Porównanie algorytmów optymalizacyjnych na przykładzie problemu pakowania do pojemników

Informatyka i Ekonometria AGH

Studia II stopnia

Mateusz Mulka, Maciej Nagły, Krzysztof Księżyc

Wstęp teoretyczny

Problem pakowania do pojemników (ang. *Bin Packing Problem*, BPP) to klasyczne zagadnienie optymalizacyjne, polegające na takim rozmieszczeniu elementów o różnych rozmiarach w pojemnikach o ustalonej pojemności, aby użyć możliwie najmniejszej liczby tych pojemników. Każdy pojemnik ma ograniczoną pojemność, a suma rozmiarów elementów w nim umieszczonych nie może jej przekroczyć.

Problem ten należy do grupy problemów NP-trudnych, co oznacza, że wraz ze wzrostem liczby elementów rośnie znacząco trudność znalezienia rozwiązania optymalnego. Dlatego w praktyce często stosuje się podejścia przybliżone lub heurystyczne, które pozwalają na uzyskanie dobrych (choć niekoniecznie optymalnych) rozwiązań w rozsądnym czasie.

BPP znajduje zastosowanie w wielu dziedzinach, takich jak logistyka (załadunek towarów), produkcja (cięcie materiałów), informatyka (zarządzanie pamięcią czy przydział zasobów), a także w różnych systemach automatyzacji.

W ramach projektu porównane zostaną trzy różne algorytmy rozwiązujące problem pakowania do pojemników. Wszystkie algorytmy będą testowane na tych samych danych wejściowych z ustalonym z góry seedem losowym, co zapewnia porównywalność wyników. Dane zawierają 1000 elementów w proporcjach 60% małe (rozmiar 5-29) oraz 40% duże (rozmiar 80-109), natomiast rozmiar pojemników wynosi 180. Dzięki takiemu doborowi danych wejściowych, została zapewniona wystarczająca trudność problemu, aby móc oczekiwać realnej poprawy wyniku. Ocena skuteczności będzie opierać się na dwóch metrykach: poprawie rozwiązania (różnicy liczby pojemników między pierwszą a ostatnią iteracją) oraz czasie wykonania danego algorytmu.

Celem projektu jest zbadanie, które podejście daje najlepsze wyniki w zadanym czasie oraz jak poszczególne algorytmy radzą sobie z tym samym zestawem danych.

Algorytm roju cząstek (PSO)

Algorytm roju cząsteczek (*Particle Swarm Optimization*, PSO) to populacyjna metoda optymalizacji inspirowana zachowaniem zbiorowym zwierząt, takich jak insekty, ptaki czy ryby. W PSO grupa potencjalnych rozwiązań, nazywana rojem, przeszukuje przestrzeń rozwiązań w poszukiwaniu najlepszego rozwiązania danego problemu. Każde rozwiązanie reprezentowane jest przez cząstkę, która zapamiętuje zarówno swoje najlepsze dotychczasowe rozwiązanie, jak i zna najlepsze rozwiązanie wśród całej populacji.

W klasycznym PSO cząstki poruszają się w przestrzeni rozwiązań, modyfikując swoje pozycje i prędkości w zależności od własnego doświadczenia oraz doświadczenia całego roju.

Ruch każdej cząstki uzależniony jest od trzech komponentów:

- **inercji** (utrzymania kierunku ruchu z poprzednich iteracji),
- **komponentu kognitywnego** (przyciągania do najlepszego własnego rozwiązania),
- **komponentu socjalnego** (przyciągania do najlepszego rozwiązania całego roju).

PSO jest szeroko stosowane do problemów optymalizacji ciągłej i dyskretnej, a jego główne zalety to prostota implementacji, niewielka liczba parametrów oraz dobra zdolność eksploracji przestrzeni rozwiązań.

Implementacja w projekcie

W projekcie zastosowano dyskretną wersję algorytmu roju cząsteczek do rozwiązania problemu pakowania do pojemników. Każda cząstka reprezentuje permutację elementów, a jej jakość oceniana jest na podstawie liczby pojemników potrzebnych do zapakowania elementów według tej permutacji. Do oceny użyto heurystyki First Fit.

Cząstki modyfikują swoje pozycje poprzez zaplanowane zamiany elementów, dążąc do najlepszego znanego rozwiązania — zarówno własnego, jak i globalnego w roju. W każdej iteracji ruchy te są ograniczane do maksymalnie 20 zamian, aby kontrolować intensywność przeszukiwania.

Dla porównywalności wyników, algorytm działał na wspólnym zbiorze danych z ustalonym seedem losowym.

Parametry algorytmu

- Liczba cząstek: **80**
- Liczba iteracji: **200**
- Współczynnik bezwładności (inercji): **0.5**
- Współczynnik przyciągania do najlepszego własnego rozwiązania: **1.0**
- Współczynnik przyciągania do najlepszego globalnego rozwiązania: **1.0**
- Maksymalna liczba zamian pozycji w jednej iteracji: **20**

Wyniki działania algorytmu

- Liczba użytych pojemników w pierwszej iteracji: **273**
- Liczba użytych pojemników w ostatniej iteracji: **267**
- Czas wykonania: **511.76 sekundy**

Algorytm osiągnął poprawę o **6** pojemników względem stanu początkowego, co wskazuje na skuteczne przeszukiwanie przestrzeni rozwiązań, choć przy relatywnie wysokim koszcie czasowym.

Algorytm genetyczny (Genetic Algorithm, GA)

Algorytm genetyczny to populacyjna metoda optymalizacji inspirowana mechanizmami ewolucji biologicznej, takimi jak selekcja naturalna, krzyżowanie (rekombinacja) i mutacja. W GA każda jednostka w populacji (osobnik) reprezentuje potencjalne rozwiązanie problemu, a jego jakość określana jest za pomocą funkcji dopasowania (fitness function).

Proces ewolucji w GA polega na wielokrotnym generowaniu nowych populacji przez wybór najlepszych osobników, a następnie stosowanie do nich operatorów genetycznych. W rezultacie populacja stopniowo zbliża się do optymalnego rozwiązania.

W klasycznym GA stosuje się trzy główne komponenty:

- Selekcja – wybór najlepszych osobników do reprodukcji (np. selekcja turniejowa),
- Krzyżowanie (crossover) – tworzenie nowych rozwiązań poprzez łączenie fragmentów „rodziców” (np. order crossover),
- Mutacja – wprowadzanie losowych zmian w rozwiązaniach w celu zwiększenia różnorodności genetycznej.

GA znajduje szerokie zastosowanie w problemach optymalizacji kombinatorycznej, takich jak pakowanie, harmonogramowanie czy optymalizacja tras. Główne zalety to elastyczność w doborze reprezentacji, odporność na lokalne minima i dobre wyniki przy odpowiednim dostrojeniu parametrów.

Implementacja w projekcie

W projekcie zastosowano klasyczny algorytm genetyczny do rozwiązania problemu pakowania elementów do pojemników. Każde rozwiązanie reprezentowane jest jako permutacja indeksów elementów, która następnie jest dekodowana za pomocą heurystyki First Fit.

W każdej generacji algorytm:

- ocenia wszystkie osobniki na podstawie liczby potrzebnych pojemników,
- wybiera pary rodziców przy użyciu selekcji turniejowej,
- generuje nowe osobniki przez order crossover,
- poddaje potomstwo mutacjom (losowe zamiany elementów),
- aktualizuje populację na podstawie potomków.

Proces ten powtarzany jest przez ustaloną liczbę generacji. W każdej iteracji monitorowana jest najlepsza znaleziona permutacja oraz jej ocena.

Dla zachowania porównywalności wyników, algorytm korzystał z tego samego zbioru danych wejściowych i zadanego ziarna losowego jak w przypadku PSO.

Parametry algorytmu

- Liczba osobników w populacji: 100
- Liczba generacji: 200
- Prawdopodobieństwo krzyżowania: 0.9
- Prawdopodobieństwo mutacji: 0.05
- Rozmiar turnieju selekcyjnego: 3

Wyniki działania algorytmu

- Liczba użytych pojemników w pierwszej generacji: **271**
- Liczba użytych pojemników w ostatniej generacji: **269**
- Czas wykonania: **407** sekund

Algorytm genetyczny wykazał skuteczność w optymalizacji rozmieszczenia elementów, poprawiając wynik względem rozwiązania początkowego. Dzięki kombinacji selekcji, krzyżowania i mutacji możliwe było efektywne przeszukiwanie przestrzeni rozwiązań, co przekłada się na zadowalający rezultat końcowy przy umiarkowanym czasie wykonania.

Algorytm wyżarzania symulowanego (Simulated Annealing, SA)

Algorytm wyżarzania symulowanego (Simulated Annealing, SA) to metoda optymalizacji inspirowana procesem fizycznym wyżarzania metali, w którym materiał jest powoli chłodzony, by osiągnąć stan o minimalnej energii. W kontekście optymalizacji, SA pozwala na akceptowanie gorszych rozwiązań z pewnym prawdopodobieństwem, co umożliwia wyjście z lokalnych minimów i znalezienie lepszych globalnych rozwiązań.

Algorytm rozpoczyna się od losowego rozwiązania, a następnie w każdej iteracji generuje jego niewielką modyfikację (np. zamianę dwóch elementów). Jeśli nowe rozwiązanie jest lepsze, jest zawsze akceptowane. Jeśli jest gorsze, może zostać zaakceptowane z

prawdopodobieństwem zależnym od aktualnej temperatury, która stopniowo maleje według schematu schładzania.

Simulated Annealing jest stosunkowo prosty do implementacji i dobrze nadaje się zarówno do problemów dyskretnych, jak i ciągłych.

Implementacja w projekcie

W projekcie zastosowano dyskretną wersję algorytmu wyżarzania symulowanego do rozwiązania problemu pakowania przedmiotów do pojemników. Rozwiązanie reprezentowane jest przez permutację elementów, której jakość oceniana jest na podstawie liczby pojemników potrzebnych do spakowania przedmiotów zgodnie z heurystyką First Fit.

W każdej iteracji algorytmu losowo zamieniane są dwa elementy w permutacji. Nowe rozwiązanie jest akceptowane zgodnie z zasadami wyżarzania symulowanego, co umożliwia eksplorację przestrzeni rozwiązań i unikanie utknięcia w lokalnych minimach.

Dla zapewnienia porównywalności wyników, algorytm został uruchomiony na wspólnym zbiorze danych z ustalonym seedem losowym.

Parametry algorytmu

- Liczba iteracji: 10 000
- Początkowa temperatura: 100,0
- Współczynnik chłodzenia: 0,995

Wyniki działania algorytmu

- Liczba użytych pojemników na początku (iteracja 0): 277
- Liczba użytych pojemników po 1000 iteracjach: 271
- Liczba użytych pojemników po 2000 iteracjach: 269
- Liczba użytych pojemników po 3000 iteracjach: 268
- Liczba użytych pojemników po 4000–9000 iteracjach: 268
- Liczba użytych pojemników na końcu: 268
- Czas wykonania: 191,53 sekundy

Algorytm uzyskał poprawę o 9 pojemników względem stanu początkowego, co świadczy o jego skuteczności w poszukiwaniu lepszych rozwiązań poprzez wyważoną eksplorację i eksploatację przestrzeni rozwiązań.

W porównaniu do algorytmu roju cząstek (PSO), Simulated Annealing osiągnął nieco lepszy wynik końcowy przy znacząco krótszym czasie wykonania, co wskazuje na jego wysoką efektywność w tego typu problemie.

Porównanie wyników

W projekcie porównano trzy różne podejścia do rozwiązania problemu pakowania przedmiotów do pojemników: Algorytm roju cząstek (PSO), Algorytm genetyczny (GA) oraz Algorytm wyżarzania symulowanego (SA). Wszystkie algorytmy były testowane na tym samym zbiorze danych wejściowych przy ustalonym seedzie losowym.

Algorytm	Liczba użytych pojemników	Czas wykonania (sekundy)	Uwagi
PSO	267	511,76	Dobra poprawa wyniku, długi czas wykonania
GA	269	407,00	Poprawa przy umiarkowanym czasie działania
SA	268	191,53	Najkrótszy czas, skuteczna optymalizacja

Algorytm roju cząstek (PSO) osiągnął najlepszy wynik liczby pojemników (267), ale jego czas wykonania był ponad dwukrotnie dłuższy niż w przypadku SA.

Algorytm genetyczny (GA) poprawił wynik, ale uzyskał wynik końcowy gorszy niż PSO i podobny do SA, przy średnim czasie wykonania.

Algorytm wyżarzania symulowanego (SA) uzyskał wynik minimalnie gorszy niż PSO (268 pojemników), jednak czas jego działania był ponad dwa razy krótszy, co świadczy o dużej efektywności tego podejścia.

Podsumowując, **Simulated Annealing** okazał się najbardziej wydajnym algorytmem w kontekście kompromisu między jakością rozwiązania a czasem wykonania, natomiast **PSO** uzyskał najlepszy wynik końcowy kosztem znacząco większego czasu działania.

Analiza różnic w wynikach

Różnice w jakości rozwiązań i czasie działania algorytmów wynikają bezpośrednio z charakterystyki ich działania oraz strategii przeszukiwania przestrzeni rozwiązań:

PSO intensywnie eksploruje przestrzeń rozwiązań, ponieważ każda cząstka stara się jednocześnie dążyć do swojego najlepszego rozwiązania i najlepszego rozwiązania globalnego. Powoduje to skuteczne poszukiwanie lepszych rozwiązań, ale wymaga dużej liczby operacji i synchronizacji całego roju, co znacząco zwiększa czas wykonania.

GA opiera się na operacjach genetycznych (selekcji, krzyżowaniu i mutacji), które zapewniają szeroką eksplorację przestrzeni rozwiązań. Jednak proces tworzenia nowych pokoleń jest stosunkowo kosztowny obliczeniowo i nie zawsze gwarantuje szybkie dojście do globalnego optimum, co tłumaczy umiarkowany czas działania i wynik końcowy.

SA stosuje lokalne zmiany (np. zamianę dwóch elementów) i dzięki mechanizmowi probabilistycznego akceptowania gorszych rozwiązań szybko ucieka z lokalnych minimów. Ze względu na brak operacji na całych populacjach, SA ma bardzo niskie koszty pojedynczej

iteracji, co przekłada się na najkrótszy czas wykonania przy zachowaniu wysokiej jakości wyniku.

Podsumowując, **PSO** zapewnia najdokładniejsze wyniki kosztem czasu, **GA** oferuje kompromis pomiędzy jakością a czasem, a jego wyniki często zależą od krzyżówek, natomiast **SA** jest najszybszy i najbardziej efektywny w krótkim czasie, choć może nie osiągnąć absolutnie najlepszego wyniku.