Zaawansowane programowanie

wykład 5: algorytmy dokładne

prof. dr hab. inż. **Marta Kasprzak**
Instytut Informatyki, Politechnika Poznańska

Algorytmy dokładne

- Algorytmy dokładne służą rozwiązywaniu problemów w sposób dokładny (czyli *nieheurystyczny*).
  W przypadku problemów optymalizacyjnych oznacza to gwarancję wygenerowania rozwiązania optymalnego
- Problemy trudne obliczeniowo rozwiązywane są algorytmami działającymi w tzw. wykładniczym czasie:
  - problemy *silnie NP-trudne* (*silnie NP-zupełne*) rozwiązywane są algorytmami *wykładniczymi*, np. algorytmem branch-and-bound lub branch-and-cut
  - problemy *NP-trudne* (*NP-zupełne*) w zwykłym sensie mogą zostać rozwiązane algorytmami *pseudowielomianowymi*, np. algorytmem programowania dynamicznego

Branch & bound

- Metoda podziału i ograniczeń (ang. *branch-and-bound*) opiera się na przeszukiwaniu (najczęściej w głąb) drzewa reprezentującego przestrzeń rozwiązań problemu. Stosowane w tej metodzie odcięcia redukują liczbę przeszukiwanych węzłów (wykładniczą względem rozmiaru instancji)
- Metoda jest skomponowana — z grubsza rzecz ujmując — z dwóch podstawowych procedur:
  - rozgałęzianie (ang. *branching*) — dzielenie zbioru rozwiązań reprezentowanego przez węzeł na rozłączne podzbiory, reprezentowane przez następników tego węzła
  - ograniczanie (ang. *bounding*) — pomijanie w przeszukiwaniu tych gałęzi drzewa, o których wiadomo, że nie zawierają optymalnego rozwiązania w swoich liściach

Branch & bound

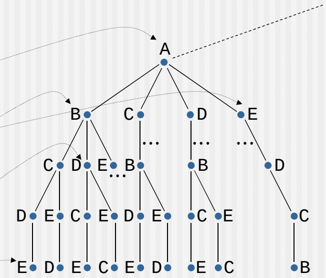
- Rozgałęzianie

'A' może być w korzeniu drzewa, jeśli rozpoczyna każde rozwiązanie

Następniki węzła muszą wyczerpywać wszystkie możliwe połączenia

Węzeł reprezentuje zbiór rozwiązań osiągalnych z niego

Liście drzewa reprezentują kompletne rozwiązania



Branch & bound

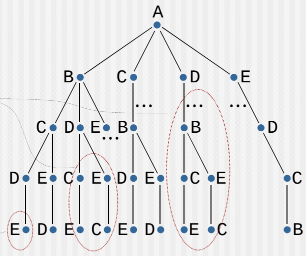
- Ograniczanie

Odcięcia w drzewie opierają się na bieżącej wartości funkcji celu

Im ta wartość bliższa optymalnej, tym większe gałęzie można pominąć

Wstępna heurystyka poprawia efektywność odcięć

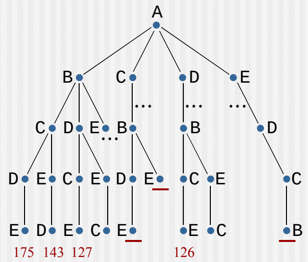
Miejsca odcięć wykrywa się z wyprzedzeniem (predykcja)



Branch & bound

- Przykład dla problemu komiwojażera (bez predykcji)

	A	B	C	D	E
A	0	31	32	11	18
B	31	0	59	23	46
C	32	59	0	40	15
D	11	23	40	0	27
E	18	46	15	27	0



Branch & bound

- W węźle drzewa porównywane są wartości tzw. dolnego i górnego ograniczenia (ang. *lower and upper bound*). Wynik tego porównania wpływa na decyzję o odcięciu gałęzi drzewa w tym węźle
- Przy założeniu minimalizacji funkcji celu, wartość tej funkcji dla najlepszego osiągniętego do tej pory rozwiązania stanowi *górne ograniczenie*
- W (prawie) każdym węźle obliczana jest aktualna wartość *dolnego ograniczenia*, która musi być nie większa niż wartość funkcji celu najlepszego rozwiązania możliwego do osiągnięcia z tego węzła

Branch & bound

- Dolne i górne ograniczenie (minimalizacja funkcji celu)

W bieżącym poddrzewie poszukujemy rozwiązań z obszaru pomiędzy dolnym i górnym ograniczeniem

Gdy obliczona w węźle wartość dolnego ograniczenia jest nie mniejsza niż górnego, odcinamy poddrzewo wychodzące z tego węzła — żadne zawarte w nim rozwiązanie nie będzie lepsze od posiadanego

Branch & bound

- Wartość dolnego ograniczenia musi zostać obliczona poprawnie w tym sensie, że rzeczywiście żadne rozwiązanie nie będzie miało lepszej wartości funkcji celu. Z drugiej strony, wartość ta może odbiegać od rzeczywistej wartości funkcji celu najlepszego rozwiązania w poddrzewie
- Należy dążyć do tego, aby wartość dolnego ograniczenia była jak najbliższa rzeczywiście istniejącemu rozwiązaniu. Pozwoli to na dokonanie bardziej efektywnych odcięć, czyli skróci czas obliczeń
- Dokładne obliczenie optymalnej wartości dolnego ograniczenia (równej wartości funkcji celu optymalnego rozwiązania w poddrzewie) wiąże się z wykładniczym czasem obliczeń, stąd stosuje się szybkie metody przybliżone

Branch & bound

- Przykładowo, dolne ograniczenie dla problemu komiwojażera można obliczyć sumując m+1 najmniejszych niewłączonych jeszcze do trasy odległości z macierzy, gdzie m jest liczbą nieodwiedzonych miast
- Bardziej dokładnym przybliżeniem byłoby sumowanie odcinków o najmniejszej długości spośród dochodzących do nieodwiedzonych miast (plus powrót do źródła), po jednym na każde takie miasto
- Dalej przybliżając tę wartość, można zliczać takie najmniejsze odległości z macierzy, które łączą dwa nieodwiedzone miasta, z uczynieniem wyjątku dla połączeń z bieżącą częścią rozwiązania
- Krokiem dalej jest obliczanie w każdym węźle drzewa rozwiązania dla problemu przydziału, w którym łączy się pozostałe miasta w pary i każde nieodwiedzone miasto występuje w dwóch takich parach

Branch & bound

- Problem przydziału (ang. *assignment problem*) sformułowany jest następująco:

$$\min z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1,\ldots,n$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,\ldots,n$$

c_{ij} — koszt przydziału

x_{ij} — zmienna decyzyjna (o wartości 0/1)

- Innymi słowy, należy z kwadratowej macierzy kosztów $n \times n$ wybrać n pozycji takich, że każdy wiersz i każda kolumna macierzy jest wybrana dokładnie raz oraz suma wskazanych kosztów jest minimalna

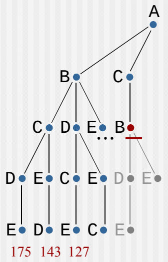
Branch & bound

- Problem przydziału rozwiązywany jest w wielomianowym czasie tzw. metodą węgierską. Mimo to zastosowanie tego podejścia w każdym węźle drzewa może wydłużyć obliczenia zamiast je skrócić
- Obiekty w wierszach i kolumnach mogą stanowić, w zależności od interpretacji, rozłączne lub identyczne zbiory (np. osoby i zadania lub miasta w problemie komiwojażera)
- Rozwiązanie problemu przydziału dla miast z problemu komiwojażera daje zbiór rozłącznych cykli. Minimalna wartość z zawsze będzie poprawnym dolnym ograniczeniem
- Aby wykluczyć niepożądany (w problemie komiwojażera) wybór kosztu z przekątnej macierzy, pozycjom tym przypisuje się na wstępie duże wartości

Branch & bound

- Przykład rozwiązania problemu przydziału w węźle drzewa

	A	B	C	D	E
A	0	31	32	11	18
B	31	0	59	23	46
C	32	59	0	40	15
D	11	23	40	0	27
E	18	46	15	27	0



A	D	E	
∞	23	46	B
11	∞	27	D
18	27	∞	E

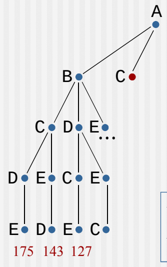
Rozw.: B-D,D-E,E-A
Wartość f. celu: 68
Dolne ogr.: 159
Górne ogr.: 127

13

Branch & bound

- Przykład rozwiązania problemu przydziału w węźle drzewa

	A	B	C	D	E
A	0	31	32	11	18
B	31	0	59	23	46
C	32	59	0	40	15
D	11	23	40	0	27
E	18	46	15	27	0



A	B	D	E	
∞	59	40	15	C
31	∞	23	46	B
11	23	∞	27	D
18	46	27	∞	E

Rozw.: C-E,B-D,D-B,E-A
Wartość funkcji celu: 79
Dolne ograniczenie: 111
Górne ograniczenie: 127

14

Branch & bound

- Porównanie metod o różnym stopniu dokładności stosowanych do obliczenia dolnego ograniczenia — przykład (→ slajd 10)

metoda	rozwiązanie	wartość
m+1 najmniejszych odległości	A-D, C-E, A-E	11+15+18=44
najmniejsze odległości dochodzące do m+1 miast	A-D, C-E, A-E	11+15+18=44
najmniejsze odległości pomiędzy parami miast	B-D, D-E, A-D	23+27+11=61
problem przydziału	B-D, D-E, A-E	23+27+18=68

15

Branch & bound

- Wynik metody obliczającej dolne ograniczenie może, wraz z częścią już istniejącego rozwiązania, stanowić poprawne rozwiązanie głównego problemu. W takim przypadku jest to najlepsze (lub jedno z najlepszych) rozwiązanie możliwe do osiągnięcia w poddrzewie wychodzącym z analizowanego węzła i można w tym miejscu zakończyć rozgałęzianie
- Czasami można zaoszczędzić na obliczeniach, przechodząc z węzła do jego następnika, gdyż problemy rozwiązywane w sąsiednich węzłach są podobne
- Struktura drzewa często opierana jest na elementach rozwiązania — każdy element w jednym węźle — które po dodaniu do siebie tworzą rozwiązanie. Można jednak zastosować inny schemat, np. węzeł oznacza brak danego elementu w rozwiązaniu

16

Branch & bound

- Oprócz właściwego doboru schematu rozgałęziania i ograniczania, istotnym elementem jest uporządkowanie następników węzła. Kolejność ich odwiedzania ma wpływ na wcześniejsze osiągnięcie lepszego górnego ograniczenia, a więc na czas obliczeń
- Najczęściej stosowanymi strategiami są:
  - *least-cost-next*
  - *least-lower-bound-next*
  - *last-in-first-out*
  - *first-in-first-out*
- Uporządkowanie zależy w dużej mierze od rodzaju problemu

17

Branch & bound

- Można zrezygnować z obliczania dolnego ograniczenia na najniższych poziomach drzewa z uwagi na oszczędność czasu, ew. najwyższych z uwagi na niską skuteczność
- Warto wyposażyć algorytm w mechanizm przerywania zbyt długich obliczeń. Wtedy zamiast stracić dokonane obliczenia możemy uzyskać wynik przybliżony, często o bardzo dobrej jakości
- Oprócz najlepszego rozwiązania osiągniętego do momentu przerwania obliczeń algorytm może zwrócić najniższą wartość dolnego ograniczenia obliczoną dla wszystkich nierozgałęzionych węzłów. Wiemy wtedy, że poszukiwana wartość optymalna znajduje się pomiędzy tymi dwiema wartościami

18

Branch & cut

- Metoda podziału i odcięć (ang. *branch-and-cut*) powstała przez połączenie dwóch metod: podziału i ograniczeń oraz płaszczyzn odcinających (ang. *cutting-plane*)
- Metoda — podobnie jak *branch-and-bound* — służy do rozwiązywania problemów kombinatorycznych, czyli takich, w których zmienne mają wartości całkowite. Problem jest wyrażany zazwyczaj w postaci układu równań i nierówności programowania liniowego całkowitoliczbowego (ang. *integer linear programming*, ILP, przykład na slajdzie 11)
- Rozwiązanie problemu ILP jest trudne obliczeniowo. W praktycznych podejściach stosuje się metodę przybliżoną, polegającą na rozwiązaniu problemu bez ograniczenia wartości zmiennych do liczb całkowitych (metodą *simplex*), z zamianą uzyskanych wartości ułamkowych na całkowitoliczbowe

19

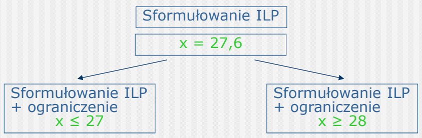
Branch & cut

- Proste zaokrąglenie wartości ułamkowych do najbliższych liczb całkowitych najczęściej nie sprawdza się, gdyż wartość taka może być niedopuszczalna (naruszająca ograniczenia z problemu)
- Metoda *cutting-plane* Ralpha Gomory'ego polega na wprowadzaniu do sformułowania problemu dodatkowych zmiennych i dodawaniu nierówności mających na celu wyeliminowanie wartości ułamkowych kolejnych zmiennych
- Metoda Gomory'ego jest powiązana z postacią równań z metody simplex (opis obu tych metod wykracza poza program wykładu). W praktyce stosuje się w metodzie *branch-and-cut* także uproszczone podejście, bez dodatkowych zmiennych i z prostszymi nierównościami (kolejny slajd)

20

Branch & cut

- W każdym węźle drzewa rozwiązywany jest problem ILP w sposób przybliżony metodą *simplex* i dodawana jest nierówność w dwóch wariantach dla wybranej zmiennej, co prowadzi do dwóch nowych sformułowań i rozgałęzienia węzła



- W momencie uzyskania rozwiązania bez wartości ułamkowych dla zmiennych całkowitoliczbowych, mamy rozwiązanie dopuszczalne problemu i kończymy rozgałęzianie w tym węźle

21

Branch & cut

- Wartość funkcji celu najlepszego dotąd otrzymanego rozwiązania całkowitoliczbowego stanowi górne ograniczenie. Dolnym ograniczeniem jest wartość funkcji celu wyliczona metodą *simplex* dla problemu przybliżonego
- Liczba wywołań metody *simplex* bywa ograniczana, nie uruchamia się jej wtedy w każdym węźle
- Podobnie jak w *branch-and-bound*, wstępna heurystyka poprawia jakość odcięć
- Stosuje się wstępne przetworzenie problemu ILP obejmujące:
  - eliminację zbędnych zmiennych
  - ustalenie zmiennych o stałej wartości
  - uproszczenie nierówności

22

Branch & cut

- W przypadku zero-jedynkowego programowania liniowego, w którym zmienne decyzyjne przyjmują wartości 0 lub 1, rozgałęzianie może zostać zrealizowane w jeszcze bardziej uproszczony sposób. Metoda *simplex* może być używana wtedy do obliczania dolnego ograniczenia w wybranych węzłach
- Przykład 0-1 LP — problem plecakowy

$$\max\ f = \sum_{i=1}^{n} w_i x_i$$

$$\sum_{i=1}^{n} s_i x_i \le k$$

$$x_i \in \{0,1\}, \quad i = 1,\dots,n$$

Dane w problemie:

- *n — liczba elementów*
- *s_i — rozmiar elementu*
- *w_i — wartość elementu*
- *k — rozmiar plecaka*

23

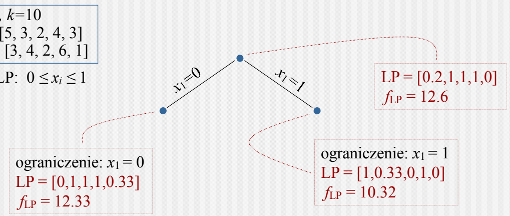
Branch & cut

- Przykład rozgałęzienia dla problemu plecakowego

Instancja problemu:

n=5, k=10
s_i = [5, 3, 2, 4, 3]
w_i = [3, 4, 2, 6, 1]

dla LP: $0 \le x_i \le 1$



24

Programowanie dynamiczne

[R. Bellman, *Proceedings of the National Academy of Sciences of the USA* 38, 1952, 716–719]

- Programowanie dynamiczne (ang. *dynamic programming*) jest metodą stosowaną do optymalnego rozwiązania problemów zarówno wielomianowych, jak i NP-trudnych (NP-zupełnych)
- Problemy te muszą spełniać *zasadę optymalności Bellmana*, tzn. decyzja optymalna podjęta w kroku i jest nadal optymalną w kroku i+1 i kolejnych. Mają one tzw. *optymalną podstrukturę*, czyli rozwiązania optymalne dla podproblemów składają się na rozwiązanie optymalne całego problemu
- Nie ma nawrotów w tej metodzie

Programowanie dynamiczne

- W programowaniu dynamicznym wypełnia się k-wymiarową macierz wartości o rozmiarze ograniczonym zazwyczaj (w zastosowaniach praktycznych) wielomianem będącym funkcją rozmiaru instancji i największej liczby występującej w instancji
- Najbardziej znanym w bioinformatyce algorytmem programowania dynamicznego jest algorytm dopasowania dwóch sekwencji (algorytm Needlemana-Wunscha, algorytm Smitha-Watermana)
- Na kolejnych slajdach przedstawiony jest przykład algorytmu programowania dynamicznego dla problemu plecakowego (→ slajd 23). Algorytm ten ma złożoność pseudowielomianową $O(n \cdot k)$

Programowanie dynamiczne

Instancja problemu:

n=5, k=10
s_i = [5, 3, 2, 4, 3]
w_i = [3, 4, 2, 6, 1]

Algorytm:

i=0..n, j=0..k,
$\forall i \;\; f(i, 0) = 0$
$\forall j \;\; f(0, j) = 0$
$f(i, j) = f(i-1, j)$ gdy $j < s_i$,
$f(i, j) = \max\{ f(i-1, j-s_i)+w_i, f(i-1, j)\}$ wpp.

Optimum: $f(n, k)$

Tablica programowania dynamicznego:

i \ j	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	3	3	3	3	3	3
2	0	0	0	4	4	4	4	4	7	7	7
3	0	0	2	4	4	6	6	6	7	7	9
4	0	0	2	4	6	6	8	10	10	12	12
5	0	0	2	4	6	6	8	10	10	12	12

Programowanie dynamiczne

Instancja problemu:

n=5, k=10
s_i = [5, 3, 2, 4, 3]
w_i = [3, 4, 2, 6, 1]

Rozwiązanie odczytujemy od końca, cofając się z pola (n,k) po kolei do pól, z których wywiedzione zostały wartości składające się na optymalną ścieżkę. Kończymy na wartości 0.

Tablica programowania dynamicznego:

i \ j	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	3	3	3	3	3	3
2	0	0	0	4	4	4	4	4	7	7	7
3	0	0	2	4	4	6	6	6	7	7	9
4	0	0	2	4	6	6	8	10	10	12	12
5	0	0	2	4	6	6	8	10	10	12	12

Rozwiązaniem jest podzbiór elementów: {2, 3, 4}

Literatura – cd.

- Christos H. Papadimitriou, Kenneth Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, 1982.