# Algorytmy równoległe dla problemów optymalizacji dyskretnej

Wojciech Wieczorek

Optymalizacja z użyciem klastrów komputerowych

### Plan dzisiejszego wykładu

- Podstawowe pojęcia optymalizacji dyskretnej
- Algorytmy sekwencyjne
- Równoległy algorytm z nawrotami
- Równoległy "Best-First Search"

### Optymalizacja dyskretna

- Metoda wyznaczania najlepszego (optymalnego)
  rozwiązania (poszukiwanie ekstremum funkcji) z punktu
  widzenia określonego kryterium jakości (np. kosztu,
  drogi, wydajności).
- Słowo dyskretny odnosi się do charakteru rozwiązania (jest nim obiekt dyskretny, taki jak: liczba całkowita, skończony przeliczalny zbiór elementów, skończony podgraf, permutacja n-elementowa itd.).

### Definicje

- Problem optymalizacji dyskretnej będziemy definiowali parą (S, f), gdzie S jest skończonym zbiorem dopuszczalnych rozwiązań.
- Funkcja f zwana funkcją kosztu przyporządkowuje każdemu elementowi zbioru S liczbę rzeczywistą.
- Celem jest znalezienie takiego rozwiązania  $X_{opt}$ , że  $f(X_{oot}) \leq f(X)$  dla wszystkich  $X \in S$ .
- W ten sposób można sformułować bardzo wiele praktycznych problemów.

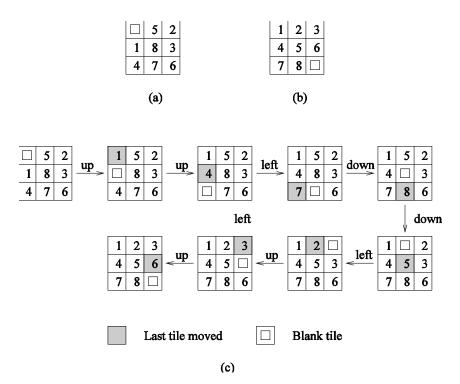
### Optymalizacja dyskretna: przykład 1

- W problemie programowania 0/1 dana jest macierz A wymiaru m x n, wektor b wymiaru m x 1 oraz wektor c wymiaru n x 1.
- Celem jest wyznaczenie wektora x długości n, którego składowe przyjmują wartość 0 lub 1.
- Wektor x musi spełniać warunki Ax ≤ b, a wyrażenie c x powinno przyjąć najmniejszą wartość.

### Optymalizacja dyskretna: przykład 2

- Problem 8-układanki składa się z kratownicy 3×3 zawierającej osiem ponumerowanych przesuwnych kwadratów.
- Jedno pole kratownicy jest puste. Umożliwia ono
  przesunięcie jednego z przyległych kwadratów w to miejsce.
  Wtedy puste pole "przechodzi" na miejsce przesuniętego
  kwadratu.
- Celem układanki jest doprowadzenie wszystkich kwadratów na swoje miejsce za pomocą minimalnej liczby przesunięć.

### Przykład 8-układanki



(a) Pozycja początkowa; (b) pozycja docelowa; (c) ciąg przesunięć prowadzących od (a) do (b).

### Optymalizacja dyskretna: podstawy

- Przestrzeń dopuszczalnych rozwiązań S jest zazwyczaj bardzo duża.
- Z tego powodu problem optymalizacji dyskretnej warto zdefiniować jako wyszukiwanie ścieżki o minimalnym koszcie, łączącej wierzchołek początkowy z jednym z wierzchołków docelowych w grafie.
- Każdy element X w S może być reprezentowany przez taką ścieżkę.
- Graf ten nazywany jest przestrzenią stanów.

### Optymalizacja dyskretna: podstawy

- Często możliwe jest oszacowanie kosztu osiągnięcia celu z pewnego stanu pośredniego.
- Oszacowanie to, zwane estymatą heurystyczną, może być pomocne w efektywnym wyszukiwaniu rozwiązania.
- Jeśli estymata jest niedoszacowana (faktyczny koszt będzie większy), to heurystyka taka nazywana jest *optymistyczną*.
- Heurystyka optymistyczna ma tę pożądaną własność, że przeszukiwanie za jej pomocą gwarantuje uzyskanie rozwiązania optymalnego.

### Przykład heurystyki optymistycznej

#### Dla problemu 8-układanki:

- Każde pole układanki będziemy indeksowali parą liczb (w, k).
- Odległość pomiędzy pozycjami (i, j) oraz (k, l) zdefiniujmy jako |i k| + |j l|. Zwana jest ona *odległością miejską* (Manhattan).
- Suma odległości miejskich między bieżącą pozycją a pozycją początkową obliczona po wszystkich kostkach jest heurystyką optymistyczną.

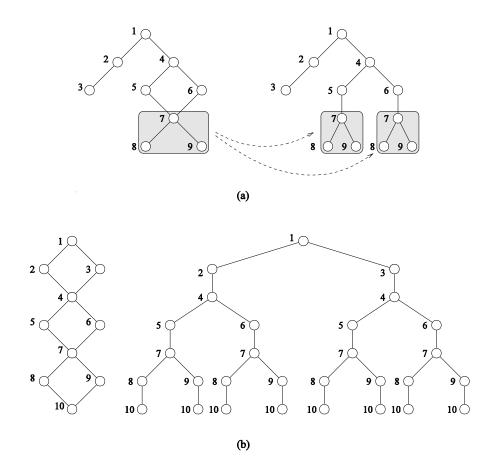
## Równoległa optymalizacja dyskretna

- Ponieważ problemy optymalizacji dyskretnej są z reguły obliczeniowo trudne (np. NP-zupełne) powstaje wątpliwość co do sensowności zrównoleglania?
- Są problemy, dla których średni czas działania jest wielomianowy.
- Często w czasie wielomianowym możemy znaleźć rozwiązanie bliskie optymalnemu.
- Dane wejściowe są na tyle małe, że wykładniczy czas działania nie jest przeszkodą.

# Algorytmy sekwencyjne dla wspomnianych problemów

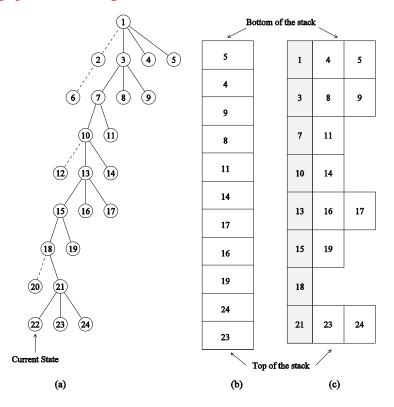
- Przestrzeń stanów jest drzewem czy grafem?
- W przypadku 0/1 programowania liniowego jest drzewem, natomiast dla 8-układanki jest grafem.
- Zarówno implementacja jest prostsza jak i czas działania jest krótszy, gdy mamy do czynienia z drzewem.

## Sekwencyjne algorytmy przeszukiwania



Dwa przykłady rozwinięcia grafu w drzewo

### Sekwencyjne wyszukiwanie z nawrotami



(a) Drzewo stanów (b) Stos zawierający nieprzetworzone wierzchołki (c) Informacja o rodzicach

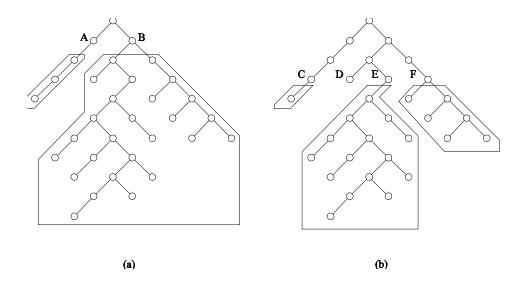
### Algorytm BFS

- Podczas przeszukiwania przestrzeni stanów korzysta z heurystyki.
- Nieprzetworzone wierzchołki przechowywane są w kolejce priorytetowej, gdzie kolejność wynika z estymaty heurystycznej.
- W każdym kroku algorytmu rozwijamy najbardziej obiecujący wierzchołek, a jego synów wstawiamy do kolejki priorytetowej.
- Algorytm BFS znajduje rozwiązanie optymalne, jeśli mamy do czynienia z heurystyką optymistyczną.

### Algorytm BFS

- Jeśli przestrzeń stanów jest grafem, to musimy włączyć do algorytmu zbiór odwiedzonych wierzchołków.
- Jeśli kolejny wygenerowany wierzchołek jest już w kolejce priorytetowej lub jest w zbiorze odwiedzonych wierzchołków z lepszą estymatą heurystyczną, to nie jest on wstawiany do kolejki priorytetowej.

## Równoległy algorytm z nawracaniem



Podział przestrzeni stanów w zależności od głębokości odciącia

#### Przykład: problem plecakowy

- Mamy zbiór 30 przedmiotów o następujących rozmiarach: {1,3,9,8,4,3,7,5,4,6,8,2,1,6,4,4,8,9,3,6,5,1,8,5,2,4,3,5,7,2} oraz wartościach: {1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9,1,2,3}.
- Celem jest wyznaczenie podzbioru przedmiotów
  o łącznym rozmiarze nie przekraczającym R = 40, tak
  aby suma ich wartości była możliwie największa.

### Zero-jedynkowe sformułowanie problemu

$$\max 1^*x[1] + 2^*x[2] + ... + 3^*x[30],$$

przy ograniczeniu:

$$1*x[1] + 3*x[2] + 9*x[3] + ... + 2*x[30] <= 40,$$

Gdzie *x* jest wektorem 0/1.

### Równoległa implementacja

```
#include <algorithm>
#include <iostream>
#include <mpi.h>
int cutoff count=0; // Licznik na glebokosci odciecia
int cutoff depth=5; // Glebokosc odciecia
int x[30]; // Sciezka poszukiwan
int id; // Numer procesu
int p; // Liczba procesow
int sizes[] =
{1,3,9,8,4,3,7,5,4,6,8,2,1,6,4,4,8,9,3,6,5,1,8,5,2,4,3,5,7,2};
int values[] =
\{1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9,1,2,3\};
const int R=40;
int best x[30];
int best value=0;
int best of all;
```

```
void Parallel Backtrack(int level) {
  if (level == 30) {
    int s=0;
    for (int i=0; i<30; i++)
      s += x[i]*values[i];
    if (s > best_value) {
      best value = s;
      std::copy(x, x+30, best_x);
  }
 else {
    if (level == cutoff depth) {
      if (++cutoff count % p != id) return;
    for (int i=0; i<=1; i++) {
      x[level] = i;
      if (i==0) Parallel Backtrack(level+1);
      else {
        int s=0;
        for (int i=0; i<=level; i++)
          s += x[i]*sizes[i];
        if (s <= R) Parallel Backtrack(level+1);</pre>
```

```
int main(int argc, char* argv[]) {
  MPI Init(&argc, &argv);
  MPI Comm_rank(MPI_COMM_WORLD, &id);
  MPI Comm size(MPI COMM WORLD, &p);
  Parallel Backtrack(0);
  MPI Allreduce(&best value, &best of all, 1, MPI INT,
                MPI MAX, MPI COMM WORLD);
  for (int i=0; i<p; i++) {
    if (i == id && best_value == best_of_all)) {
      for (int j=0; j<30; j++)
        std::cout << best x[i];</pre>
      std::cout << std::endl;</pre>
    MPI Barrier(MPI COMM WORLD);
  return MPI Finalize();
```

## Równoległy BFS

