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 Ograniczenia planu zajęć.

Problem układania planu zajęć można sprowadzić do przyporządkowania określonym wydarzeniom odpowiednich okien czasowych i zasobów, tak by spełnić początkowe założenia. Dla planu zajęć wydarzeniami będą poszczególne lekcje odbywające się w ciągu jednego tygodnia, dostępnymi oknami czasowymi będą godziny w których mogą odbywać się te zajęcia (np. od poniedziałku do piątku w godzinach między 8:00 a 18:00), natomiast zasobami będzie cała reszta danych przypisana do zajęć: grupy uczniów, prowadzący zajęcia, sala.

Układanie planu zajęć wymaga przestrzegania odpowiednich ograniczeń. Nie można w prosty sposób wypełnić okien czasowych według jakiejś przyjętej kolejności ponieważ najprawdopodobniej pojawią się konflikty i plan zajęć nie będzie możliwy do zrealizowania. By tego uniknąć trzeba zachować tzw. ograniczenia twarde, czyli takie, które kategorycznie muszą zostać spełnione by plan mógł być możliwy do zrealizowania. Istnieją również ograniczenia miękkie, które określają jakość planu końcowego. Ograniczenia te nie muszą zostać spełnione, ale jeżeli jest to możliwe, algorytm układający plan zajęć bierze je pod uwagę.

Przykłady ograniczeń:

- Twarde:
  - Niepodzielność zasobów
  - Przypisanie wszystkich zajęć do okien czasowych

- Miękkie:
  - Brak okienek dla studentów
  - Odpowiednie przerwy między zajęciami
  - Jak najmniej dni roboczych dla prowadzących
  - Brak bloków zajęć danego typu

Niepodzielność zasobów oznacza, że jeden nauczyciel nie może być w dwóch miejscach jednocześnie, więc może prowadzić tylko jedne zajęcia w danym oknie czasowym, w jednej sali nie mogą się odbywać w tym samym czasie różne zajęcia, a dla danej grupy lekcje nie mają prawa się nakładać. Naruszenie tego ograniczenia było by fizycznie nie możliwe. Obowiązek przypisania zajęć do jakichś okien czasowych oznacza, że na planie końcowym muszą się znaleźć wszystkie lekcje przewidziane w siatce zajęć dla danej grupy. Jest to podstawowe założenie bez którego układanie planu traci swój sens. Często jednak dla konkretnej siatki zajęć spełnienie już tych warunków jest niemożliwe. Wynika to najczęściej z za małej liczby zasobów (za mało nauczycieli mogących prowadzić jeden przedmiot, za mało sal). Algorytmy szukają wtedy planu z najmniejszą liczbą konfliktów, a pozostałe przypadki uzupełniamy już ręcznie szukając jakichś kompromisów (dołożenie okna czasowego, zatrudnienie dodatkowego nauczyciela lub pomieszczenie dwóch mniejszych grup w jednej sali).

Ograniczenia miękkie to tak naprawdę życzenia użytkowników co do tego jak dany plan ma wyglądać. Wpływają one na końcową ocenę planu zajęć i mogą mieć określone wagi w zależności od ich istotności. Niektóre plany zajęć układane są pod uczniów (mała liczba okienek, zajęcia równomiernie rozłożone, brak bloków zajęć danego typu, odpowiednia przerwa między zajęciami), a niektóre pod prowadzących (zajęcia skumulowane w dwa dni tygodnia by umożliwić prace na innej uczelni). Wszystko zależy od osoby konfigurującej algorytm i definiującej funkcję oceny planu zajęć.

Funkcja oceny planu zajęć najczęściej polega na nakładaniu punktów karnych za złamanie określonych ograniczeń. Każde ograniczenie ma ustawioną własną wartość punktów karnych przy czym ograniczenia twarde powinny mieć dużo większą wagę od ograniczeń miękkich. Im więcej punktów karnych tym plan jest gorszy. To właśnie na podstawie funkcji oceny większość algorytmów wyszukuje optymalny plan zajęć.

2.2 Rozmiar problemu.

By dobrze zrozumieć potrzebę używania algorytmów heurystycznych przy wyszukiwaniu optymalnego planu zajęć warto zobrazować skalę problemu. W tym celu przedstawiony zostanie przykład planu zajęć dla amerykańskiej szkoły średniego rozmiaru. W tym przykładzie, każde ze zdarzeń ma już przyporządkowane zasoby.

- Okna czasowe - 40
- Nauczyciele - 27
- Sale - 30
- Grupy uczniów - 24
- Zdarzenia - 832
- Ograniczenia - 4

Mamy dostępne 40 okien czasowych, co rozłożone na 5 dni tygodnia daje 8 godzin zajęć dziennie. Jednocześnie odbywać się mogą 24 zajęcia. Liczba ta wynika z ograniczenia niepodzielności zasobów i jest minimalną liczbą z pośród ilości nauczycieli, sal i grup uczniów. Maksymalna liczba zdarzeń, które mogą się odbywać w danym tygodniu wynosi więc:

$$40 * 24 = 960$$

$$960 > 832$$

Dostępna liczba zdarzeń jest większa od liczby zdarzeń zawartych w siatce zajęć podanego przykładu, więc teoretycznie realizacja tego planu zajęć jest możliwa.

Na jedną grupę przypada około 35 zajęć, które muszą być rozłożone na 40 okien czasowych. Uwzględniając możliwość pojawienia się okienek w czasie zajęć mamy więc 40 wyrazowy ciąg zdarzeń czyli permutację. Liczba możliwości w jaki sposób można te zajęcia rozłożyć, uwzględniając możliwość okienek, wynosi więc:

$$40! = 815915283247897734345611269596115894272000000000$$

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

Przedstawione dane mają z góry przyporządkowane zasoby do zdarzeń, więc każde zajęcia mają przyporządkowaną grupę, nauczyciela i salę. Dzięki temu obszar poszukiwań algorytmu 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.

Takie podejście, w którym zasoby z góry są przypisane do zdarzeń, umożliwia szybszą pracę 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.

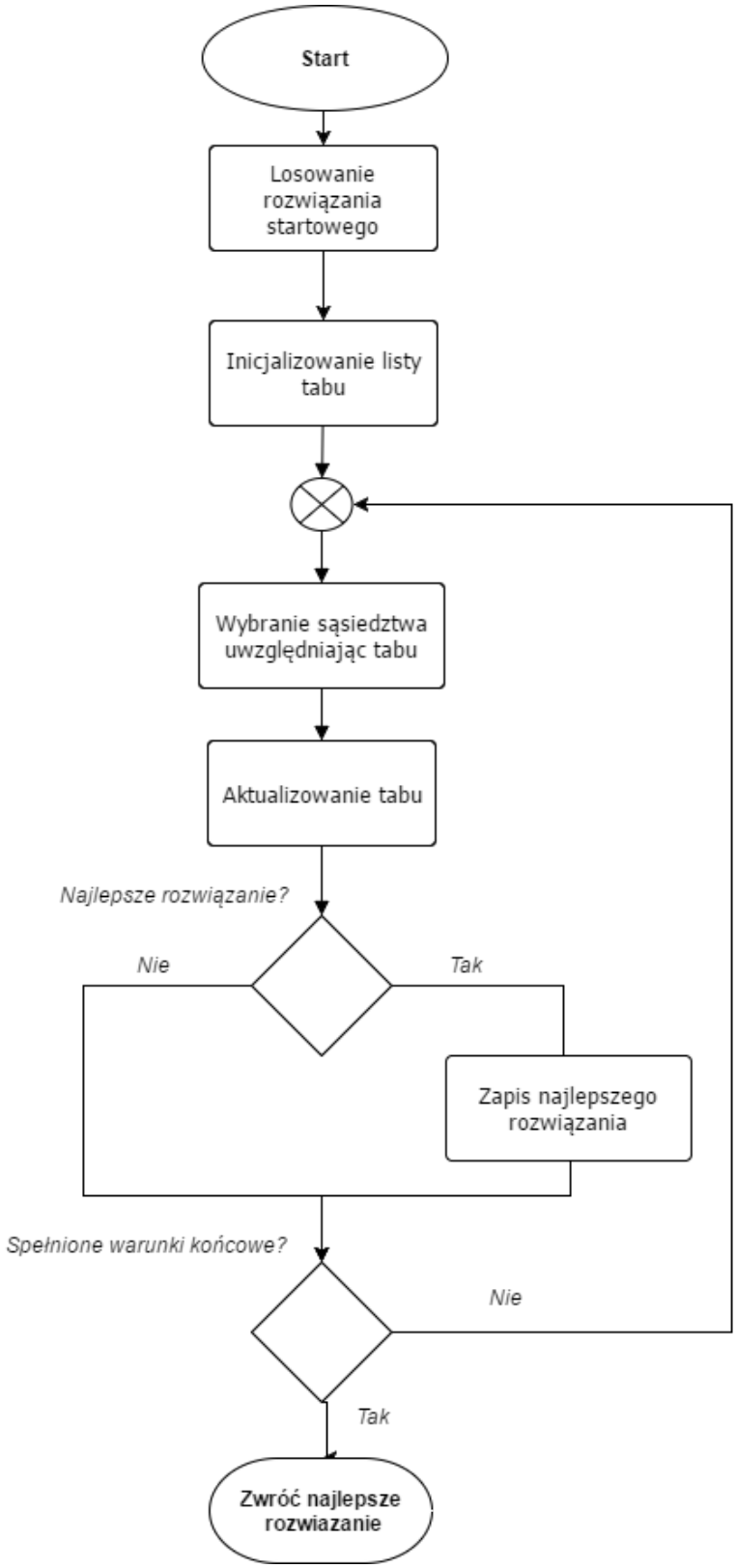
Może się zdarzyć, że zabranianie pewnych ruchów doprowadzi do przestojów lub całkowicie zablokuje kolejny ruch (sytuacja, gdy wszystkie dostępne przejścia są na liście tabu). Jest to możliwe ponieważ algorytm przechowuje tylko transformacje, a więc atrybuty pewnych rozwiązań a nie całe rozwiązania. Kryterium 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