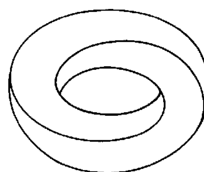
Algorytm APPROX-SUBSET-SUM
jako rozwiązanie problemu optymalnego wykorzystania czasu pracy

Mariusz Gromada
grudzień 2002
mariusz.gromada@wp.pl
http://multifraktal.net



1 Wstęp

Ideę problemu optymalnego wykorzystania czasu pracy dobrze obrazuje następujący przykład. Wyobraźmy sobie pracownika pewnej firmy. Przypuśćmy, że pracownik każdego dnia powinien spędzać w miejscu zatrudnienia t godzin, wykonując zadania przydzielone na bieżący dzień. Przełożony jest wymagający i pracownik nie może zakończyć pracy przed wykonaniem każdego z zadań. Nadgodziny są płatne dodatkowo. W interesie pracodawcy leży to, aby optymalnie zaplanować czas pracy pracownika. Pracodawca nie chce płacić „za nic", więc pracownik powinien zawsze wypracować co najmniej t godzin. Nadgodziny słono kosztują, czyli powinno być ich jak najmniej. Ogólnie zadanie przełożonego polega na optymalnym doborze takich k zadań spośród n możliwych do wykonania, przy założeniu znajomości szacowanego czasu wykonywania każdego z nich. Optymalny dobór oznacza, że czas wykonywania wybranych k zadań powinien być nie mniejsza od t i możliwie najkrótszy.

1.1 Problem optymalnego wykorzystania czasu pracy

Dane: para (S, t) gdzie:
$S = \{x_1, x_2, \ldots, x_n\}$ - zbiór dodatnich liczb całkowitych,
x_i - czas wykonania $i - tego$ zadania $(i = 1, \ldots, n)$,
t - czas pracy (dodatnia liczba całkowita).

Problem 1.1 *Które zadania wybrać, by czas ich wykonania był nie mniejszy od t i możliwie najkrótszy?*

Zagadnienie minimalizacji. Niech z_i oznacza sumę i różnych elementów ze zbioru S $(i = 1, 2, \ldots, n)$. Zapisujemy:

$$\begin{aligned} z_i &\geqslant t \\ z_i &\to \min \end{aligned} \tag{1}$$

Rozwiązanie powyższego zagadnienia, o ile istnieje, oznaczamy przez z_o i dalej nazywamy rozwiązaniem optymalnym problemu 1.1.

1.2 Problem sumy podzbiorów (klasyczny i optymalizacyjny)

Dane: para (S, t) gdzie:
$S = \{x_1, x_2, \ldots, x_n\}$ - zbiór dodatnich liczb całkowitych,
t - dodatnia liczba całkowita.

Problem 1.2 (klasyczny) *Czy istnieje podzbiór zbioru S, którego suma równa jest dokładnie docelowej wartości t?*

Problem ten jest $NP - zupeny$ - nie istnieje algorytm działający w czasie wielomianowym rozwiązujący to zadanie. Możliwe jest skonstruowanie algorytmu wykładniczego.

Dane: trójka (S, t, ϵ) gdzie:
$S = \{x_1, x_2, \ldots, x_n\}$ - zbiór dodatnich liczb całkowitych,
t - dodatnia liczba całkowita,
ϵ - parametr opisujący jakość aproksymacji $(0 < \epsilon < 1)$.

Problem 1.3 (optymalizacyjny) *Należy znaleźć podzbiór zbioru S, którego suma jest możliwie największa, ale nie przewyższa ustalonej dodatniej liczby t.*

Zagadnienie maksymalizacji. Niech z_i oznacza sumę i różnych elementów ze zbioru S $(i = 1, 2, \ldots, n)$. Zapisujemy:

$$\begin{aligned} z_i &\leqslant t \\ z_i &\to \max \end{aligned} \tag{2}$$

Rozwiązanie powyższego zagadnienia (o ile istnieje) oznaczamy przez $\tilde{z}_o$.

Problem 1.3 rozwiązuje algorytm **Approx-Subset-Sum** przy ustalonym parametrze jakości aproksymacji.

1.3 Algorytm Approx-Subset-Sum

Approx-Subset-Sum(S,t,e)

1. $n \leftarrow |S|$
2. $L_0 \leftarrow< 0 >$
3. **for** $i \leftarrow 1$ **to** n
4. **do** $L_i \leftarrow$ **MERGE-LISTS**(L_{i-1},$L_{i-1} + x_i$)
5. $L_i \leftarrow$ **TRIM**(L_i,$\frac{\epsilon}{n}$)
6. usuń z L_i wszystkie elementy większe niż t
7. niech z będzie największą wartością w L_n
8. **return** z

Wykorzystane zostały następujące procedury:

- **MERGE-LISTS**(L,L') - łączy listy L i L' i zwraca posortowany ciąg (zakładając, że listy L i L' sa posortowane niemalejąco otrzymujemy złożoność $O(|L| + |L'|)$)
- **TRIM**(L,δ) - skraca listę L z parametrem skracania $0 < \delta < 1$ (L - posortowana niemalejąco, złożoność $O(|L|)$)

1.3.1 Procedura TRIM(L,δ)

TRIM(L,δ)

1. $m \leftarrow |L|$
2. $L' \leftarrow y_1$
3. $last \leftarrow y_1$
4. **for** $i \leftarrow 2$ **to** m
5. **do if** $last < (1 - \delta)y_i$
6. **then** dołącz y_i na koniec listy L'
7. $last \leftarrow y_i$

8. **return** L'

Skrócenie listy L przez δ oznacza usunięcie z listy L możliwie największej liczby elementów tak, żeby dla każdego elementu y usuniętego z listy L na liście L', będącej wynikiem skrócenia L, istniał element $z \leqslant y$ taki, że:

$$\frac{y-z}{z} \leqslant \delta$$

lub równoważnie:

$$(1-\delta)y \leqslant z \leqslant y$$

Procedura **TRIM** może znacznie skrócić długość listy L_i. Przekonamy się później, że dzięki niej otrzymujemy algorytm działający w czasie wielomianowym. Ma to poważne zastosowanie praktyczne. Dobrym tego przykładem może być załadunek ciężarówki o maksymalnej nośności t skrzynkami o podobnych ciężarach.

Algorytm **Approx-Subset-Sum** znajduje poprawne rozwiązanie, o ile ono istnieje, problemu 1.3 i jest on w pełni **wielomianowym schematem aproksymacji**. Rozwiązanie to mieści się w zakresie błędu ϵ (od rozwiązania optymalnego $\tilde{z_o}$)

$$(1-\epsilon)\tilde{z_o} \leqslant z \leqslant \tilde{z_o}$$

2 Rozwiązanie problemu

Rozwiązanie problemy optymalnego wykorzystania czasu pracy polega na modyfikacji algorytmu Approx-Subset-Sum. Niezbędne zmiany nie mają wpływu na rząd złożoności zmodyfikowanego algorytmu Approx-Subset-Sum.

2.1 Modyfikacja Approx-Subset-Sum

Postawmy problem optymalnego wykorzystania czasu pracy:
Dane: trójka (S, t, ϵ) gdzie: $S = \{x_1, x_2, \ldots, x_n\}$ - zbiór dodatnich liczb całkowitych,
t - dodatnia liczba całkowita,zz ϵ - parametr opisujący jakość aproksymacji $(0 < \epsilon < 1)$.

Zadanie sprowadzamy do optymalizacyjnego problemu sumy podzbiorów.

Oznaczamy:

$P = \sum_{i=1}^{n} x_i$ - oczywiste jest, że aby istniało rozwiązanie to $t \leqslant P$,
$y_{n-i} = P - z_i$ - suma elementów nie uwzględnionych przy obliczaniu z_i.

Zagadnienie minimalizacji

$$\begin{array}{l} z_i \geqslant t \\ z_i \rightarrow \min \end{array}$$

można równoważnie zapisać jako zagadnienie maksymalizacji.

$$\begin{array}{l} P - z_i \leqslant P - t \\ (P - z_i) \rightarrow \max \end{array} \Leftrightarrow \begin{array}{l} y_{n-i} \leqslant P - t \\ y_{n-i} \rightarrow \max \end{array} \tag{3}$$

a to jest już problem sumy podzbiorów. Rozwiązanie znajdujemy natychmiast wykorzystując algorytm y_o=**Approx-Subset-Sum(S,P-t,ϵ)** gdzie $z = P - y_o$ jest poszukiwaną przez nas wartością. Z własności algorytmu Approx-Subset-Sum łatwo wynika oszacowanie:

$$z + 0 \leqslant z \leqslant (1 + \epsilon) z_o$$

Zmodyfikowany algorytm Approx-Subset-Sum(S,t,ϵ)

1. $P = \sum_{i=1}^{n} x_i$
2. $n \leftarrow |S|$
3. $L_0 \leftarrow < 0 >$
4. **for** $i \leftarrow 1$ **to** n
5. **do** $L_i \leftarrow$ **MERGE-LISTS**(L_{i-1},$L_{i-1} + x_i$)
6. $L_i \leftarrow$ **TRIM**(L_i,$\frac{\epsilon}{n}$)
7. usuń z L_i wszystkie elementy większe niż $P - t$
8. niech z będzie największą wartością w L_n
9. **return** $P - z$

Zaproponowany algorytm działa w prawie identycznym czasie co algorytm Approx-Subset-Sum. Różnica występuje jedynie w pierwszym kroku, gdzie wyznaczana jest suma elementów z zbioru S. Jest to w pełni wielomianowy schemat aproksymacji, który dobrze przybliża wartość rozwiązania optymalnego.

3 Analiza znanych wyników

Opiszemy znane wyniki dla algorytmu Approx-Subset-Sum. Przykład zaczerpnięty z [1]. Rozpatrujemy zbiór: $S = \{104, 102, 201, 101\}$, $t = 308$, $\epsilon = 0,20$

```
wiersz 2: L0 = <0>
wiersz 4: L1 = <0, 104>
wiersz 5: L1 = <0, 104>
wiersz 6: L1 = <0, 104>

wiersz 4: L2 = <0, 102, 104, 206>
wiersz 5: L2 = <0, 102, 206>
wiersz 6: L2=  <0, 102, 206>

wiersz 4: L3= <0, 102, 201, 206, 303, 407>
wiersz 5: L3= <0, 102, 201, 303, 407>
wiersz 6: L3= <0, 102, 201, 303>

wiersz 4: L4= <0, 101, 102, 201, 203, 302, 303, 404>
wiersz 5: L4= <0, 101, 201, 302, 404>
wiersz 6: L4= <0, 101, 201, 302>
```

Odpowiedź udzielona przez algorytm to $z = 302$. Rozwiązanie optymalne $\tilde{z}_o = 307 = 104 + 102 + 101$. Wynik mieści się z powodzeniem w zakresie $\epsilon = 20\%$ od optymalnego rozwiązania.

Literatura

[1] Thomas. H. Cormen, Charles E. Leiserson, Ronald L. Rivest: *Wprowadzenie do algorytmów*