadzący zajęcia, grupy uczniów itp.

Układanie planu zajęć wymaga spełnienia określonych ograniczeń. Mianowicie nie można wypełnić dostepnych okien czasowych według z góry przyjętej kolejności, ponieważ jest prawdopodobne, że pojawią się konflikty i plan zajęć nie będzie możliwy do zrealizowania. Aby tego uniknąć trzeba spełnić tzw. ograniczenia twarde, czyli takie, które zawsze muszą zostać spełnione, by plan mógł być możliwy do zrealizowania. Istnieją również ograniczenia miękkie, które określają jakość ułożonego planu. Ograniczenia te nie muszą zostać spełnione, ale algorytm układania planu zajęć powinien brać je pod uwagę.

Przykłady ograniczeń:

- twarde:
  - zasób nie może być wykorzystany w dwóch miejscach w tym samym czasie (np. dany nauczyciel nie może prowadzić jednocześnie zajęć w dwóch różnych salach),
  - nie występują zajęcia, które nie zostały przypisane do okien czasowych w ułożonym planie.
- miękkie:
  - brak dłuższych przerw między zajęciami dla uczniów,
  - odpowiednie przerwy między zajęciami (np. 15 - lub 20 - minutowe),
  - liczba dni roboczych, w których nauczyciele prowadzą zajęcia, powinna być minimalizowana,
  - brak bloków zajęć danego typu (np. dana grupa uczniów nie powinna mieć kolejno czterech lekcji matematyki).

Często jednak dla wyjściowej siatki zajęć spełnienie wymagań twardych jest niemożliwe. Wynika to najczęściej z za małej liczby zasobów (za mało nauczycieli mogących prowadzić jeden przedmiot lub za mało sal). W algorytmach szuka się wtedy planu z najmniejszą liczbą niespełnionych wymagań, które są usuwane ręcznie przez szukanie określonych kompromisów (np. dołożenie okna czasowego, zatrudnienie dodatkowego nauczyciela, zaplanowanie zajęć tego samego typu dla dwóch mniejszych grup w jednej sali).

Ograniczenia miękkie to w istocie życzenia nauczycieli i uczniów co do tego jak plan powinien wyglądać. Niespełnienie tych ograniczeń wpływa na końcową ocenę planu zajęć, niespelnione ograniczenia mogą mieć określone wagi w zależności od ich istotności. Niektóre plany zajęć układane są ze szczególnym uwzględnieniem uczniów (mała liczba okienek, równomierne rozłożenie zajęć, brak bloków zajęć tego samego typu, odpowiednia przerwa między zajęciami), a niektóre z uwzględnieniem potrzeb prowadzących (zajęcia skumulowane w ciągu dwóch dni tygodnia by umożliwić prace w innej szkole). Założenia te z reguły precyjzuje osoba uruchomiająca wykonanie planu przez odpowiednie skonfigurowanie parametrów algorytmu i zdefiniowanie funkcji oceny wygenerowanego planu.

Funkcja oceny planu zajęć z reguły przewiduje nakładanie punktów karnych za niespełnienie określonych ograniczeń. Każde ograniczenie ma ustaloną liczbę punktów karnych, przy czym ograniczenia twarde powinny mieć dużo większą wagę od ograniczeń miękkich. Im więcej punktów karnych ma plan tym jest gorszy. To właśnie na podstawie funkcji oceny planu większość algorytmów starta się ułożyć jak najlepszy plan zajęć.

## 2.2 Rozmiar problemu

By dobrze zrozumieć potrzebę używania algorytmów heurystycznych przy wyszukiwaniu najlepszego planu zajęć warto przybliżyć pojęcie skali problemu. W tym celu przedstawimy przykład planu zajęć dla amerykańskiej szkoły średniego rozmiaru. W przykładzie tym, każde ze zdarzeń ma przyporządkowane wcześniej zasoby. Przykład ten zawiera:

- okna czasowe - 40,
- nauczyciele - 27,
- sale - 30,
- grupy uczniów - 24,
- zdarzenia - 832,
- ograniczenia - 4.

Dostępnych jest 40 okien czasowych, które rozłożone na pięć dni zajęć w tygodniu dają osiem godzin zajęć dziennie. Jednocześnie odbywać się mogą 24 zajęcia. Liczba ta wynika z niepodzielności zasobów i jest minimalną liczbą spośród liczby nauczycieli, sal i grup uczniów. Maksymalna liczba zdarzeń, które mogą się odbywać w danym tygodniu wynosi więc:

$$40 \cdot 24 = 960,$$

$$960 > 832.$$

Maksymalna liczba zdarzeń jest większa od liczby zdarzeń zawartych w siatce zajęć podanego przykładu, więc realizacja planu zajęć zgodnego z powyższą specyfikacją zasobów jest możliwa.

Na jedną grupę uczniów przypada około 35 zdarzeń (832/24), które powinny być rozłożone na 40 okien czasowych. Uwzględniając możliwość wystąpienia okienek w trakcie zajęć, mamy więc 40-wyrazowy ciąg zdarzeń. Liczba możliwośc,i w które można te zajęcia rozłożyć,określa liczba permutacji 40!, która wynosi:

$$40! = 815915283247897734345611269596115894272000000000$$

Jest to 48 cyfrowa liczba możliwości rozłożenia zajęć w oknach czasowych dla jednej grupy. Liczbę tę trzeba pomnożyć przez liczbę grup.

Przedstawiony przykład ma z góry przyporządkowane zasoby do zdarzeń, więc każde zajęcia mają przyporządkowaną grupę, nauczyciela i salę. Dzięki temu przestrzeń możliwych rozwiązań dla algorytmu ukladania planu zajęć zmniejsza się. Szukany jest tylko termin odbywania się zajęć. Tak przygotowane dane najtrafniej odzwierciedlają realia współczesnych szkół, gdzie najczęściej prowadzący z góry przypisany jest do jakiejś grupy, a konkretne przedmioty mogą się odbywać tylko w przystosowanych do tego celu salach.

Podejście, w którym zasoby z góry są przypisane do zdarzeń, umożliwia szybsze działanie algorytmu. Jednak trzeba pamiętać, że ma to również wpływ na wynik końcowy. Może się zdarzyć przypadek, w który zamiana nauczycieli sprawi, że cały plan znacznie się zmieni i osiągniemy o wiele lepszy rezultat. Jednak z uwagi na złożoność obliczeniową takiego rozwiązania, jak i na dostępną liczbę danych testowych, które mają już przypisane zasoby, zdecydowano się wybrać wariant, w którym algorytm poszukuje tylko okien czasowych dla wszystkich dostępnych zdarzeń.

# Rozdział 3

# Algorytm wyszukiwania z tabu.

## 3.1 Algorytmy heurystyczne i metaheurystyczne.

Optymalizacja rozwiązań wykorzystywana jest w prawie każdym aspekcie życia. Podwyższanie jakości usług, obniżanie kosztów wyrobów czy minimalizacja zużycia surowców to w bardzo popularne zagadnienia. Medycyna, logistyka, ekonomia to tylko niektóre dziedziny, w których pojęcie to znajduje swoje zastosowanie. Optymalizacja to minimalizacja bądź maksymalizacja pewnej funkcji, zwanej często funkcją oceny, która jednoznacznie określa jakość danego rozwiązania. Znalezienie minimum lub maksimum wymaga więc wyznaczenia funkcji oceny dla każdego możliwego wariantu danego problemu i wyborze tego o najlepszym wyniku. Niestety często rozmiar zadania, dla którego szukamy optymalnego rozwiązania, jest tak duży, że przeszukanie wszystkich możliwości bądź nawet zastosowanie jakiegoś algorytmu znajdującego najlepsze rozwiązanie nie jest możliwe ze względu na czas wykonania. Liczba operacji, które należy wykonać często rośnie wykładniczo w stosunku do rozmiaru problemu. Problem ten stwarza miejsce dla heurystyki.

„Terminem heurystyka (z języka greckiego heurisko - znajduję) określa się sposób postępowania oparty na zdobytym doświadczeniu, wykorzystaniu istniejących faktów i reguł, w celu znalezienia odpowiedzi na postawione pytanie." [1] Algorytmy heurystyczne więc to takie algorytmy, które opierając się na własnym przebiegu i otrzymywanych wynikach, starają się znaleźć jak najlepsze rozwiązanie, jednak nie zawsze musi to być rozwiązanie najlepsze z dostępnych. W zamian za możliwość otrzymania gorszego rozwiązania otrzymujemy szybszy czas działania algorytmu. Algorytmy heurystyczne

wykorzystywane są gdy dokładne algorytmy są z przyczyn technicznych zbyt kosztowne lub gdy są nieznane (np. przy problemie przewidywania pogody). Często też używa się ich by nakierować pełen algorytm na rozwiązanie optymalne, co w rezultacie skróci czas wykonania algorytmu.

Algorytmy heurystyczne możemy podzielić ze względu na sposób w jaki generowane są nowe rozwiązania.

- Algorytmy probabilistyczne - wykorzystują czynnik losowości, często kolejne rozwiązanie wybierane jest losowo z określonej puli. Co może doprowadzić do różnych wyników końcowych otrzymanych z kolejnych uruchomień algorytmu.

- Algorytmy deterministyczne - nie zawierają czynnika losowego. Otrzymywane rozwiązanie zawsze powinno być takie same, przy każdym uruchomieniu algorytmu o takich samych parametrach.

W niektórych algorytmach wykorzystane są dwie heurystyki, nadrzędna i podrzędna. Pierwsza z nich steruje i wspiera działanie drugiej. Takie zjawiska nazywane są przez niektórych metaheurystykami. [1] Inna definicja metaheurystyki to „procesy iteracje działające zgodnie z klasyczną metodą heurystyczną, wspomagane inteligentnie przez różne koncepcje eksplorowania i eksploatowania przestrzeni rozwiązań z użyciem technik uczących. Wspomaganie to ustrukturalnia informacje w celu sprawnego znalezienia rozwiązań bliskich optymalnemu.” [3] Jednak po raz pierwszy termin ten został użyty przez Freda Glovera w 1986 roku jako określenie algorytmów, które nie rozwiązują bezpośrednio żadnego problemu, lecz określają w jaki sposób budować algorytmy podrzędne w celu uzyskania rozwiązania. [2]

## 3.2 Tabu.

Przykładem algorytmu metaheurystycznego jest algorytm wyszukiwania z tabu. Algorytm ten swoje początki ma w 1977 roku kiedy to Fred Glover przedstawił pracę na temat wykorzystania pamięci krótkotrwałej i długotrwałej w przeszukiwaniu lokalnym. Pamięć krótkotrwała służyła do zapamiętywania ostatnich ruchów algorytmu i była nadpisywana przez kolejne jego iteracje, natomiast pamięć długotrwała miała na celu zapamiętać najbardziej atrakcyjne elementy przestrzeni poszukiwań. To właśnie w oparciu o tą zasadę, Glover zaproponował w 1986 roku, algorytm *Tabu Search*. Jest

on uznawany za autora algorytmu mimo tego, że w tym samym roku Michael Hansen opublikował pracę opisującą bardzo podobną heurystykę. Na przestrzeni lat algorytm został ulepszony i aktualnie dostępnych jest wiele jego różnych modyfikacji (np. *Probabilistic Tabu Search* lub *Reactive Tabu Search*).

### 3.2.1 Zasada działania algorytmu.

Wyszukiwanie z tabu to metaheurystyka służąca do rozwiązywania problemów optymalizacji. Algorytm ten opiera się na iteracyjnym przeszukiwaniu przestrzeni rozwiązań, z użyciem tzw. sąsiedztwa oraz na zapamiętywaniu ostatnio wykonanych ruchów w celu uniknięcia powtarzalności. Wywodzi się on bezpośrednio z metody przeszukiwania lokalnego, jednak jest od niej zdecydowanie skuteczniejszy dzięki mechanizmowi wychodzenia z minimów lokalnych. Mechanizm ten pozwala na pogorszenie aktualnego wyniku w celu uzyskania wyniku jeszcze lepszego. Możliwe jest to dzięki przestrzeni tabu czyli listy ruchów, które algorytm już wykonał, co zabezpiecza tę metodę przed powrotem w obszary niedawno przeszukane. Obecność ruchów na liście tabu jest tymczasowa, co w konsekwencji blokuje dany ruch przez określoną ilość iteracji. Możliwe jest złamanie tej zasady, ale tylko wtedy gdy ruch spełnia tzw. kryterium aspiracji. Warunkiem końcowym tej metody najczęściej jest określona liczba iteracji algorytmu lub osiągnięcie satysfakcjonującego rozwiązania. Możliwe jest również monitorowanie aktualnego wyniku i, jeżeli nie ulega on poprawie przez określoną ilość iteracji, zatrzymanie algorytmu.

### 3.2.2 Sąsiedztwo.

Najważniejszym czynnikiem od którego zależy sukces końcowy metody jest poprawne zdefiniowanie sąsiedztwa, które będzie przeszukiwane w danej iteracji. Do sąsiedztwa powinny należeć elementy różniące się w sposób nieznaczny, jednak wystarczający, by umożliwić algorytmowi przejście w każdy obszar przestrzeni rozwiązań. Sposób w jaki definiowane jest sąsiedztwo zależy bezpośrednio od danego problemu i typu jego rozwiązań. Rozwiązaniami mogą być np. wektory binarne, wektory liczb rzeczywistych czy jakiekolwiek permutacje zbiorów. Jeżeli na przykład rozwiązaniem będzie permutacja jakiegoś zbioru $n$ elementowego to sąsiedztwem możemy określić jedno z trzech typów przejść między permutacjami:

- wstaw(x, y) - wstawienie elementu y na pozycję x. (Permutacje z powtórzeniami)

- zamień(x, y) -zamienienie elementów na pozycjach x i y.

- odwróć(x, y) - odwrócenie kolejności występowania elementów pomiędzy indeksami x i y.

Sąsiedztwem danego ciągu będzie więc każdy inny ciąg poddany jednej, wybranej na początku, metodzie zamiany dla wszystkich możliwych parametrów (x, y). Dzięki tak zdefiniowanemu sąsiedztwu możliwe jest łatwe zidentyfikowanie przejścia poprzez parę indeksów (x, y). To właśnie ta para będzie zapisana na liście Tabu, a ruch ten zablokowany przez następne iteracje.

Może się jednak okazać, że generowane sąsiedztwa są zbyt duże by każdorazowo przeszukiwać je w całości. Stosowane jest wtedy zawężanie sąsiedztwa za pomocą określonej metody. Jedną z nich jest metoda losowego doboru sąsiedztwa. Wprowadza to element probabilistyczny do algorytmu co zmniejsza prawdopodobieństwo powstania niepożądanych cykli, jednak przy takim podejściu możemy pominąć obszary przestrzeni rozwiązań, w których znajduje się to optymalne.

### 3.2.3 Kryteria aspiracji.

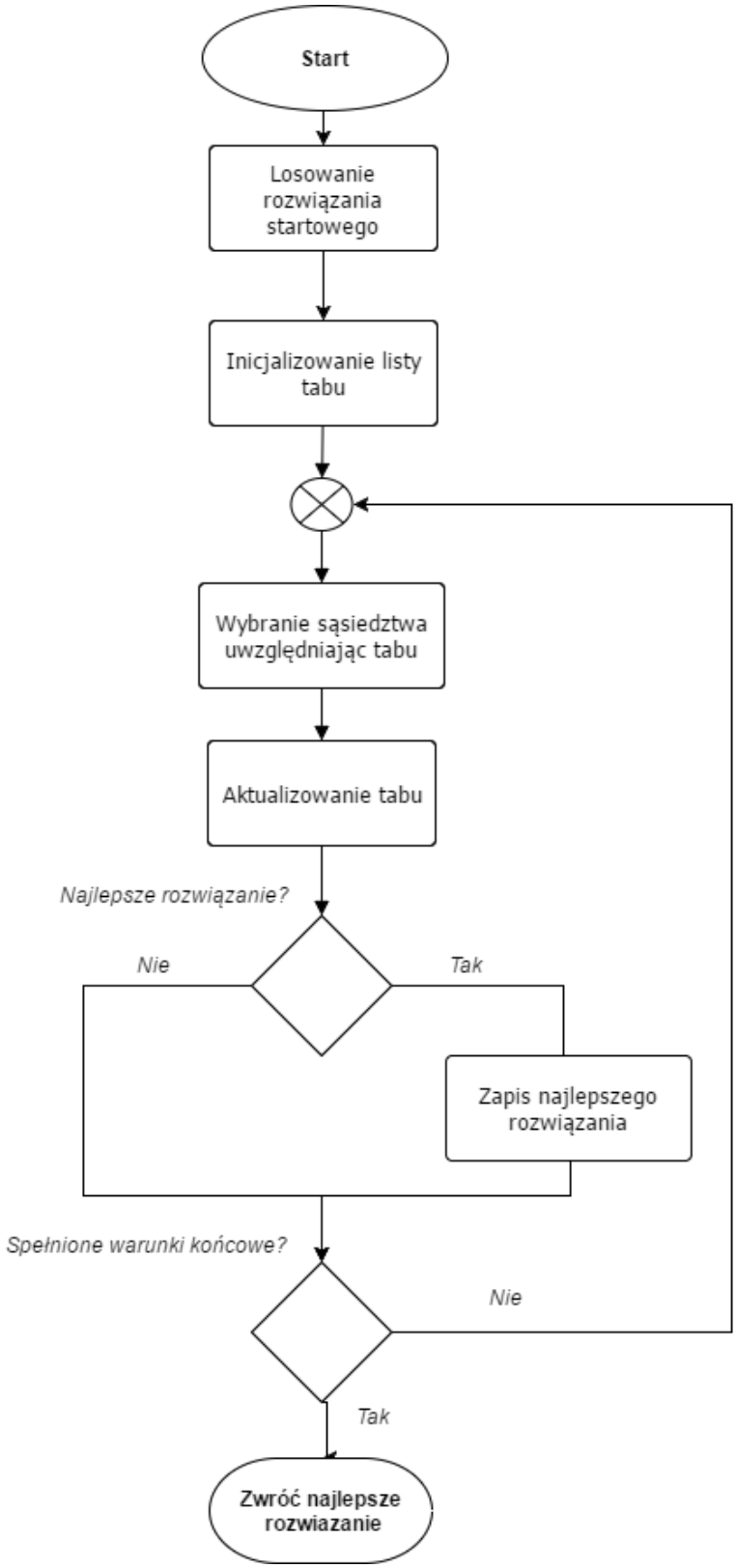
Może się zdarzyć, że zabranianie pewnych ruchów doprowadzi do przestojów lub całkowicie zablokuje kolejny ruch (sytuacja, gdy wszystkie dostępne przejścia są na liście tabu). Jest to możliwe ponieważ algorytm przechowuje tylko transformacje, a więc atrybuty pewnych rozwiązań a nie całe rozwiązania. Kryterium aspiracji umożliwia wyjście z takiej sytuacji. Spełnienie takiego kryterium pozwala na złamanie zakazu tabu, czyli wykonanie ruchu, który znajduje się na liście ostatnio użytych. Najpopularniejszym i najprostszym kryterium aspiracji jest uzyskanie najlepszego znanego wyniku. Musi być ono lepsze od aktualnie najlepszego w celu uniknięcia zapętleń. Jednak większość kryteriów aspiracji jest dużo bardziej skomplikowana i opiera się na wyspecjalizowanych metodach przewidywania możliwości powstania cyklu po wykonaniu określonego ruchu.

### 3.2.4 Dywersyfikacji i intensyfikacja.

Ważnym aspektem algorytmu jest pamięć długoterminowa. Służy ona do przechowywania informacji o przeszłych iteracjach i do budowania statystyk. Dzięki tym statystykom możemy modyfikować strategię poszukiwania. Głównym celem takich modyfikacji może być spełnienie jednego z dwóch kryteriów:

- intensyfikacja - jeżeli według statystyki w danym obszarze znajduje się dużo dobrych rozwiązań, algorytm zagęści obszar poszukiwań. Dzięki temu istnieje szansa na znalezienie jeszcze lepszego wyniku w danym obszarze.
- dywersyfikacja - jest to rozproszenie obszaru poszukiwań. Najczęściej stosowanym rozwiązaniem jest nakładanie kary na ruchy, które powtarzają się w perspektywie dłuższego czasu. Efektem tego jest przeniesienie algorytmu w inne rejony poszukiwań co zmniejsza szanse na pominięcie najlepszych rozwiązań.

Wykorzystanie intensyfikacji i dywersyfikacji w znaczniej mierze poprawia efektywność algorytmu, dlatego kryteria te są wykorzystywane w większości nowych implementacji algorytmu *Tabu Search*.




Rysunek 3.1: Schemat algorytmu Tabu Search

# Bibliografia

[1] A. Debudaj-Grabysz, J. Widuch, and S. Deorowicz. *Algorytmy i struktury danych. Wybór zaawansowanych metod.* Wydawnictwo Politechniki śląskiej, Gliwice, 2012.

[2] Fred Glover. *Future Paths for Integer Programming and Links to Artificial Intelligence.* Oxford: Elsevier, 1986.

[3] I.H. Osman and J.P. Kelly. *Meta-heuristics: An Overview.* Kluwer Academic Publishers, 1996.

# Dodatek A

# Symbole przyjęte w pracy

Jeśli w tekście nie wykazano inaczej, stosowane symbole należy rozumieć jako:

$f(x, y)$ - jasność piksela o współrzędnych $(x, y)$ w obrazie wejściowym,

$g(x, y)$ - jasność piksela o współrzędnych $(x, y)$ w obrazie wynikowym,

$t$, $t_i$ - wartości progowe,

$G$ - liczba poziomów szarości obrazu; $G = 256$,

$P(i, j)$ - macierz GLCM,

$M(i, j)$ - maska przetwarzania.

# Dodatek B

# Inny, przykładowy dodatek