Optymalizacja wielokryterialna

Metody ewolucyjne

1 Wstęp

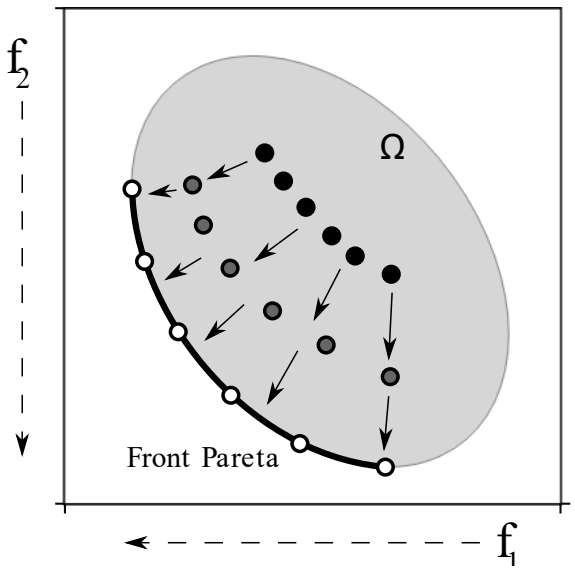
Heurystyki są to algorytmy, które wykorzystują pewne przybliżone, zachłanne, bądź probabilistyczne procedury do rozwiązania problemu optymalizacji. Algorytmy te **nie gwarantują odnalezienia optimum**, jednak często oferują one uzyskanie **bardzo dobrych wyników w stosunkowo krótkim czasie**, czyniąc je doskonałą alternatywą do algorytmów dokładnych, np. opartych o programowanie liniowe.

Algorytmy ewolucyjne są to takie heurystyki, które jak sama nazwa wskazuje, **naśladują naturalne procesy zachodzące w przyrodzie**, takie jak np. wykorzystanie populacji rozwiązań, reprodukowanie ich oraz nadawanie presji selekcyjnej. W kontekście problemów mających charakter wielokryterialny, algorytmy ewolucyjne zyskały na przestrzeni lat znacząco na znaczeniu dzięki ich populacyjnej naturze.

Fundusze Europejskie Polska Cyfrowa | Rzeczpospolita Polska | Unia Europejska Europejski Fundusz Rozwoju Regionalnego

„Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”, projekt finansowany ze środków Programu Operacyjnego Polska Cyfrowa POPC.03.02.00-00-0001/20

Po pierwsze dzięki ewoluowaniu całego zbioru rozwiązań, algorytmy takie mogą skonstruować przybliżenie frontu Pareta podczas ich jednokrotnego uruchomienia. Ponadto, dzięki wymianie informacji między osobnikami w populacji, dokonanej poprzez reprodukcję rozwiązań, algorytmy te skutecznie minimalizują liczbę kroków wymaganą do przybliżenia frontu Pareta. Przykład przebiegu takiej ewolucji znajduje się na Rysunku 1, gdzie kolorem ciemnym zaznaczono populację rozwiązań na wczesnym etapie ewolucji, kolorem szarym na etapie pośrednim, białym na etapie końcowym.



Rysunek 1: Przykładowy przebieg procesu ewolucji w kontekście optymalizacji wielokryterialnej

2 Algorytm NSGA-II

Klasycznym algorytmem ewolucyjnym do rozwiązywania problemów optymalizacji wielokryterialnej jest algorytm NSGA-II. Kolejne punkty tego

rozdziału mają za zadanie wyjaśnić zasadę działania tego algorytmu. Punkty te będą często odwoływać się do 2-kryterialnego problemu **postawionego w zadaniu domowym.**

Problem do rozwiązania w zadaniu domowym jest zdefiniowany następująco:

$$\min f_1(\boldsymbol{x}) = x_1^{10} l,$$

$$\min f_2(\boldsymbol{x}) = (1 - x_1^{10}) l,$$

$$l = 1 + |x_2^{10} - 0.5| + x_3 + x_4 + x_5,$$

$$\text{gdzie } x_1, x_2 \in |0,1|, x_3, x_4, x_5 \in \{0, 1\}.$$

Istnieje więc w problemie 5 zmiennych decyzyjnych, z czego pierwsze dwie są ciągłe, a pozostałe trzy binarne. Jest to sztucznie utworzony problem, którego głównym celem jest uwypuklenie pewnych problemów na jakie może natrafić algorytm optymalizujący.

Powyższy problem składa się tak naprawdę z dwóch członów: część określająca pozycję na froncie Pareta oraz część określająca odległość od niego. Odległość jest zdefiniowana przez czynnik l występujący w definicjach obydwu funkcji celu. Łatwo więc zauważyć, że rozwiązanie $\boldsymbol{x}$ jest – tzn. jego wektor ocen – Pareto optymalne, jeżeli l przyjmuje wartość minimalną. Dzieje się to wtedy, gdy $x_2 = 0.5$, $x_3 = 0$, $x_4 = 0$, $x_5 = 0$. Zmienna x_1 może natomiast przyjmować dowolną wartość w zależności, od której różne rozwiązania Pareto optymalne będą uzyskiwane. Pewnym problemem dla algorytmu optymalizującego mogą być wysokie wykładniki występujące przy zmiennych x_1 oraz x_2. Powodują one „spychanie" podstawionych wartości do zera.

Uwaga: Przedstawiony wyżej problem jest silnie nieliniowy. Tym samym pozornie nie można go rozwiązać wykorzystując programowanie liniowe, jednak

często istnieje wiele „trików” pozwalających przekształcić nieliniowy problem w liniowy. Tutaj wystarczyło by dokonać podstawienia $x_1' = x_1^{10}$ oraz $x_2' = x_2^{10}$. Po uzyskaniu rozwiązania można powrócić do oryginalnych zmiennych poprzez pierwiastkowanie. Zmienne binarne x_{3-5} nie stanowią większego problemu dla optymalizatora ze względu na ich znikomą liczbę. Często także takie zmienne mogą być traktowane jako ciągłe, bez zmiany optymalności uzyskiwanych rozwiązań.

i. Ustanowienie reprezentacji rozwiązań

Ważnym krokiem podczas implementacji algorytmu ewolucyjnego jest ustanowienie reprezentacji rozwiązań w przestrzeni decyzji. Warto podkreślić, że heurystyki dają większą swobodę na tym etapie, niż np. metody oparte o programowanie liniowe. W kontekście analizowanego problemu, można przyjąć bezpośredni genotyp, tzn. reprezentację osobnika jako wektor 5 zmiennych decyzyjnych x_{1-5}. W takim przypadku, każda zmienna decyzyjna może przyjąć jedną z dozwolonych wartości, tzn. pierwsze dwie zmienne są ciągłe od 0 do 1, pozostałe trzy zmienne są natomiast binarne.

ii. Utworzenie populacji początkowej

Niech P_0 oznacza populację początkową a N rozmiar takiej populacji. Do utworzenia P_0 można wykorzystać pewne dedykowane problemowi metody, jednak na potrzeby tego laboratorium wystarczy rozważyć procedurę losową. Innymi słowy wartość każdej zmiennej decyzyjnej osobnika zostanie wylosowana z powiązanej z nią dziedziny. Następnie mając tak przygotowany zbiór N

losowych rozwiązań, należy je ocenić z punktu widzenia kryteriów. W przypadku problemu w zadaniu domowym, są to kryteria f_1 oraz f_2.

iii. Nadanie presji selekcyjnej

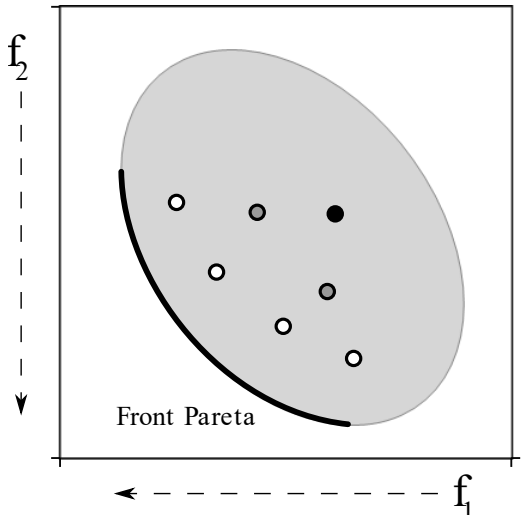
Ten etap ma za zadanie wskazać te rozwiązania, które są korzystne z punktu widzenia próby przybliżenia frontu Pareta. Klasyczne algorytmy ewolucyjne, takie jak NSGA-II, realizują ten etap poprzez odpowiednie posortowanie populacji. Takie sortowanie jest na ogół dwuetapowe. W NSGA-II po pierwsze rozwiązania są sortowane poprzez przydzielenie ich do tak zwanych **frontów niezdominowanych**. Ten etap sortowania ma za zadanie promować rozwiązania, które są bliżej frontu Pareta. Dalej, rozwiązania zawarte w tych samych niezdominowanych frontach są sortowane według miary ***crowding distance***. Miara ta ma za zadanie promować dobrze rozmieszczone rozwiązania, starając się uzyskać równomierny rozkład rozwiązań przybliżających front Pareta. Obydwa kryteria sortowania są realizowane następująco:

- **Przydział do frontów niezdominowanych**: Niech P_l oznacza populację w l-tej iteracji działania algorytmu. Procedura rozpoczyna się od identyfikacji rozwiązań niezdominowanych w P_l i przedzielenia ich do pierwszego frontu F_l. Innymi słowy:

  $$F_1 = \{s^j : P_l : \neg\exists_{s^k \in P_l, s^k \neq s^j} \; s^k \Delta s^j \}.$$

  Jeżeli jakieś rozwiązania w P_l nie zostały przydzielone do F_1, algorytm konstruuje kolejny front F_1 w analogiczny sposób uwzględniając nieprzypisane jeszcze rozwiązania (w tym przypadku $P_l \backslash F_1$). Proces ten jest powtarzany do momentu aż każde rozwiązanie zostanie przypisane do jakiegoś frontu. Algorytm NSGA-II zawsze promuje rozwiązania znajdujące

się w niższych frontach. Dzięki czemu nadawana jest presja selekcyjna „przesuwająca” populację w stronę frontu Pareta. Przykładowy przydział do niezdominowanych frontów znajduje się na Rysunku 2, gdzie białe kropki reprezentują rozwiązania w F_1, szare w F_2 a czarne w F_3. Dla każdego rozwiązania w wyższym froncie istnieje przynajmniej jedno rozwiązanie w froncie niższym, które je dominuje. Natomiast rozwiązania w tym samym froncie są wzajemnie niezdominowane.



Rysunek 2: Przykładowy przydział rozwiązań do frontów niezdominowanych

- **Obliczenie wartości crowding distance:** Promowanie rozwiązań wykorzystując tylko konstrukcję frontów niezdominowanych nie zagwarantuje jeszcze, że populacja będzie ewoluowała w taki sposób, że będzie pokrywać front Pareta równomiernie. By wymusić na populacji taki rozkład, NSGA-II oblicza dla każdego rozwiązania wartość crowding distance, która określa poziom „zagęszczenia” rozwiązań wokół niego. Obliczenia te są wykonywane dla każdego z niezdominowanych frontów z osobna, nie dla całej populacji. Innymi słowy, podczas obliczania wartości

crowding distance rozwiązań w jednym froncie, ignoruje się rozwiązania w innych frontach.

Niech s_i^j oznacza ocenę rozwiązania s^j w froncie F na i-tym kryterium. Dalej, niech s_i^- i s_i^+ oznaczają oceny na i-tym kryterium rozwiązań bezpośrednio sąsiadujących – tzn. jedna realizacja powinna być lepsza a druga gorsza. Wtedy miara crowding distance dla s^j jest obliczana następująco:

$$cd(s^j) = \sum_{i=1}^{M} (s_i^+ - s_i^-).$$

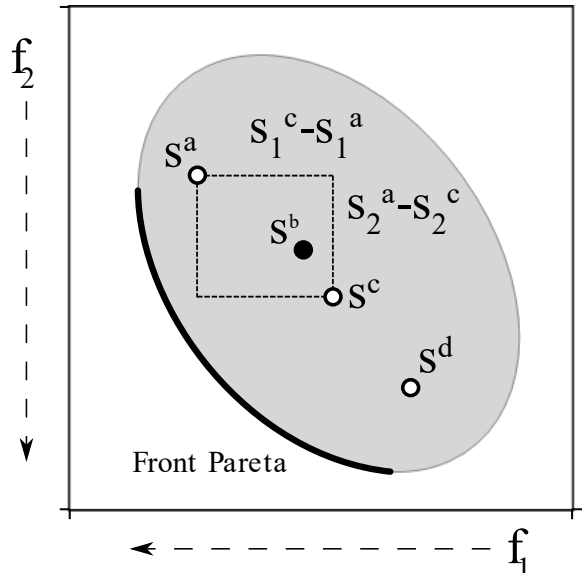
Im większa wartość $cd(s^j)$, tym rozwiązanie s^j jest bardziej promowane. Należy tutaj zwrócić uwagę na to, że oceny rozwiązań na kryteriach powinny być uprzednio odpowiednio znormalizowane. W sytuacji, w której s^j jest rozwiązaniem granicznym z punktu widzenia pewnego kryterium, tzn., nie istnieje w froncie rozwiązanie lepsza lub gorsze, $cd(s^j)$ jest ustanowione na ∞. Przykładowe wyznaczenie wartości $cd(s^j)$ znajduje się na Rysunku 3. Na ilustracji zaznaczono przykładowe rozwiązanie s^{b}, dla którego miara crowding distance jest obliczana w oparciu o rozwiązania s^{a} i s^{c}:

$$cd(s^b) = (s_1^c - s_1^a) + (s_2^a - s_2^c).$$

Dla przykładu, miara crowding distance dla s^{a} i s^{c} wynosi ∞, ponieważ są to rozwiązania graniczne.

Obliczenie wartości crowding distance wymaga posortowania populacji M razy (dla każdego kryterium osobna). Jednak szczególnym przypadkiem jest sytuacja, w której $\mathrm{M} = 2$. Wtedy wystarczy posortować rozwiązania używając tylko jednego kryterium. Jako że rozwiązania pochodzą z jednego

niezdominowanego frontu, posortowanie ich wykorzystując np. f_1, automatycznie posortuje je z punktu widzenia f_2 (rozwiązanie, które jest najlepsze na f_1 jest jednocześnie najgorsze na f_2 itd.).



Rysunek 3: Przykład wyznaczenia wartości crowding distance

iv. Konstrukcja rozwiązań potomnych

Algorytm NSGA-II jest algorytmem generacyjnym. Oznacza to, że w jednej iteracji algorytmu – także zwanej generacją – aktualizuje on całą populację rozwiązań, tzn. tworzy on populację potomną o rozmiarze populacji wejściowej N (istnieją także algorytmy „stacjonarne", które podczas pojedynczej iteracji generują tylko jedno rozwiązanie potomne). Możliwych schematów postępowania na tym etapie jest wiele. Poniżej zostanie przedstawiony sposób najbardziej podstawowy, wykorzystujący pary rozwiązań do konstrukcji rozwiązań potomnych.

Populacja P_o jest budowana przy wykorzystaniu obecnych rozwiązań – „rodziców" – w populacji P_l. By zwiększyć szansę na konstrukcję lepszego potomstwa, do

reprodukcji powinny być wybierane rozwiązania zajmujące wysokie miejsca w posortowanej populacji P_l. Często do wyboru takich par rodziców wykorzystuje się selekcję turniejową. Zakładając, że populacja P_l jest posortowana według frontów niezdominowanych i wartości crowding distance, turniej o rozmiarze k polega na wylosowaniu k rozwiązań z P_l. Losowanie może być z zwracaniem lub bez. Zwycięzcą turnieju jest rozwiązanie zajmujące lepszą pozycję w posortowanej populacji. W taki sposób zostaje wybrane jedno rozwiązanie „rodzic". Jako że należy wygenerować N rozwiązań potomnych wykorzystując pary rodziców, taki turniej winien być powtórzony $2N$ razy. W zależności od wybranego k, zwiększana bądź zmniejszana może być „zachłanność" selekcji. Dla $\mathrm{k} = 1$ wybór jest z rozkładu jednorodnego. Im większe k, tym większa szansa wyboru rozwiązań zajmujących wysokie pozycje w posortowanej populacji P_l.

Mając N par rozwiązań-rodziców, algorytm konstruuje N rozwiązań potomnych. Najczęściej w tym celu wykorzystuje się dwa **operatory genetyczne: krzyżowania i mutacji**. Pierwszy operator ma za zadanie połączyć dwa rozwiązania rodzicielskie w jeden genotyp rozwiązania potomnego, potencjalnie łącząc ich najlepsze cechy, realizując tym samym **paradygmat eksploatacji**. Operator mutacji ma z kolei za zadnie dokonać niewielkiej perturbacji tak utworzonego genotypu potomnego. W ten sposób realizowany jest **paradygmat eksploracji**, potencjalnie umożliwiający na ucieczkę z optimów lokalnych.

Poniżej przedstawiono dwa przykładowe operatory krzyżowani i mutacji:

- **Operator krzyżowania z pojedynczym punktem krzyżowania**: Niech $\boldsymbol{x}^j$ i $\boldsymbol{x}^k$ oznaczają wektory decyzyjne dwóch rodziców. Niech p oznacza losową liczbę całkowitą oznaczającą punkt cięcia. Wtedy wektor decyzyjny

rozwiązania potomnego jest konstruowany w ten sposób, że pierwsze p wartości jest pobrane od $\boldsymbol{x}^j$, a pozostałe od $\boldsymbol{x}^k$.

- **Operator mutacji**: Operatory mutacji są na ogół dużo prostsze do zaimplementowania. Często uznaje się, że każda zmienna ma szansę zmutować z szansą $1/n$, gdzie n jest liczbą zmiennych decyzyjnych. Innymi słowy, średnio jedna zmienna ma szansę zmutować. W przypadku zmiennych binarnych taką mutacją może być prosta zamiana z 0 na 1, i na odwrót. W przypadku zmiennych ciągłych można wykorzystać np. mutację gaussowską. Polega ona na dodaniu do obecnej wartości pewnej wartości odchylającej, pochodzącej z rozkładu normalnego o ustalonym odchyleniu standardowym. Należy jednak zabezpieczyć procedurę przed wyjściem tak zaktualizowanej wartości poza dozwolony zakres – dziedzinę.

v. Zastosowanie reguły „przetrwają najlepiej przystosowani"

Na tym etapie algorytm przechowuję obecną populację P_l oraz populację potomną P_o. NSGA-II jest algorytmem elitystycznym, co oznacza, że najlepsze rozwiązania z obydwu populacji mogą przejść do kolejnej iteracji. W tym celu tworzona jest populacja łączna $P_M = P_l \cup P_o$. Dalej jest ona sortowana w ten sam sposób jak w punkcie 2. Ostatecznie po posortowaniu, pierwsze N najlepszych rozwiązań w P_M przechodzi do kolejnej generacji algorytmu, stanowiąc populację P_{l+1}.

vi. Kolejna iteracja

Należy powtórzyć kroki iii-v, dopóki kryteria stopu nie zostaną osiągnięte. Najczęściej jest to limit na liczbę generacji algorytmu bądź skonstruowanych rozwiązań.

3 Zadanie domowe

Zadaniem domowym jest implementacja algorytmu NSGA-II w języku Python do rozwiązania uprzednio zdefiniowanego problemu. Szablon z częściowo uzupełnionymi funkcjami jest dostępny w pliku **Homework_NSGA-II.ipynb**. Skrypt został oparty o wersję języka Python 3.8, bibliotekę Numpy w wersji 1.19 oraz bibliotekę Matplotlib w wersji 3.3. Jest to plik typu **Jupyter Notebook**. Wszystkie polecenia są zawarte w tym skrypcie. Pomocniczo dołączony jest powyższy plik w formacie HTML. Praca przy wykorzystaniu Jupyter Notebooka nie jest wymagana **(ale zalecana)** – można powyższe skrypty przekopiować do regularnego skryptu *.py.

Fundusze Europejskie Polska Cyfrowa | Rzeczpospolita Polska | Unia Europejska Europejski Fundusz Rozwoju Regionalnego

„Akademia Innowacyjnych Zastosowań Technologii Cyfrowych (AI Tech)”, projekt finansowany ze środków Programu Operacyjnego Polska Cyfrowa POPC.03.02.00-00-0001/20