GIS Sprawozdanie II

Krzysztof Burliński, Piotr Kałamucki

21 stycznia 2014

Spis treści

1 Treść zadania **2**

2 Algorytm **2**

3 Struktury danych **3**

4 Projekty testów **4**

5 Złożoność **4**

6 Wejście/wyjście, warunki stopu **4**

7 Testy poprawnościowe **5**

8 Parametry heurystyki **5**

9 Wnioski **6**

1 Treść zadania

Dany jest graf G=(V,E). Gracze na zmianę wybierają wierzchołek w grafie, który następnie jest usuwany razem ze wszystkimi przyległymi wierzchołkami. Gracz wygrywa wtedy i tylko wtedy gdy jego przeciwnik nie może wykonać ruchu, czyli nie ma już żadnego wierzchołka do wyboru.

2 Algorytm

Poszukując algorytmu dla zadanego projektu, zaczęlismy od rozwiązań poprawnych. Nie mogąc takiego znaleźć zacząliśmy, tak jak jest w temacie projektu, poszukiwać odpowiedniej heurestyki dla tego problemu. Generalized Kayles jest uogólnieniem gry Kayles z tablicy na graf (z racji, że nie było to sprecyzowane w treści, przyjęliśmy nieskierowany).

Strategia dla gry Kayles została już opracowana, gdyż gra ta należy do zbioru gier symetrycznych. Gracz rozpoczynający może zapewnić sobie zwycięstwo dzieląc listę na dwie części, a potem powielając ruchy przeciwnika. W ten sposób, pierwszym gracz, który nie będzie mógł wykonać ruchu staje się gracz, który wykonuje ruch jako drugi.

W przypadku Generalized Kayles sytuacja nie jest niestety bardzo podobna to oryginału. Głównym powodem jest to, że pierwszy gracz nie może po prostu (poza rzadkimi przypadkami drzew symetrycznych) dokonać podziału grafu na dwie części za pomocą jednego ruchu.

Po analizie zasad gry udało nam się zauważyć, że gracz może być pewny zwycięstwa w chwili, gdy w grze pozostaje nieparzysta liczba grafów posiadających wierzchołek sąsiadujący z wszystkimi innymi wierzchołkami. Jest do jednak dość rzadka sytuacja, ponadto, zasady tej nie udało nam się uogólnić na grafy nie posiadające tej własności.

Kolejną dokonaną obserwacją jest fakt, że jeśli istnieje wierzchołek, po wybraniu którego powstaje parzysta liczba izomorficznych grafów, gracz znajdujący się w tej sytuacji również może być pewny zwycięstwa, korzystając z teorii wygrywającej gier symetrycznych. Niestety, nie udało nam się odkryć uogólnienia tej zasady na pozostałe sytuacje.

Wybierając heurestykę dla postawionego problemu postanowiliśmy dążyć do postawienia gracza w sytuacji wygrywającej, znanej z gier symetrycznych. W tym celu stworzyliśmy funkcję heurestyczną, która każdemu wierzchołkowi będzie przypisywała wartość, tym większą, im bardziej niepodobny dany wierzchołek jest do pozostałych w grafie. Podobnieństwo obliczamy w następujący sposób: dla każdego wierzchołka obliczamy trzy zmienne:

- x_a - największa odległość od tego wierzchołka do dowolnego innego w tym grafie.
- x_b - liczba wierzchołków sąsiadujących z tym wierzchołkiem

- x_c - liczba wierzchołków znajdujących się w odległości 2 od danego wierzchołka.

Z tych trzech współczynników wartość heurestyki, przy użyciu arbitralnych wag wa, wb, wc, wyznacza się jako odległość do najbliższego wierzchołka w przestrzeni tworzonej przez ten wektor odleglości. Wierzchołkiem do usunięcia przez komputerowego gracza jest wierzchołek, którego wartość heurestyki jest największa, czyli jego najbliższy sąsiad jest najdalej ze wszystkich rozpatrywanych wierzchołków.

Oto wzór na odległość między punktami x i y:

$$d(x_a, x_b, x_c, y_a, y_b, y_c) = (x_a - y_a)^2 + (x_b - y_b)^2 + (x_c - y_c)^2$$

Proponowana heurestyka będzie starała się promować wierzchołki, które są zupełnie inne reszty.

3 Struktury danych

Implementacja algorytmu opiera się na klasie Graph, która jest reprezentacją grafu w postaci list sąsiedztwa:

```
public class Graph {
    private Map<Vertex, Vertex> vertices;
/* konstruktor, metody */
}
```

Klasa Vertex:

```
public class Vertex implements Serializable {
        public enum Color {
                VISITED, UNVISITED
        }

        private final String name;
        private Color color = Color.UNVISITED;
        private int distance = 0;
        private Set<Vertex> neighbours;
        private float farthest = 0;
        private float proximity = 0;
        private float extended = 0;
/* konstruktor, metody */
}
```

Jak widać, informacje przechowywane są w formie mapy wierchołek - wierzchołek. Każdy wierzchołek jest opisany ciągiem znaków, najczęściej jest to jeden znak alfanumeryczny. Nasza implementacja grafu umożliwia następujące operacje:

- dodanie wierzchołka - public void addVertex(String vertex)
- dodanie krawędzi - public void addEdge(String vertexFrom, String vertexTo)
- usunięcie wierchołka zgodnie z regułami gry - public void kaylesRemove(String vertex)
- odczytanie zbioru wierchołków - public Set getVertices()
- odczytanie sąsiadów wierzchołka - public Set getNeighbours(String vertex)

4 Projekty testów

Testy heurystyki będą przeprowadzane w trakcie gry przeciwko graczowi ludzkiemu. Do tego celu użyjemy 3 predefiniowanych grafów o rosnącym stopniu komplikacji - coraz większej liczbie wierzchołków. Na pierwszym grafie będzie można łatwo zauważyć jakie ruchy zagwarantują wygraną, na drugim grafie będzie to trudniejsze, a na ostatnim - znacznie trudniejsze. Ostatni graf będzie zawierał kilkadziesiąt wierzchołków. Drugim etapem testów (jako że nasza heurystyka umożliwia parametryzację) jest uruchomienie gier komputer - komputer. Będą odbywały się one na znacznie większych losowych grafach (kilkaset, kilka tysięcy wierzchołków) i pokażą czy istnieją wartości parametrów, które dają lepsze wyniki.

5 Złożoność

Złożoność algorytmu jest zależna od jej składowych. Pierwsza z nich, czyli liczba sąsiadów wierzchołka jest w pesymistycznym wypadku liniowa. Druga z nich, liczba wierzchołków w odległości 2 od danego wierzchołka, może być policzona za pomocą algorytmu przeszukiwania wszerz, stąd jego złożoność jest w najgorszym wypadku liniowa. Trzecią składową liczy się w ten sam sposób, więc złożoność również jest liniowa. Wartości te należy policzyć dla każdego wierzchołka grafu, za każdym ruchem gracza.

Stąd, maksymalna liczba przejść zbioru wierzchołków to:

$$N * (N + N + N)$$

Co oznacza kwadratową złożoność obliczeniową.

6 Wejście/wyjście, warunki stopu

Z racji, że algorytm ten będzie służył do wyboru ruchów grze, wejście będzie bezpośrednio powiązane z początkowym stanem gry. Algorytm będzie

akceptował dwa rodzaje wejść. Pierwszym będzie graf, podany za pomocą liczby wierzchołków i listy krawędzi pomiędzy tymi wierzchołkami. Drugim sposobem jest podanie liczby wierzchołków i krawędzi. Algorytm wtedy sam stworzy graf, losowo rozmieszczając krawędzie między wierzchołkami.

Algorytm nie będzie miał skomplikowanego wyjścia. W przypadku gry między dwoma komputerami będzie po prostu zwracał informację, który z nich wygrał.

Zakończenie gry odbywa się, gry któryś z graczy nie może już wykonać ruchu. W takiej sytuacji wygrywa jego przeciwnik.

7 Testy poprawnościowe

W celu sprawdzenia poprawności algorytmów zaimplementowano testy jednostkowe. Klasa GraphTests zawiera 4 testy:

```
shouldAddNewEdge();
shouldClearGraph();
testDistanceToFarthestVertex();
testGame();
```

Testowana jest nasza implementacja grafu (dodawanie krawędzi, wierzchołków, usuwanie wierzchołków zgodnie z regułami gry) jak i pomocnicze metody na potrzeby heurystyki - obliczanie odległości do najdalszego wierzchołka (pojęcie zbliżone do średnicy grafu). W naszej implementacji grafu obliczenie pozostałych składowych heurystyki jest trywialne - liczba sąsiadów i liczba sąsiadów drugiego stopnia są bardzo łatwo dostępne.

8 Parametry heurystyki

Nasza heurystyka posiada 3 parametry - wagi poszczególnych składowych. Wartości wag znajdują się w zakresie $[0, 1]$ sumując się do 1. Przeprowadzono eksperyment który miał na celu poznanie optymalnych wartości poszczególnych wag. Eksperyment polegał na uruchomieniu 120 gier, w których przeciwko sobie grały heurystyki z różnymi wagami poszczególnych składowych. Grafy na których rozgrywały się poszczególne gry były losowe (losowa liczba wierzchołków i krawędzi, losowa relacja między nimi. Otrzymane wyniki to:

- p_1 0.114583336, (odległość do najdalszego wierzchołka od danego wierzchołka)
- p_2 0.5052083, (liczba sąsiadów drugiego stopnia)
- p_3 0.38020834 (liczba sąsiadów pierwszego stopnia)

9 Wnioski

Analizując optymalne parametry heurystyki widać, że największy wpływ na jej wyniki ma druga składowa - ilość sąsiadów 2 stopnia. Wynika to z faktu, że usuwając wierzchołek z największą ilością sąsiadów 2 stopnia, usunięta zostanie największa część grafu. W końcowej fazie gry, gdy zostało już niewiele wierzchołków, jest to bardzo ważne.

Zauważono również, że wpływ na wynik decyzji podejmowanych na początku gry, gdy jest jeszcze wiele wierzchołków, jest niewielki. Kluczowe decyzje podejmowane są w przypadku, gdy zostało kilka możliwych ruchów do końca rozgrywki.