Wrocław, 11 listopada 2024

Programowanie **r**ównoległe i **r**ozproszone

Laboratorium 4

Programowanie równoległych metaheurystyk

prowadzący: *Dr inż. Radosław Idzikowski*

1 Wprowadzenie

Celem laboratorium jest projekt i implementacja wybranego algorytmu metaheurystycznego dla dowolnego problemu optymalizacji dyskretnej. Dopuszczalny jest dowolny język programowania, ale zaleca się użycie języka `C#` lub `Python`. Czas przewidziany na wykonanie zadania to termin dwóch zajęć wraz z ocenieniem pracy. Praca będzie oceniana na bieżąco w trakcie zajęć wraz z postępem wykonywania kolejnych przykładów. **Po ukończeniu każdego zadania należy zawołać prowadzącego w celu zaakceptowania etapu i odnotowania postępów.**

2 Zadania

W ramach zajęć należy w zespołach wykonać następujące zadania:

1. Podstawowy algorytm równoległy.
2. Zaawansowane techniki akceleracji.
3. Szczegółowa analiza efektywności.

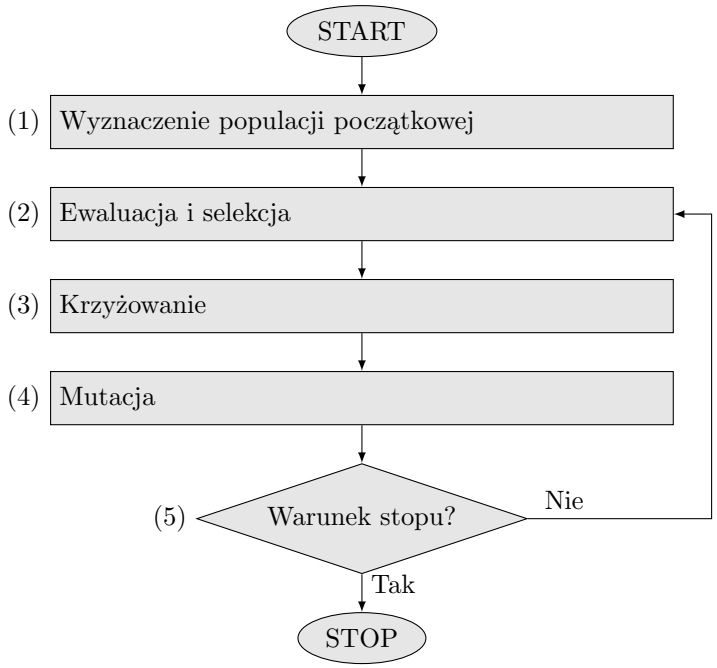
*. Implementacja zaawansowanych mechanizmów.

Za wykonanie zadania nr 1 jest ocena dostateczna, za każde kolejne zadanie jest +1 do oceny. Na ocenę bardzo dobrą (5.0) należy wykonać wszystkie trzy zadania. Zadanie oznaczone gwiazdką jest dla chętnych. Po zakończeniu zajęć należy kody źródłowe napisanych programów przesłać do prowadzącego.

3 Opis zadań

3.1 Zadanie 1

W ramach zadania napisać wybrany algorytm metaheurystyczny w wersji równoległej. Dobrym wyborem będzie m.in. (a) algorytm Genetyczny (*Genetic algorithm*, `GA`), (b) Sztuczna kolonia pszczół (*Artificial Bee Colony*, `ABC`) lub (c) przeszukiwanie z zabronieniami (*Tabu search*, `TS`). W przypadku algorytmów populacyjnych (`GA` oraz `ABC`) należy zrównoległość operację wykonywane na populacji. W przypadku `TS` równoległe można przeglądać sąsiedztwo. Cel zadania można osiągnąć przy użyciu dowolnej biblioteki (zarówno nisko jak i wysoko poziomowej). Po wcześniejszym uzgodnieniu z prowadzącym dopuszcza się również inne algorytmy. **UWAGA!** niedopuszczalne jest proste równoległe uruchomianie algorytmu (tzw. *multi-start*), obowiązkowe jest zrównoleglenie mechanizmu wewnątrz algorytmu. Algorytm może być zaimplementowany dla dowolnego problemu optymalizacji dyskretnej, najlepiej transportowego lub szeregowania zadań. Możliwy jest alternatywny sposób rozwiązania zadania - dla odpowiednio trudnego problemu optymalizacyjnego można równoległe policzyć wartość funkcji celu. Jednak nie dla każdego



Rysunek 1: Schemat blokowy `GA`

problemu ta ścieżka jest możliwa, ponieważ dla problemu komiwojażera istnieje prosta akceleracja, która pozwoli przeliczyć wartość funkcji celu w czasie stałym $O(1)$.

Dla przypomnienia algorytmy populacyjne w przeciwieństwie do algorytmów przeszukiwania lokalnego operują na zbiorze rozwiązań zwanym populacją. Znacząca grupa algorytmów populacyjnych to algorytmy ewolucyjne, a w szczególności algorytmy genetyczne. Wspólną cechą wszystkich algorytmów ewolucyjnych jest mechanizm rozwiązywania problemu inspirowany biologicznym procesem doboru naturalnego i ewolucji. `GA` został zaproponowany przez Johna Hollanda w latach 70 XX wieku. Na rysunku 1 pokazano schemat blokowy `GA`. Inną dużą grupą algorytmów populacyjnych są algorytmy oparte na inteligencji roju (mrówki, pszczoły itp.).

Krok 1 Pierwszym krokiem algorytmu `GA` jest wygenerowanie populacji początkowej. Każdy osobnik z populacji składa się z następujących elementów:

- *genotyp* – jeden lub więcej *chromosomów*, gdzie chromosom to ciąg kodowy genów reprezentujący rozwiązanie,
- *fenotyp* – wartość liczbowa odpowiadająca genotypowi, w tym wypadku wartość funkcji celu.

Gen to pojedynczy element (cecha), dla `TSP` jest to miasto do odwiedzenia. Poprzez *locus* oznacza się pozycje genu w chromosomie. Sposób generowania populacji początkowej ma istotny wpływ na efektywność `GA`. Zbyt podobna populacja początkowa można prowadzić do przedwczesnej zbieżności algorytmu. Konkretne osobniki można generować analogicznie jak w przypadku algorytmów lokalnego poszukiwania: (1) losowo lub (2) stosując heurystykę. W algorytmach wyspowych istnieje w tym samym czasie kilka odizolowanych od siebie populacji (każda na innej wyspie), a możliwość wymiany informacji między nimi jest możliwa jedynie co kilkanaście lub kilkadziesiąt pokoleń (iteracji). Wybór populacji o zbyt dużym rozmiarze będzie prowadził do znacznego wydłużenia czasu pracy `GA`.

Krok 2 Kolejnym krokiem algorytmu jest selekcja (wybór) bieżącej populacji rodziców, która będzie używana w procesie krzyżowania. Jeden (zazwyczaj dobrze przystosowany) osobnik

może zostać wybrany kilka razy, a inny słabszy ani razu. W literaturze najczęściej spotyka się następujące rodzaje selekcji:

1. Losowa (jednorodna) – każdy osobnik ma równe szansa bycia wybranym,
2. Ruletka – lepiej przystosowane osobniki mają większą szansę bycia wybranym niż słabsze (koło ruletki o zmieniaj szerokości przegródek),
3. Turniej – wybór najlepszego osobnika spośród k losowych osobników.

Należy pamiętać, że rozmiar populacji w trakcie czasu pracy algorytmu najczęściej jest stały. Każdą selekcję poprzedza ewaluacja (ocena) populacji.

Krok 3 Krzyżowanie jest procesem tworzenia osobników potomnych na bazie osobników rodzicielskich tak by powstałe „dzieci" były częściowo podobne do „rodziców" (jest to poniekąd odpowiednik sąsiadów w metaheurystykach poszukiwania lokalnego). Najczęściej przyjmuje się model w którym na wejściu jest para rodziców, a na wyjściu para dzieci, ale nie jest to jedyna możliwość. Generalnie procedurę tworzenia osobników potomnych nazywamy operatorem krzyżowania lub krócej krzyżowaniem. Krzyżowania mogą być jedno-, dwu- lub wielopunktowe. W krzyżowaniu zawsze jest przepisywania część genotypu jednego z rodziców, tzw. fragment dopasowania wyznaczony poprzez punkty podziału. Następnie bazując na informacji genetycznej drugiego rodzica należy wyznaczyć pozostałe geny. W przypadku niektórych problemów (np. chromosomy binarne) krzyżowanie jest zazwyczaj trywialne. Jednakże dla wielu problemów, w tym dla komiwojażera, osobnik potomny może reprezentować rozwiązanie niedopuszczalne (np. powstały ciąg nie jest permutacją). Dobrze zaprojektowany operator krzyżowania gwarantuje powstanie osobników reprezentujących rozwiązania dopuszczalne. Jeśli tak nie jest można zastosować dodatkowe procedury naprawcze.

Dla problemu komiwojażera popularne są następujące operatory krzyżowania:

1. Losowo (*Half Crossover*, `HX`),
2. Z zachowaniem porządku (*Order Crossover*, `OX`),
3. Z częściowym mapowaniem (*Partially Mapped Crossover*, `PMX`).

Wymieniono jedynie najbardziej popularne typy krzyżowania. W standardowym modelu drugi potomek powstaje przy założeniu tych samych punktów przecięcia, ale zamieniając rodziców miejscami. Niezależnie od typu krzyżowania powinno się zagwarantować, aby wybrano dwóch różnych rodziców, w celu uniknięcia tworzenia duplikatów danego osobnika. Ponadto zazwyczaj stosuje się prawdopodobieństwo zajścia krzyżowania, w przypadku niepowodzenia para jest pomijana lub przepisywana do pokolenia dzieci. W niektórych implementacjach po etapie krzyżowania istnieje możliwość powstania większej liczby dzieci niż wielkość populacji. Wtedy nadmiar zostaje odrzucony w trakcie procesu selekcji. Można się spotkać również z dedykowanymi krzyżowaniami wyłącznie dla danego problemu.

Krok 4 Z pewnym prawdopodobieństwem każdy osobnik może zostać poddany mutacji. Poprzez mutację rozumiane jest zazwyczaj losowe zaburzenie. Celem mutacji jest zwiększenie różnorodności genetycznej oraz dodanie do populacji genów obecnie w niej niereprezentowanych. Przeciwdziała to przedwczesnej zbieżności algorytmu. Prawdopodobieństwo mutacji jest zwykle kilka lub kilkunastokrotnie mniejsze niż prawdopodobieństwo krzyżowania. Mutację najczęściej wykonuje się poprzez wykonanie ruchu znanego z algorytmów lokalnego poszukiwania (zamień, wstaw, odwróć itd.).

W przypadku tzw. algorytmów memetycznych wykonywana jest dodatkowa procedura. Algorytmy te bowiem nie bazują na genach (informacja genetyczna, zależna wyłącznie od genomu rodziców), ale na memach (informacja kulturowa/środowiskowa, zależna od środowiska w którym osobnik „dorasta"). Przykładowo, umiejętności pływania czy mówienia po polsku nie mają podłoża genetycznego, a mimo to dzieci rodziców posiadających te umiejętności zwykle same je

posiadają. Wspomniana procedura jest więc niejako symulacją „dorastania/uczenia się" osobnika. Typowym sposobem realizacji takiej metody jest zastosowanie metaheurystyki poszukiwania lokalnego (np. `TS`) lub innej heurystyki (np. 2-OPT z uciętym czasem trwania) z danym osobnikiem jako rozwiązaniem początkowym. Procedura taka nie powinna jednak działać zbyt długo.

Krok 5 W algorytmach populacyjnych każdą iterację algorytmu nazywamy pokoleniem. Poza tym możliwe warunki stopu są w zasadnie analogiczne jak przy algorytmach lokalnego poszukiwania, czyli liczba iteracji lub limit czasu.

3.2 Zadanie 2

Algorytm z zadania pierwszego należy rozbudować o mechanizmy poznane w trakcie semestru. W celu zwiększenia wydajności algorytmu można zastosować mechanizmy wykorzystujące programowanie rozproszone lub akcelerację przy użyciu obliczeń na `GPU`.

3.3 Zadanie 3

Dla napisanego algorytmu należy przeprowadzić szczegółową analizę efektywności w zależności od rozmiaru problemu oraz wydajności w zależności od użytej techniki obliczeń równoległych. Efektem analizy powinny być wykresy oraz wnioski.

3.4 Zadanie *

Implementacja zaawansowanych mechanizmów opartych na własnościach badanego problemu. Dla problemów szeregowania zadań warto zastosować własności blokowe (np.: `TSAB`) lub analogicznie bazujące na wzorcach dla problemów transportowych. Innym ciekawym ulepszeniem jest zastosowanie akceleracji liczenia funkcji celu, jeśli taka jest znana dla badanego problemu.