Wrocław, 27 maja 2021

Sterowanie **P**rocesami **D**yskretnymi

Laboratorium 6

Algorytmy metaheurystyczne

prowadzący: *mgr inż. Radosław Idzikowski*

1 Cel laboratorium

Celem laboratorium jest zapoznanie się algorytmami metaheurystycznymi. Obejmuje to odpowiednie zdefiniowane problemu (ograniczeń i funkcji celu), danych wejściowych oraz implementacje dedykowanych algorytmów, a także interpretację wyników.

2 Przebieg zajęć

Laboratorium obejmuje zajęcia nr 12-14 (6 godzin zajęć, w tym oddanie). Praca odbywa się w ramach dwuosobowych zespołów. Każdy zespół otrzymuje do opracowania problem do wyboru. W tym wypadku $FP||C_{\max}$ lub $J||C_{\max}$ oraz dedykowane dla niego algorytmy.

W trakcie zajęć należy zaimplementować wybrane algorytmy z poniższej listy.

- Symulowane Wyżarzanie (*Simulated Annealing*),
- Przeszukiwanie z Zabronieniami (*Tabue Search*),
- Algorytm Genetyczny z wykorzystaniem pakiety `DEAP` *Genetic Algorithm*.

Oprócz implementacji samych algorytmów, należy w programie zapewnić możliwość porównania z wcześniej napisanymi algorytmami dla wybranego problemu.

Na ocenę `5.0` należy zaimplementować jeden wybrany algorytm dla problemu $J||C_{\max}$ lub dwa wybrane algorytmy dla problemu $FP||C_{\max}$. Aby otrzymać oceną `4.0` trzeba napisać jeden wybrany algorytm dla problemu $FP||C_{\max}$. W celu uzyskania oceny `3.0` wystarczy zaimplementować prosty algorytm Przeszukiwania Losowego (*Random Search*) lub Przeszukiwania Lokalnego (*Local Search*) dla dowolnego problemu szeregowania zadań.

3 Metody rozwiązania

3.1 Symulowane Wyżarzanie

Algorytm metaheurystyczny o charakterze probalistycznym inspirowany procesem metalurgicznym. Na początku należy przyjąć temperaturę początkową $T_0 > 0$ co będzie symbolizowane przez podgrzanie metalu czyli naszego rozwiązania. Jako rozwiązanie początkowe możemy przyjąć: (1) permutację naturalną, (2) losową, (3) wynik alg. zachłannego. Warunkiem zatrzymania algorytmu jest spadek temperatury T naszego rozwiązania poniżej temperatury końcowej T_{end}. Dla każdej temperatury będziemy wykonywać L iteracji wewnętrznych (epok). W każdej epoce należy wykonać jeden pojedynczy losowy ruchu: (1) zamień (*swap*), (2) wstaw (*insert*), (3) odwróć (*twist/reverse*) lub (4)

zamień sąsiedni (*adjacent swap*). Przez całe działanie algorytmu wykonujemy jeden typ ruchu (dopuszczalne są ruchy hybrydowe). Jeśli Rozwiązanie po wykonaniu ruchu jest lepsze niż aktualne, należy je zaakceptować (podmienić), w przeciwnym wypadku z pewnym prawdopodobieństwem p również możemy je zaakceptować:

$$p = e^{\frac{\Delta C_{\max}}{T}} \tag{1}$$

gdzie

$$\Delta C_{\max} = C_{\max}(\pi) - C_{\max}(\pi_{\text{new}}) \tag{2}$$

Prawdopodobieństwo akceptacji gorszego rozwiązania będzie malało wraz ze spadkiem temperatury oraz przy bardzo dużej różnicy wartości funkcji celu. W procesie metalurgicznym gorący materiał jest bardziej podatny na zmiany niż zimny. Idea algorytmu polega na przeglądaniu początkowo jak największego sąsiedztwa, również przechodząc przez gorsze rozwiązania, aby na końcu zbiec do któregoś minimum lokalnego akceptując już tylko lepsze rozwiązania. Po wykonaniu wszystkich iteracji wewnętrznych (epok) należy obniżyć temperaturę według ustalanego schematu chłodzenia:

- liniowego $T' = T - x$,
- geometrycznego $T' = \alpha T$,
- logarytmicznego $T' = \frac{T}{\ln(it+1)}$, gdzie it – nr iteracji algorytmu.

Alternatywnie algorytm można przerwać w momencie niezaakceptowania ani jednego rozwiązania podczas wykonywania całej epoki.

Algorithm 1 Simulated Annealing

```
1: T ← T_0
2: π ← InitSolution()
3: while T > T_end do
4:     for k = 1 to L do
5:         i ← RandomInt(1, n)
6:         j ← RandomInt(1, n)
7:         π_new ← π.Move(i, j)
8:         if Calculate(π_new) > Calculate(π) then
9:             r ← RandomDouble(0, 1)
10:            if r ⩾ e^(ΔC_max / T) then
11:                π_new ← π
12:            end if
13:        end if
14:        π ← π_new
15:        if Calculate(π) < Calculate(π*) then
16:            π* ← π
17:        end if
18:    end for
19:    T ← ReduceTemperature(T)
20: end while
```

3.2 Przeszukiwanie z Zabronieniami

Algorytm metaheurystyczny możemy pozbawić losowości jeśli jako rozwiązanie początkowe przyjmiemy permutację naturalną lub wynik algorytmu zachłannego zamiast rozwiązania losowego. Warunkiem stopu algorytmu jest liczba iteracji lub limit czasowy. Algorytm sprawdza całe sąsiedztwo

wybranego typu i następnie wybiera najlepszego sąsiada, o ile ten nie znajduję się na liście tabu (można wprowadzić mechanizm przełamania listy tabu jeśli sąsiad jest znacząco lepszy od najlepszego rozwiązania znalezionego do tej pory). Algorytm zawsze musi przejść do któregoś sąsiada, nawet jeśli jest gorszy niż aktualne rozwiązanie. Następnie ruch jest zabraniany na zadaną liczbę iteracji (kadencja). Algorytm możemy modyfikować zależnie od celu jaki chcemy uzyskać. Algorytm w trakcie działania może zmieniać kadencję: (1) zmniejszać, aby dokładniej przeszukiwać sąsiedztwo w okolicach optimum lokalnego (intensyfikacja) lub (2) zwiększać, aby wymusić szybsze wyjście z optimum lokalnego i przejść w inny obszar przeszukiwań (dywersyfikacja). Innym narzędziem dywersyfikacji jest metoda zdarzeń krytycznych, która wymuszą wygenerowanie nowego aktualnego rozwiązania (zazwyczaj losowego) po zadanej liczbie iteracji bez poprawy.

Algorithm 2 Tabu Search

```
1: π ← InitSolution()
2: tabuList.Clear();
3: for it = 1 to itLimit do
4:     C_best ← ∞
5:     for j = 1 to n do
6:         for k = j + 1 to n do
7:             if tabuList[j, k] < it then
8:                 π_new ← π.Move(i, j)
9:                 if Calculate(π_new) < C_best then
10:                    C_best ← Calculate(π_new)
11:                    j* ← j
12:                    k* ← k
13:                end if
14:            end if
15:        end for
16:    end for
17:    π ← π.Move(j*, k*)
18:    tabuList[j*, k*] ← it + cadance
19:    if Calculate(π) < Calculate(π*) then
20:        π* ← π
21:    end if
22: end for
```

3.3 Algorytm genetyczny

Algorytm metaheurystyczny inspirowany teorią ewolucji Darwina. Algorytm operuję na całej populacji rozwiązań. Do poprawnego działania koniecznie będzie napisanie własnej funkcji służącej do ewaluacji rozwiązania. Do rozwiązania należy użyć pakietu optymalizacyjnego DEAP. Pełna dokumentacja pakietu znajduje się pod tym linkiem.

opracował: *Radosław Idzikowski*