Sztuczna Inteligencja i Inżynieria Wiedzy
laboratorium

Ćwiczenie 2. Problem spełniania ograniczeń

opracowanie: A. Filipiak, A. Janz

Cel ćwiczenia

Zapoznanie się z podstawowymi algorytmami stosowanymi do rozwiązywania problemów spełniania ograniczeń (ang. *Constraint Satisfaction Problem*, CSP), poprzez własnoręczną implementację i zbadanie ich właściwości.

Realizacja ćwiczenia

- Zapoznanie się z działaniem algorytmów: sprawdzenia wprzód (ang. *forward checking*), przeszukiwania przyrostowego z powracaniem (ang. *backtracking*).
- Zapoznanie się z problemami: N-hetmanów, kwadrat łaciński, kolorowanie grafu
- Sformułowanie dwóch wybranych problemów jako problemów spełniania ograniczeń, poprzez podanie zmiennych, ich dziedzin i ograniczeń
- Implementacja algorytmów sprawdzenia wprzód oraz przeszukiwania przyrostowego z powracaniem
- Porównanie nakładu obliczeniowego obu metod z użyciem dwóch miar: czasu działania procedury oraz liczby wywołań rekurencyjnych i powrotów
- Zaproponowanie i implementacja heurystyk: wyboru kolejnej zmiennej do wartościowania, wyboru kolejnej wartości do przypisania
- Zbadanie wpływu zastosowanych heurystyk na nakład obliczeniowy
- Prezentacja najciekawszych (zdaniem studenta) wyników
- Dyskusja otrzymanych wyników
- Przygotowanie sprawozdania zawierającego powyższe punkty

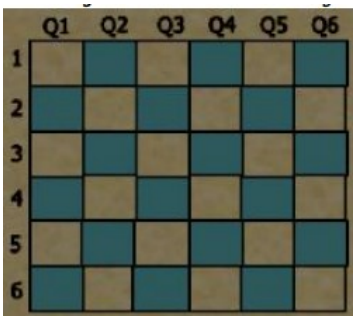
Dwa z wybranych problemów z niżej opisanych należy zdefiniować jako problem CSP i rozwiązać wykorzystując algorytmy sprawdzenia wprzód oraz przeszukiwania przyrostowego z powracaniem dla różnych wartości *N*, gdzie *N* definiuje wielkość wybranego problemu. Należy zaimplementować metody z użyciem wybranego języka obiektowego (C++, Java, Python, C#) i przeprowadzić badania z wykorzystaniem napisanego własnoręcznie programu.

W badaniach należy porównać specyfikę przetwarzania obu metod w zależności od wartości parametru *N*, pod kątem czasu przetwarzania i liczby wykonywanych operacji (wywołań rekurencyjnych i powrotów), dla znajdowania pierwszego jak i wszystkich istniejących rozwiązań danego problemu o zadanej wielkości. Dla jednego z problemów wymagana jest dokładna analiza czasu przetwarzania z wykorzystaniem profilera oraz dokonanie niezbędnych optymalizacji – wskazanie najdroższych czasowo operacji, wprowadzenie zmian oraz określenie ich wpływu na czas przetwarzania.

Należy także zaproponować dla jednego z problemów dwie heurystyki: wyboru kolejnej zmiennej do wartościowania i wyboru wartości do przypisania oraz zbadać ich wpływ na czas przetwarzania i liczbę wykonywanych operacji.

Problem 1. N-hetmanów

Dana jest tablica o wymiarach *N*x*N*, dla *N*=6 tablica wygląda następująco:



Należy ustawić *N* hetmanów na tablicy w ten sposób, by każdy z hetmanów nie atakował innego hetmana. Hetman atakuje inną figurę, jeżeli figura znajduje się na przekątnej, bądź na prostej, na której znajduje się hetman. Dokładny opis problemu (dla *N*=8) można znaleźć pod adresem:
http://pl.wikipedia.org/wiki/Problem_ośmiu_hetmanów

Problem 2. Kwadrat łaciński

Kwadrat łaciński rzędu *n* jest to macierz kwadratowa *L: N*x*N* o elementach pochodzących z *n*-elementowego zbioru liczb *S*: {1, 2, …, *n*}, dla której zachodzi własność, taka że w każdym wierszu oraz kolumnie takiej macierzy dany element występuje wyłącznie raz. Kwadrat łaciński można również przedstawić w formie kolorowej kraty o wymiarze *N*x*N*, której kolory odpowiadają elementom macierzy spełniając przy tym narzucone ograniczenia. Zadanie sprowadza się do rozwiązania problemu wyznaczenia kwadratu łacińskiego rzędu *N*: {2, 3, …, *k*}, rozpoczynając od macierzy zerowej (pustej kraty):

Macierz zerowa

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Kwadrat łaciński

$$\begin{bmatrix} 1 & 4 & 2 & 5 & 3 & 6 \\ 6 & 3 & 4 & 2 & 5 & 1 \\ 2 & 1 & 5 & 6 & 4 & 3 \\ 4 & 2 & 6 & 3 & 1 & 5 \\ 3 & 5 & 1 & 4 & 6 & 2 \\ 5 & 6 & 3 & 1 & 2 & 4 \end{bmatrix}$$

Kwadrat łaciński i jego własności: https://en.wikipedia.org/wiki/Latin_square
Liczba wszystkich kwadratów łacińskich rzędu *N:* https://oeis.org/A002860

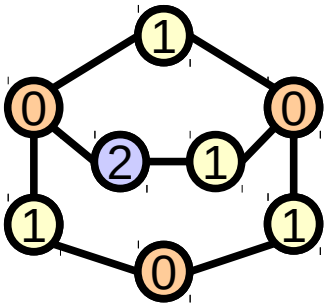
Problem 3. Kolorowanie grafu L(2,1)

Klasyczne zadanie kolorowania grafu (węzłów) polega na takim przypisaniu kolorów do jego węzłów, aby żadne dwa z połączonych krawędzią nie miały tej samej barwy. Kolorowanie L(2,1) wprowadza dodatkowe warunki (w interpretacji koloru jako etykiety numerycznej):

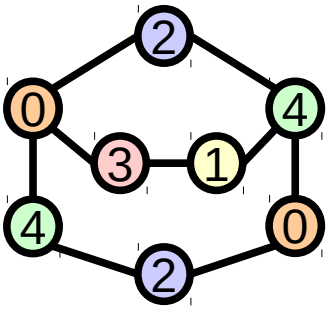
1. Wartości kolorów sąsiadujących ze sobą węzłów muszą różnić się o co najmniej 2
2. Wartości kolorów węzłów odległych o dwa przejścia krawędzi muszą różnić się między sobą o co najmniej 1

Opis problemu:
https://en.wikipedia.org/wiki/Graph_coloring
https://en.wikipedia.org/wiki/L(2,1)-coloring

Kolorowanie grafu

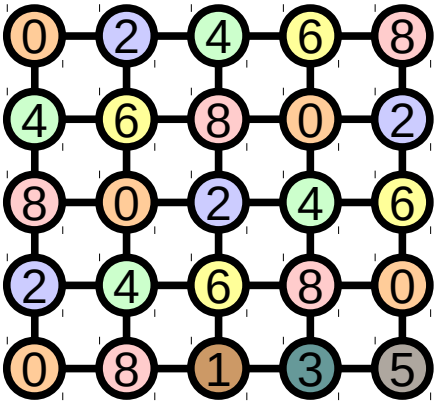


Kolorowanie grafu L(2,1)



Zadanie polegać będzie na znalezieniu minimalnego zbioru {1,2, …, *M*}, który pozwala na kolorowanie L(2,1) grafu w postaci dwuwymiarowej kraty o wymiarach *N*x*N*. Ze względu na specyficzny charakter grafu, rozwiązanie można zinterpretować klasycznie w postaci grafu (węzły i łączące je krawędzie) lub jako kolorowanie pól tablicy, tak aby różnica wartości pomiędzy polami sąsiadującymi w pionie lub poziomie wynosiła co najmniej 2, a różnica pomiędzy polami sąsiadującymi po skosie wynosiła co najmniej 1.

Krata w postaci grafu



Krata w postaci tablicy

0	**2**	**4**	**6**	**8**
4	**6**	**8**	**0**	**2**
8	**0**	**2**	**4**	**6**
2	**4**	**6**	**8**	**0**
0	**8**	**1**	**3**	**5**

Problem zaproponowany przez studenta

Zamiast problemu wyznaczania kwadratu łacińskiego, można zaproponować (konsultując uprzednio wybór z prowadzącym i uzyskują jego akceptację) łamigłówkę logiczną, możliwą do zdefiniowania jako problem CSP np. N-hetmanów, kakuro, kryptarytmy, etc. Wybrane zagadnienie musi umożliwiać sterowanie parametrem określającym rozmiar!

Ocena ćwiczenia

1pkt	Sformułowanie wybranych problemów jako problemów CSP
2pkt	Implementacja algorytmów sprawdzenia wprzód i przeszukiwania przyrostowego z powracaniem dla obu problemów
4pkt	Przeprowadzenie badania czasów przetwarzania zaimplementowanych metod oraz nakładu obliczeniowego w zależności od parametru *N** dla obu problemów
1pkt	Analiza i optymalizacja kodu z wykorzystaniem profilera
2pkt	Zaproponowanie heurystyk wyboru kolejnego elementu z dziedziny i kolejnej zmiennej oraz zbadanie ich wpływu na działanie metod

* w przypadku samodzielnie zaproponowanego zagadnienia, badamy wpływ parametru określającego wielkość problemu

Pytania pomocnicze:

1. Który z algorytmów (Backtracking, Forward Checking) działa szybciej? W jakich warunkach? Z czego wynikają różnice w czasach?
2. Jakie są przewidywania co do czasu działania obu algorytmów dla N większych niż przetestowane?
3. Który z algorytmów wykonuje mniejszą liczbę kroków? Jak się ona przekłada na czas działania algorytmów?
4. Czy heurystyki przyspieszają czas działania algorytmów? Czy zmniejszają liczbę wykonywanych kroków?

Źródła

1. Notatki z wykładu i materiały do ćwiczeń.
2. Wojna A., Przeszukiwanie przestrzeni stanów — problemy z więzami http://www.mimuw.edu.pl/~awojna/SID/wyklady/przesz_z_wiezami.pdf