[POP] Dokumentacja wstępna projektu

Zadanie 13 – karty na prostokątnej planszy o wymiarach $4 \times n$.

Michał Szwejk 331445

Kamil Marszałek 331401

1 Polecenie

W każdej komórce planszy prostokątnej o rozmiarze $4 \times n$ wpisano liczbę całkowitą z_{ij}. Masz do dyspozycji m kart, które musisz rozmieścić na planszy. Poprawny rozkład kart zakłada, że żadna para kart nie może zajmować komórek sąsiadujących w pionie lub poziomie. Twoim zadaniem jest znalezienie takiego rozkładu kart na planszy, aby suma liczb zapisanych w komórkach planszy była jak największa. Nie musisz wykorzystywać wszystkich kart.

2 Reprezentacja rozwiązania

Rozwiązanie jest reprezentowane jako macierz zmiennych binarnych $x_{ij} \in \{0, 1\}$, gdzie i i j odpowiadają wierszowi i kolumnie na planszy. Podejście to można dodatkowo uprościć wykorzystując wektory mask bitowych – każdej kolumnie odpowiada liczba całkowita dodatnia, której reprezentacja binarna odwzorowuje przyjęte wartości.

3 Programowanie dynamiczne

Zagadnienie można rozwiązać wykorzystując programowanie dynamiczne. Jako podproblem definiujemy maksymalizację sumy wartości obecnej kolumny i wartości zakumulowanej wynikającej z rozwiązania poprzednich podproblemów. Dobierając maksymalne lokalne rozwiązanie należy uwzględnić ograniczenie sąsiedztwa (analizujemy dwie kolejne maski, ich iloczyn bitowy musi być równy 0) oraz ograniczenie liczby kart (łączna liczba użytych kart nie może przekraczać m).

Formalnie, niech $DP[i][p][k]$ oznacza maksymalną sumę wartości, jaką można uzyskać rozpatrując kolumny od i do końca planszy, przy założeniu, że w poprzedniej kolumnie użyto maski p, natomiast zostało użyte już k kart. Wówczas:

$$DP[i][p][k] = \begin{cases} 0, & \text{jeśli } i = n \text{ lub } k = 0, \\ \max\limits_{\substack{m \in M \\ m \,\&\, p = 0 \\ |m| \leq k}} \big(W[i][m] + DP[i+1][m][k + |m|]\big), & \text{w przeciwnym razie.} \end{cases}$$

Gdzie M oznacza zbiór wszystkich dopuszczalnych masek kolumn (bez sąsiadujących jedynek), $|m|$ liczbę kart wybranych w masce m, a $W[i][m]$ sumę wartości pól zakrytych przez maskę m w kolumnie i.

Rozwiązanie końcowe stanowi wartość $DP[0][0][m]$, czyli maksymalny możliwy zysk dla całej planszy.

4 Strategia zachłanna

Z planszy wybieramy kolejno komórki, dla których opisująca ją wartość liczbowa z_{ij} jest największa. Zaznaczamy je (dołączamy do rozwiązania) i usuwamy ich sasiądów w pionie i poziomie tak długo, aż ograniczenie na liczbę kart m przestanie być spełnione. Po znalezieniu wstępnego rozwiązania strategią zachłanną dodatkowo je ulepszamy naprawiając lokalnie regiony o wymiarach $k \times 4$ wykorzystując programowanie dynamiczne. Takie podejście pozwoli nam naprawić miejsca, w których suma wartości sąsiadów danego pola jest większa niż ono same (mimo iż pojedynczo ma większą wartość). Regiony wybierane są losowo, a ich rozmiar i liczba wszystkich lokalnych poprawek są parametrami algorytmu.

5 Algorytm A*

Algorytm A* łączy podejście programowania dynamicznego z przeszukiwaniem kierowanym heurystyką. Każdy stan opisuje częściowe rozwiązanie (aktualną kolumnę, maskę bitową i liczbę użytych kart), a jego ocena wyrażona jest wzorem:

$$f(s) = g(s) + h(s),$$

gdzie $g(s)$ oznacza uzyskany dotąd zysk, a $h(s)$ szacuje maksymalny możliwy zysk w dalszej części planszy.

Heurystyka $h(s)$ obliczana jest za pomocą *blokowego programowania dynamicznego* — plansza dzielona jest na bloki (np. po 10 kolumn), dla których lokalnie wyznaczane są maksymalne wartości zgodne z ograniczeniami sąsiedztwa i liczby kart. Poza bieżącym blokiem wartości heurystyki są rozszerzane (*propagowane*) o globalną sumę największych dodatnich elementów planszy, co gwarantuje, że $h(s)$ jest dopuszczalna i monotoniczna. Dzięki temu A* analizuje znacznie mniej stanów niż pełne DP, zachowując poprawność i optymalność wyniku.

6 Planowane eksperymenty

W eksperymentach porównamy jakość i czas działania zaproponowanych algorytmów na zestawie instancji o różnych rozmiarach planszy i liczbie kart. Pola planszy zostaną zainicjowane wartościami z różnych przedziałów (np. $[-100, 100]$, $[-1000, 1000]$) aby ocenić wpływ rozkładu wartości na efektywność algorytmów.