Symulowane wyżarzanie do rozwiązania problemu jednowymiarowego problemu upakowaniat

Joanna Kołodziejczyk

1 Problem do rozwiązania

Celem laboratorium jest implementacja algorytmu symulowanego wyżarzania dla jednowymiarowego problemu upakowania (bin packing problem).

1.1 Definicja problemu

Biorąc pod uwagę n elementów o różnych rozmiarach a_j i nieskończoną liczbę pojemników (każdy o stałej, skończonej pojemności b), należy upakować elementy w **minimalną** liczbę pojemników, aby żaden pojemnik nie został przepełniony.

Przy rozwiązywaniu zadania trzeba brać pod uwagę następujące ograniczenia:

- Każdy element może być przypisany tylko do jednego pojemnika.
- Całkowita waga/rozmiar umieszczonych w jednym pojemniku elementów musi być mniejsza lub równa zadanej pojemności pojemników. Jeżeli i jest indeksem pojemnika, a k jest znalezioną lub minimalną liczbą pojemników to ograniczenie można wyrazić jako nierówność:

$$\exists poj : \forall_{i \in 0..k-1} : \left(\sum_{\forall j \in poj[i]} a_j \right) \leq b$$

Celem jest znalezienia najmniejszej liczby k pojemników o wielkości b, które mogą zawierać wszystkie obiekty wymienione w a. Alternatywnie można by spróbować znaleźć mapowanie x, dla którego funkcja f przyjmuje najniższą wartość.

$$f(x) = \sum_{i=0}^{k-1} \max \left\{ 0, \left(\sum_{\forall j \in x[i]} a_j \right) - b \right\} \tag{1}$$

2 Algorytm

Poniżej przedstawiony jest kod algorytmu symulowanego wyżarzania dla zadania minimalizacji funkcji f_{min}.

$x \leftarrow x_0$
$eval \leftarrow f_{min}(x)$
$idx \leftarrow 0$
$T(0) = tempestimation(x)$
while $idx < idx_{max}$ **do**
  $x_n \leftarrow neighbour(x)$
  $eval_n \leftarrow f_{min}(x_n)$
  if $Prob(eval, eval_n, temp(idx)) > random()$ **then**
    $x \leftarrow x_n$
    $eval \leftarrow eval_n$
  end if
  $idx \leftarrow idx + 1$
end while
return x

2.1 Istotne elementy algorytmu

Aby wykorzystać algorytm symulowanego wyżarzania do rozwiązania problemu jednowymiarowego upakowania należy rozważyć:

- **Kodowanie kandydata na rozwiązanie**: Jednym ze sposobów jest przedstawienie x jako n wymiarowy wektor. Indeksy w wektorze (pozycje) oznaczają j-ty element z dostępnych a_j elementów. Wartości w wektorze x oznaczają numer pojemnika. Np.

  $$x = \{2, 1, 0, 4, 1, 0\}$$

  dla odpowiedniego wektora

  $$a = \{4, 2, 10, 2, 4, 5\}$$

  oznacza, że element a_0 ważący 4 jest umieszczony w pojemniku 3, w pojemniku 2 umieszczono elementy a_1 i a_4 ważące odpowiednio 2 i 4.

- x_0 - Wektor x z wartościami wylosowanymi z rozkładu jednostajnego z przedziału od $\{0..n-1\}$. Przedział określający maksymalną liczbę pojemników, to n, co oznacz, że każdy element jest umieszczony w osobnym pojemniku.

- x_n nowy kandydat otrzymywany jest poprzez procedurę $neighbour(x)$.

- Procedura $neighbour(x)$ wybiera losowo pozycję w wektorze x (indeks j) i zamienia wartość $x(j)$ na inną wygenerowaną losowo z rozkładu jednostajnego z przedziału $\{0..n-1\}$. innymi słowy przemieszcza element do innego pojemnika.

  $j \leftarrow random(n)$
  loop
    if $random(n) \neq x(j)$ **then**
      $x(j) \leftarrow random(n)$
      break

end if
end loop

- **Funkcja oceniająca jakość kandydata**. Można na potrzeby przedstawionego zadania opracować różne miary. Głównym celem jest uzyskanie najmniejszej liczby pojemników, jednocześnie dbając o to by nie przekraczać założonej pojemności. Konstrukcja funkcji będzie taka, że będziemy z pomocą algorytmu szukać jej najmniejszej wartości. f_{min} może mieć następujący wzór[1]:

$$f_{min}(x) = k + s \cdot (n + f(x)),$$

gdzie: k liczba różnych (unikalnych wartości) pojemników użytych w wektorze x. Innymi słowy ile niepustych pojemników użyto w kandydacie x. W przykładzie $x = \{2, 1, 0, 4, 1, 0\}$ $k = 4$;

$s \in \{0, 1\}$ określa czy rozwiązanie jest poprawne (0) czy nie (1). Poprawne, to takie, dla którego nie ma przepełnienia w pojemnikach, czyli $f(x) = 0$. Niepoprawne, to takie, dla którego następuję przepełnienie $f(x) > 0$. Interpretacja jest taka, że jeżeli rozwiązanie jest niepoprawne do nakładamy na nie karę dodając do liczby pojemników wartość przekroczenia powiększoną o maksymalną liczbę pojemników;

n jest liczbą elementów w wektorze a, czyli też maksymalną liczbą pojemników.

- max wartość określając maksymalną liczbę kroków w algorytmie. Wartością sterujemy kiedy nastąpi zakończenie procedury.

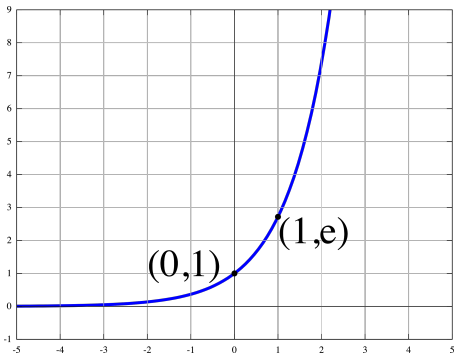
- Procedura `tempestimation(x)` powinna obliczyć początkową temperaturę, która jest wartością związaną z rozwiązywanym problemem i osiąganą wartością funkcji celu. Temperatura początkowa musi być wysoka. Jednym ze sposobów jest przyjęcie jako T_0 odchylenia standardowego od średniej z funkcji oceny. Uzyskanie tej wartości jest możliwe poprzez wielokrotnie wykonanie procedury neighbou()r i wyznaczenie wartości odchylenia standardowego.

$t \leftarrow 0$
$sumEval \leftarrow 0$
$sqSumEval \leftarrow 0$
while $t < tMax$ **do**
  $x_n \leftarrow neighbour(x)$
  $eval_n \leftarrow f_{min}(x_n)$
  $sumEval \leftarrow sumEval + eval_n$
  $sqSumEval \leftarrow sqSumEval + f_{min}(x_n)^2$
  $t \leftarrow t + 1$
end while
$mean \leftarrow sumEval/tMax$
$variance \leftarrow sqSumEval/tMax - mean^2$

[1] S. Kuri at al. „ Evolutionary Heuristics for the Bin Pasking Probelem `http://www9.in.tum.de/praktika/awbs_info_general/khuri95evolutionary.pdf`

return $std \leftarrow sqrt(variance)$

- Procedura *Prob* ma wyznaczyć z jakim prawdopodobieństwem zmienimy stan/konfigurację bieżącą na nową. Procedura *Prob* wykorzystuje funkcję eksponencjalną.



$$Prob(eval, eval_n, temp(idx)) = \begin{cases} exp(\frac{eval-eval_n}{temp()}), & \text{jeżeli } eval - eval_n < 0 \\ 1, & \text{w przeciwnym wypadku} \end{cases}$$

W każdej iteracji wartość temperatury zmniejsza się. Proces ten nazywa się schematem schładzania. Istnieje wiele modeli chłodzenia. Jednym z najprostszych jest zmienianie temperatury w każdej iteracji przez mnożenie jej przez stały współczynnik mniejszy od 0.

Procedura $temp(idx)$ zmienia temperaturę: $T(idx+1) = \alpha * T(idx)$, gdzie np. $\alpha = 0.95$.

3 Wymagania dla implementacji

Język implementacji algorytmu jest dowolny.

Do testowania należy wykorzystać paczkę z testowymi przykładami (benchmarki) do rozwiązania. Każdy plik testowy zawiera następujące dane [2] :

1. wiersz pierwszy to liczba elementów n,
2. wiersz drugi to wartość b czyli maksymalną pojemność pojemników,
3. kolejne n wierszy to wartości wag/wielkości elementów $a_j, j \in \{0..n-1\}$.

W nazwie benchmarku pierwszy człon przed znakiem „_” jest minimalnym rozwiązaniem globalnym.

Wymagane elementy programu:

- Wczytać benchmark/plik testowy z pliku.

[2] Więcej plików testowych można znaleźć na stronie `https://www2.wiwi.uni-jena.de/Entscheidung/binpp/bin1dat.htm`

- Zaimplementować algorytm zgodnie z wytycznymi podanymi w punkcie 2 i 2.1.
- Program po wykonaniu optymalizacji powinien wyświetlać:
  - stan, który został wskazany przez program jako rozwiązanie i jego wartość: ile pojemników użyto i jak rozdzielono elementy do pojemników oraz podać sumę wag z każdego pojemnika
  - demonstrować w postaci wykresu (ewentualnie listy) zmienność wartości funkcji f_{min} w procesie optymalizacji. Na osi odciętych powinny być przyrastające wartości idx a na osi rzędnych wartości funkcji f_{min}.
- Ponadto uruchomić algorytm dziesięciokrotnie i podać średnią arytmetyczną z funkcji f_{min} w 10 uruchomieniach oraz odchylenie standardowe.

Każde własne (rozsądne) propozycje modyfikacji lub wykorzystane z literatury zmiany w algorytmie lub reprezentacji lub prezentacji wyników ponad te, wymagane w instrukcji,będą dodatkowo punktowane.