Wydział Podstawowych Problemów Techniki
Politechnika Wrocławska

Replikacja obrazów za pomocą algorytmów genetycznych

Jakub Gogola
Nr indeksu: 236412

Praca inżynierska napisana
pod kierunkiem
dra Piotra Sygi

Politechnika
Wrocławska
Wrocław 2019

Spis treści

Wstęp

We wstępie pracy zostanie przedstawiona pokrótce idea algorytmów metaheurystycznych oraz krótki ich krótki rys historyczny z wyróżnieniem algorytmów genetycznych, które są tematem niniejszej pracy inżyierskiej.

Analiza problemu

W niniejszym rozdziale zostanie zdefiniowany problem replikacji obrazów rozważany w pracy. Przedstawiona zostanie również koncepcja i założenia algorytmów stosowanych przez autora do testowania oraz praktycznego zobrazowania analizowanego problemu.

2.1 Replikacja obrazów

Replikacją obrazów można określić proces, podczas którego oryginalny (wzorcowy) obraz jest reprodukowany za pomocą pewnej techniki. Użyto tutaj nie bez powodu słowa *technika*, ponieważ, pomimo że praca skupia się na pewnym algorytmie i jego implementacji w wysokopoziomowym języku programowania, to sam proces replikacji (reprodukcji) nie jest pojęciem nowym. Stosowany on był już przez wiele stuleci w przeszłości przez artystów (malarzy, rzeźbiarzy), którzy zajmowali się kopiowaniem innych dział malarskich lub rzeźbiarskich. Idea przyświecająca osobom, kopiującym działa malarskie, w dobie obecnej technologii, została przeniesiona na grunt techniki cyfrowej. Obecnie istnieje wiele narzędzi (programów lub bibliotek współpracujących z językami programowania) umożliwiających wykonywanie operacji na cyfrowych formatach obrazów. Narzędzia te stosują pewne wyspecjalizowane algorytmy, gdzie zadaniem każdego z nich jest dokonanie na obrazie pewnych zmian. Wraz z rozwojem techniki cyfrowego przetwarzania obrazów, naturalnym jest, że zdecydowano się również na cyfrową reprodukcję.

2.1.1 Idea cyfrowej replikacji obrazów

Rozważając cyfrowe podejście do replikacji obrazów należy rozumieć ten proces jako zastosowanie pewnego algorytmu, który mając do dyspozycji obraz oryginalny (reprodukowany) dąży do tego, aby stworzyć jego jak najwierniejszą kopię lub, głównie w przypadku algorytmów uczenia maszynowego (ang. *machine learning*), na podstawie pewnego zbioru obrazów utworzyć obraz mający cechy zbioru, na którym algorytm się uczył i jak najbardziej przypominający obraz stworzony przez człowieka (zdjęcie, rysunek, obraz namalowany przez malarza).

2.2 Cyfrowa reprezentacja obrazów

Obraz w postaci cyfrowej jest, na poziomie pamięci, reprezentowany, jak wszystkie dane cyfrowe, jako pewien ciąg bitów. Na poziomie języka programowania, w którym wykonywane są operacje na obrazie, obraz reprezentowany jest w pewnej skali kolorystycznej, np. RGBA, CMYK, HSV, CIELAB lub w skali szarości. Obraz przedstawiany jest jako zbiór pikseli, gdzie każdemu pikselowi jest przyporządkowany pewien kolor. W przypadku niniejszej pracy autor będzie używał dwóch skali, za pomocą których obrazy mogą być reprezentowane cyfrowo - RGBA oraz skali szarości. Poniżej zostaną opisane skale kolorystyczne oraz ich reprezentacja w języku `Go` (a dokładniej - w bibliotece `image` [zobacz [1]] wchodzącej w skład jego biblioteki standardowej), który został użyty do implementacji algorytmu opisywanego w pracy.

2.2.1 Skala RGBA

W modelu przestrzeni barw RGBA [zobacz [5]] obraz jest reprezentowany jako tablica pikseli o wymiarach $n \times m$. Każdy piksel (reprezentowany za pomocą pewnej struktury lub obiektu) posiada informację o kolorze w

tym przypadku zapisywaną za pomocą czterech parametrów zwanych kanałami. Pierwsze trzy kanały (RGB) odpowiadają odpowiednio kanałowi czerwonemu (ang. *red*), zielonemu (ang. *green*) oraz niebieskiemu (ang. *blue*). Ostatni kanał A jest nazywany kanałem *alfa* i określa on stopień przeźroczystości danego piksela.

W języku `Go` obraz zapisany za pomocą skali RGBA jest reprezentowany przez następującą strukturę:

```
type RGBA struct {
    Pix    []uint8
    Stride int
    Rect   Rectangle
}
```

Przechowywana jest w niej tablica pikseli obrazu (`Pix`) i dla każdego piksela zachowana jest kolejność kanałów, tj. R, G, B, A. Każdy kanał każdego piksela jest zakodowany za pomocą zmiennej typu `uint8` czyli za pomocą 8-bitowej liczby całkowitej bez znaku. Oznacza to, że każdy kanał może przyjmować wartości od 0 do 255 (256 różnych wartości). Parametr `Stride` oznacza odległość (w bajtach) pomiędzy dwoma sąsiadującymi wertykalnie pikselami, czyli długość jednego poziomego rzędu pikseli. `Rect` to struktura zawierająca wymiary obrazu.

2.2.2 Skala szarości

Skala szarości jest podzbiorem skali RGBA, tzn. rozważane są w niej takie kolory, w których pierwsze trzy kanały (RGB) maja dokładnie taką samą wartość. Zatem, upraszczając, można przyjąć, że każdy piksel może być kodowany jedynie za pomocą dwóch parametrów - pierwszego określającego jego jasność i drugiego - kanału *alfa*, który definiuje jego przeźroczystość.

W implementacji algorytmu, dla uproszczenia, do reprezentacji obrazów w skali szarości również będzie używana struktura `RGBA` z tym, że wartość dla każdego z trzech kanałów RGB będzie taka sama, tzn. będzie reprezentowana jedynie intensywność danego piksela.

2.3 Algorytmy metaheurystyczne

Tak jak opisano wyżej, na początku tego rozdziału, praca dotyczy cyfrowych sposobów replikacji obrazów i, co z tym związane, stosowanych do tego algorytmów i narzędzi. Autor pracy wybrał do przedstawienia tego procesu algorytm genetyczny, należący do rodziny algorytmów metaheurystycznych. W tym podrozdziale zostanie opisana idea algorytmów metaheurystycznych ze szczególnym uwzględnieniem algorytmów genetycznych. Przedstawiony zostanie również algorytm zaimplementowany przy okazji pisania tej pracy.

2.3.1 Idea i działanie

Zanim opisana zostanie idea algorytmów metaheurystycznych, należy wytłumaczyć pochodzenie ich nazwy. Pochodzi ona od słowa *heurystyka*, które wywodzi się z języka greckiego *heuriskō)* co tłumaczy się jako *znajduję*. Oznacza ono ”umiejętnośc wykrywania nowych faktów i związków między faktami”[zobacz [6]]. Przyjmując tę definicję można następnie zdefiniować słowo *metahuerystyka*, które określa ogólny algorytm służący do rozwiązywania pewnych problemów. Dokładniej - określenie to odnosi się do ”heurystyki wyższego poziomu”. Wynika to z faktu, że tego typu algorytmy nie rozwiązują bezpośrednio żadnego problemu, ale ich celem jest podanie sposobu na utworzenie odpowiedniego algorytmu.

Algorytmy metaheurystyczne używane są najczęściej do rozwiązywania problemów obliczeniowych z dużym naciskiem na problemy optymalizacyjne. Używa się ich w przypadkach, gdy rozwiązanie danego zagadnienia jest bardzo kosztowne. Dotyczy to głównie rozwiązywania problemów NP-trudnych (ang. *NP-hard problem*). W celu wyjaśnienia czym są problemy NP-trudne, zostaną przytoczone następujące definicje [zobacz [14]]:

Definicja 2.1 *Problemem **NP**, czyli niedeterministycznie wielomianowym (ang. nondeterministic polynomial) nazywamy taki problem obliczeniowy, którego rozwiązanie można zweryfikować w czasie wielomianowym. Mówimy, że dany problem należy do **klasy NP**, jeżeli spełnia definicję problemu NP-trudnego.*

Definicja 2.2 *Problemem NP-trudnym (ang. NP-hard) nazywamy taki problem obliczeniowy, którego rozwiązanie jest co najmniej tak trudne, jak rozwiązanie każdego problemu z klasy NP.*

Najbardziej znanymi problemami z rodziny problemów NP-trudnych są problem komiwojażera, problem plecakowy [zobacz [14]].

W świetle przedstawionych powyżej definicji można, intuicyjnie, rozumieć problemy NP-trudne jako te, których nie da się rozwiązać w czasie wielomianowym. Intuicja ta jest po części poprawna, ale należy ją doprecyzować. Problem obliczeniowy należy rozumieć jako problem z klasy NP, gdy wiadomo, że nie da się znaleźć jego rozwiązania działającego w czasie wielomianowym (zostało to udowodnione) lub takie rozwiązanie nie jest znane, tzn. nie udowodniono, że takie rozwiązanie nie istnieje.

W przypadku problemów obliczeniowych takich, jak opisano w powyższych akapitach, zaczęto poszukiwać algorytmów, które pozwolą uzyskanie dokładnego (lub bliskiego dokładnemu) rozwiązania w *sensownym* czasie. Słowo *sensowny* należy tutaj odnieść do czasu zbliżonego do czasu wielomianowego. Jedyną z dróg wybranych w badaniach nad tego typu problemami są algorytmy metaheurystyczne. Założeniem tych algorytmów jest znalezienie rozwiązania najbliższego rozwiązaniu optymalnemu.

Posiadając już zbiór definicji i pewną intuicję można przejść do zdefiniowania **algorytmu metaheurystycznego** [zobacz [10]].

Definicja 2.3 *Niech dany będzie problem P. Niech $L(P)$ będzie zbiorem możliwych zdań zapisanych w języku problemu P i niech będzie on traktowany jako zbiór wszystkich możliwych rozwiązań P. Rozwiązaniem problemu P jest dowolne $x \in S$, gdzie S jest podzbiorem zbioru $L(P)$. Algorytm metaheurystyczny jest opisem w jaki sposób, mając jeden z elementów zbioru $L(P)$, przejść do innego elementu należącego do tego zbioru, co, w konsekwencji, ma prowadzić do znalezienia zbioru S lub, w zależności od tego, jak problem został zdefiniowany, elementu $x \in S$.*

Zatem algorytm metaheurystyczny na podstawie jakiegoś znanego już rozwiązania problemu, dąży do wygenerowania innego rozwiązania (lub zbioru takich rozwiązań). Jednak cechą charakterystyczną tych algorytmów jest to, że nie gwarantują one znalezienia takiego rozwiązania ani nie jest znany dokładny czas działania takiego algorytmu. Ta druga cecha wynika głównie z faktu, iż w przypadku algorytmów metaheurystycznych korzysta się z generatorów pseudolosowych przy generowaniu kolejnych, potencjalnych rozwiązań, co nie gwarantuje, że algorytm na pewno od razu zacznie zbiegać do optymalnego rozwiązania. Biorąc pod uwagę obie te cechy, należy wyciągnąć wniosek, że tego typu algorytmy mają swoje zastosowanie tam, gdzie nie znane są inne algorytmy (działające w *rozsądnym* czasie) pozwalające znaleźć rozwiązanie danego problemu. Dobrym przykładem problemów, przy których rozwiązywaniu algorytmy metaheurystyczne znajdują swoje zastosowanie, są opisane problemy NP-trudne.

2.3.2 Rodzaje algorytmów metaheurystycznych

Można wyróżnić kilka rodzajów algorytmów metaheurystycznych. Niniejsza praca skupia się na algorytmach genetycznych, które zostaną opisane w kolejnym podrozdziale, ale dla umożliwienia zainteresowanemu czytelnikowi dalszych studiów literaturowych zostaną tutaj podane pozostałe rozdaje algorytmów metaheurystycznych z odesłaniem do odpowiedniej literatury. Wspomniane pozostałe typy takich algorytmów to **symulowane wyżarzanie**, **algorytmy mrówkowe**, **przeszukiwanie tabu**. Więcej na temat tych rodzajów algorytmów można przeczytać w [13], [11] i [8]. Oczywiście liczba publikacji dotyczących algorytmów metaheurystycznych jest o wiele większa i są to jedynie propozycje autora.

2.4 Algorytmy genetyczne

Wspomnianym wcześniej podzbiorem algorytmów metaheurystycznych są algorytmy genetyczne i to na nich skupia się niniejsza praca oraz powiązana z nią implementacja. W tym podrozdziale zostanie opisana pokrótce idea i sposób działania algorytmów metaheurystycznych.

2.4.1 Idea i działanie algorytmów genetycznych

Cechą algorytmów metaheurystycznych jest to, że wiele z nich jest inspirowane naturalnymi procesami. W przypadku symulowanego wyżarzania dążono do odwzorowania procesu wyżarzania stosowanego w metalurgii. Algorytmy mrówkowe są inspirowane działaniem kolonii mrówek, a dokładniej - sposobem poszukiwania

przez nie pożywienia. W przypadku algorytmów genetycznych, jak czytelnik może wyrobić sobie intuicję na podstawie ich nazwy, inspiracją do ich tworzenia było zjawisko naturalnej ewolucji biologicznej [zobacz [12]].

Działanie algorytmów genetycznych opiera się, jak wspomniano w poprzednim akapicie, na odwzorowaniu procesu biologicznej ewolucji organizmów żywych. Algorytmy genetyczne możemy zaliczyć do algorytmów stochastycznych. Przy ich opisie używa się w dużej mierze słownictwa zapożyczonego bezpośrednio z genetyki. Trafnym wstępem do wyjaśnienia działania tychże algorytmów będzie cytat z [9]: "[...] przenośnia leżąca u podstaw algorytmów genetycznych jest związana z ewolucją w naturze. W trakcie ewolucji każdy gatunek styka się z problemem lepszej adaptacji do skomplikowanego i zmiennego środowiska. "Wiedza", jaką gatunek zyskał, jest wbudowana w układ jego chromosomów". I tak - w przypadku algorytmów genetycznych autor będzie się posługiwał następującymi pojęciami zdefiniowanymi poniżej.

Definicja 2.4 *Osobnikiem lub chromosomem nazywa się jedno z rozwiązań stworzonych przez dany algorytm genetyczny podczas jednej iteracji.*

Definicja 2.5 *Populacją nazywa się zbiór rozwiązań (osobników) w danej iteracji algorytmu.*

Definicja 2.6 *DNA lub materiałem genetycznym nazywa się zbiór cech danego rozwiązania (osobnika).*

Definicja 2.7 *Operatorem genetycznym nazywamy pewna funkcję wprowadzającą określone zmiany w materiale gentycznym osobnika.*

Istotną rzeczą jest w tym miejscu sposób przechowywania informacji genetycznej każdego osobnika, czyli - wartości danego rozwiązania. W przypadku obrazów ciężko brać pod uwagę pojedynczą wartość liczbową. Rozważane tutaj będzie, jak wspomniano na początku tego rozdziału, reprezentowanie obrazu za pomocą skali RGBA lub jej pochodnej - skali szarości. Zatem obraz będzie pewną tablicą pikseli rozmiaru $n \times m$, gdzie każdemu pikselowi będzie przypisany kolor kodowany za pomocą czterech kanałów. Jest to reprezentacja dostosowana do tego konkretnego problemu. Najczęściej jednak rozważa się pewien ciąg bitów. W przypadku obrazów naturalnym jest, że na poziomie pamięci każdy obraz reprezentowany jest za pomocą pewnego ciągu bitowego. Taka reprezentacja jednak byłaby dosyć niewygodna na poziomie języka programowania. Ma jednak ona swoje zastosowanie np. dla problemu znajdowania optimum lokalnego dla danej funkcji na danym przedziale. Wtedy, każde rozwiązanie (przybliżana wartość optimum) może być reprezentowane za pomocą ciągu bitów, który w łatwy sposób może zostać zamieniony na swoją reprezentację dziesiętną. W tym wypadku reprezentacja bitowa jest wygodniejszym sposobem reprezentacji w kontekście działania na poszczególne osobniki za pomocą operatorów genetycznych.

Ogólny schemat działania algorytmu metaheurystycznego można przedstawić w następujący sposób:

1: Stwórz w wybrany sposób populacje początkową o liczebności t.
2: Zadziałaj na osobnikach wybranymi operatorami genetycznymi.
3: Oceń każdego osobnika funkcją oceny.
4: Stwórz populację do kolejnego pokolenia (iteracji) algorytmu.
5: Powtarzaj 2. - 4. aż do warunku zakończenia działania algorytmu.

Wybieranie populacji początkowej

Ten etap algorytmu genetycznego ma na celu wybranie populacji początkowej osobników do przeprowadzenia pierwszej iteracji algorytmu. Najczęściej mamy tutaj do czynienia z osobnikami z losowo wybranym materiałem genetycznym, czyli, w przypadku reprezentacji komputerowej, losowymi ciągami bitowymi generowanymi za pomocą wybranej funkcji pseudolosowej.

Operatory genetyczne

Jak zdefiniowano powyżej, operatory genetyczne są pewnymi funkcjami, które mają na celu zmodyfikowanie materiału genetycznego danego osobnika. Opisane tutaj zostaną dwa najczęściej wykorzystywane operatory - operator mutacji i krzyżowania. Zostały one użyte w implementacji algorytmu replikacji obrazów powiązanej z niniejszą pracą.

Operator mutacji polega na wprowadzeniu jednej lub więcej losowych zmian w materiale genetycznym danego osobnika. W przypadku reprezentacji bitowej osobników możemy rozumieć mutację jako zamianę (negację) jednego lub więcej bitów w danym ciągu.

Operator krzyżowania ma na celu wymianę materiału genetycznego pomiędzy dwoma osobnikami z populacji i działa bardzo podobnie jak krzyżowanie organizmów w procesie naturalnej ewolucji. Znowu, w przypadku reprezentacji bitowej, operację krzyżowania należy rozumieć jako wybranie (może być w sposób losowy) podciągów o ustalonej długości z każdego osobnika i zamiany ich miejscami w każdym z ciągów bitów.

Istotnym elementem algorytmu genetycznego jest sposób podejmowania decyzji o zastosowaniu danego operatora. W przypadku operatora mutacji najczęściej definiuje się pewnie prawdopodobieństwo zajścia mutacji w materiale genetycznym danego osobnika. W przypadku operatora krzyżowania ważny jest sposób wyboru osobników do krzyżowania - może być on zupełnie losowy lub mogą być wybrane do niego osobniki najmocniejsze. Odbywa się on na podobnych zasadach jak w przypadku wyboru osobników do każdego następnego pokolenia, co opisano poniżej.

Funkcja oceny

Tak, jak przedstawiono w definicji algorytmu metaheurystycznego, celem tegoż jest znalezienie rozwiązania (lub zbioru rozwiązań) z przestrzeni wszystkich możliwych rozwiązań danego problemu. Jako parametr wejściowy podawane jest jedno ze znanych rozwiązań i na jego podstawie algorytm próbuje przejść do innego rozwiązania, odpowiednio bliskiego rozwiązaniu poprawnemu. Bardzo istotną rolę w tym procesie odgrywa funkcja oceny. Pozwala ona określić dopasowanie każdego osobnika do oczekiwanego rozwiązania. Na jej podstawie wybierane są osobniki do kolejnego pokolenia używanego w następnej iteracji algorytmu. Funkcja oceny może też odgrywać też istotną rolę przy sprawdzaniu warunku zakończenia działania algorytmu, co zostanie opisane w dalszej części tego podrozdziału. Może ona być również używana przy wyborze osobników do operacji krzyżowania.

Jako przykład można podać funkcję oceny dla wspomnianego problemu znajdowania wartości dla optimum lokalnego funkcji. Wówczas funkcja oceny będzie sprawdzać różnicę pomiędzy rzeczywistą wartością dla optimum, a wartością danego rozwiązania (osobnika).

Wybór osobników do nowej populacji

Po etapie oceny osobników, kolejnym etapem jest wybór osobników do nowego pokolenia. Może się to odbywać na kilka różnych sposobów i jest to definiowane przez autora danego algorytmu. Istnieje kilka popularnych dróg doboru osobników:

1. **losowy wybór osobników** - w tym podejściu do następnego pokolenia osobniki wybierane są losowo (mogą się one powtarzać). Jest to sposób wyboru najbliższy temu naturalnemu.
2. **wybór osobników najmocniejszy** - do następnego pokolenia wybierane są tylko osobniki najmocniejsze.
3. **mieszany** - wybierana jest pewna pula osobników najmocniejszych, pewna pula osobników najsłabszych i pewna pula, którym funkcja oceny przypasowała wartość środkową.

Należy tutaj doprecyzować, że osobniki najmocniejsze rozumiane są jako te, które zostały ocenione jako najlepsze (funkcja oceny ma dla nich najwyższą wartość), a osobniki najsłabsze jako te, dla których funkcja oceny przyjęła wartość najniższą.

Warunek końca

Warunek zakończenia działania algorytmu jest definiowany w zależności od rozwiązywanego przezeń problemu. Może być on zdefiniowany jako pewna określona liczba N iteracji (pokoleń), po których algorytm powinien zakończyć swoje działanie. Może to być również pewne ograniczenie na dokładność rozwiązania, tj. algorytm musi znaleźć rozwiązanie, które będzie odległe od optymalnego nie mniej niż pewien ϵ.

Warunek końca gwarantuje zakończenie pracy algorytmu, ponieważ nie ma pewności (prawdopodobieństwo tego jest niewielkie), że algorytm znajdzie dokładne rozwiązanie problemu.

Uwaga dotycząca modyfikacji schematu działania algorytmu

Przedstawiony wyżej schemat algorytmu genetycznego nie jest w żaden sposób wiążący osoby tworzące takie algorytmy. Etapy 2, 3 i 4 mogą być dowolnie zamieniane lub powtarzane wielokrotnie. Jako przykład można podać algorytm, w którym najpierw stosowany jest operator mutacji na wybranych osobnikach, następnie wybierana jest nowa populacja, a przed rozpoczęciem kolejnej iteracji wykonywane jest jeszcze krzyżowanie pomiędzy osobnikami tworzącymi kolejne pokolenie.

2.5 Zastosowanie współbieżności algorytmów genetycznych

Algorytmy genetyczne są dobrym przykładem algorytmów, w których można w nieskomplikowany sposób zastosować model obliczeń równoległych. Wielowątkowość może zostać zastosowana na tych etapach algorytmów, gdzie na działa się na materiał genetyczny osobników operatorami genetycznymi. Zazwyczaj, jedynym momentem, w którym wymagana jest synchronizacja pomiędzy wątkami jest moment, gdy wybierani są przedstawiciele bieżącej populacji, którzy przejdą do kolejnej iteracji algorytmu. Powodem stosowania przetwarzania równoległego w algorytmach genetycznych jest przede wszystkim skrócenie czasu ich działania. Jest to bardzo cenna cecha tychże, ze względu na fakt, że nie da się dokładnie określić czasu, w jakim znajdą one rozwiązanie.

2.5.1 Obliczenia równoległe a algorytmy genetyczne

Dekompozycja problemu

W celu przystosowania algorytmu genetycznego do przetwarzania równoległego, należy na początku dokonać dekompozycji problemu [zobacz [7]]. W przypadku algorytmów genetycznych najlepiej sprawdzającym się podejściem będzie dokonanie dekompozycji względem danych. W przypadku algorytmu genetycznego za zbiór danych przyjmuje się zbiór wszystkich osobników w danym pokoleniu. Każdemu z dostępnych procesorów zostaje wtedy przydzielona pewna pula osobników, na których zostanie zastosowany operator genetyczny. Oprócz działania operatorami genetycznymi na poszczególne chromosomy, można łatwo zauważyć, że równolegle może być również dokonywana ocena osobników, ponieważ polega ona jedynie na porównaniu danego osobnika z wzorcowym rozwiązaniem

Synchronizacja

Dla algorytmów genetycznych wymagana jest synchronizacja pomiędzy działającymi wątkami w celu oceny osobników i wybrania nowej populacji. Po odebraniu przez jednostkę zarządzającą wątkami informacji o wykonaniu przez poszczególne procesory wszystkich zadań, tj. zadziałaniu na osobniki operatorami genetycznymi i ocenie dopasowania poszczególnych osobników, następuje wybór osobników do kolejnej iteracji algorytmu. Ten etap algorytmu genetycznego nie daje się przystosować do przetwarzania równoległego poprzez fakt, iż muszą być rozważone wszystkie osobniki z bieżącej populacji i utworzona musi zostać populacja kolejna. Ze względu na konieczność porównywania ze sobą wartości funkcji oceny dla każdego osobnika urównoleglenie tego kroku algorytmu jest niemożliwe.

2.5.2 Model współbieżności w języku Go

Jedną z cech języka `Go` jest bardzo dobrze rozwinięty model współbieżności. Jest to jedno z głównych zastosowań języka `Go`. Główną jednostką używaną w implementacjach wykorzystujących współbieżność w `Go` jest *gorutyna* (ang. *gorutine*) [zobacz [4]]. Jest to reprezentacja wątku, której główną cechą jest jej niewielki rozmiar w pamięci maszyny, co pozwala na tworzenie znacznej ich ilości z niewielkim nakładem pamięciowym. Jest Możliwe są w nim dwa podejścia do tego problemu. Jeden z zaimplementowanych modeli oparty jest na komunikację pomiędzy gorutynami za pomocą kanałów [zobacz [3]]. Odrębnym podejściem jest użycie do synchronizacji dostępu do pamięci klasycznych mutexów [zobacz [2]].

Algorytm replikacji

Na podstawie powyższych definicji i opisów dotyczących algorytmów metaheurystycznych można przedstawić algorytm genetyczny użyty w niniejszej pracy do replikacji obrazów. Algorytm zostanł tutaj przedstawiony jedynie w sposób opisowy. Z implementacją opatrzoną szczegółowymi komentarzami można zapoznać się w kodzie dołączonym do pracy.

3.1 Parametry algorytmu

3.1.1 Parametry wejściowe

1. **ścieżka do obrazu oryginalnego** - ścieżka do obrazu wzorcowego w formacie `JPG` lub `PNG`.
2. **ścieżka do zapisu wyników algorytmu** - ścieżka do lokalizacji, gdzie powinny zostać zapisane rozwiązania zwrócone przez algorytm.
3. **skala kolorów** - skala kolorów, w której reprezentowane będą poszczególne osobniki (RGBA lub skala szarości) i w której zapisany jest obraz wejściowy.
4. **liczba iteracji** - ograniczenie na liczbę iteracji (pokoleń) algorytmu.
5. **liczba osobników w pokoleniu** - liczba osobników w pojedynczej iteracji (pokoleniu).
6. **liczba najlepszych osobników** - liczba najlepszy osobników określa ile najlepszych rozwiązań (obrazów) zostanie zwrócone po zakończeniu działania algorytmu. Parametr ten jest również wykorzystywany w przypadku, gdy sposób wyboru osobników do kolejnej iteracji zostanie wybrany jako przekazywanie do następnej iteracji jedynie osobników najsilniejszych.
7. **operatory genetyczne** - parametr definiujący, które operatory genetyczne mają być stosowane przez algorytm. Dostępne operatory to operator mutacji oraz operator krzyżowania.
8. **prawdopodobieństwo mutacji** - parametr definiujący prawdopodobieństwo wystąpienia mutacji w materiale genetycznym każdego osobnika, który przyjmuje wartość rzeczywistą z zakresu $[0, 1]$.
9. **sposób wyboru osobników do kolejnej iteracji** - określa, w jaki sposób powinny być wybierane osobniki do kolejnego pokolenia algorytmu. Możliwe jest wybór losowy, wybieranie tylko najlepszych osobników lub mieszany jak opisano wyżej.
10. **sposób wyboru osobników do krzyżowania** - określa, w jaki sposób powinny być wybierane osobniki dla operatora krzyżowania. Możliwe jest wybór losowy, wybieranie tylko najlepszych osobników lub mieszany jak opisano wyżej.
11. **rodzaj wielokątów** - określa rodzaj wielokątów używanych do generowania kolejnych rozwiązań. Możliwe będzie przybliżanie kolejnych rozwiązań za pomocą dwóch rodzajów figur geometrycznych - trójkąta lub prostokąta.

3.1.2 Parametry wyjściowe

Algorytm zwraca (w postaci plików `PNG`) najlepsze osobniki z ostatniego pokolenia algorytmu. Liczba zwracanych osobników definiowana jest przez jeden z parametrów wejściowych algorytmu.

3.2 Pseudokod algorytmu

1: Stwórz populację początkową o liczebności t.
2: Użyj operatora mutacji na każdym z osobników z prawdopodobieństwem p_m.
3: Dokonaj oceny każdego osobnika i posortuj je rosnąco względem wartości funkcji oceny.
4: Wybierz do następnego pokolenia osobniki zgodnie ze sposobem wybranym przez użytkownika.
5: Zastosuj operator krzyżowania zgodnie ze sposobem wybranym przez użytkownika.
6: Powtarzaj kroki 2. - 5. aż do momentu osiągnięcia warunku końca algorytmu.

3.3 Opis etapów algorytmu

3.3.1 Wybór populacji początkowej

W przypadku opisywanego algorytmu replikacji stworzenie populacji początkowej polega na wygenerowaniu t czarnych obrazów (każdy z kanałów RGB przyjmuje wówczas wartość minimalną 0) o rozmiarze odpowiadającym rozmiarowi obrazu wzorcowego. Następnie, na każdym z wygenerowanych obrazów umieszczany jest losowy [1] kształt (trójkąt lub kwadrat) o losowych wymiarach i losowym położeniu na obrazie. Każdy z kształtów jest półprzeźroczysty co ułatwia późniejsze nakładanie kolejnych kolorów.

3.3.2 Operator mutacji

Operator mutacji ma na celu wprowadzenie losowej zmiany w materiale genetycznym każdego osobnika. Na początku działania funkcji odpowiedzialnej za dokonanie mutacji na wybranym osobniku wybierana jest rzeczywista liczba losowa z przedziału $[0, 1]$. Jeżeli jest mniejsza od prawdopodobieństwa wystąpienia mutacji, to w materiale genetycznym bieżącego osobnika wprowadzana jest mutacja. Polega ona na dodaniu losowego kształtu na obrazie. Pojawia się tutaj problem w jaki sposób wybrać kolor obszaru, w którym nakładają się na siebie dwa lub więcej kształtów o różnych kolorach. Przyjęto zasadę, że wówczas jako kolor danego piksela na opisanym obszarze przyjmuje się średnią z wartości wszystkich kanałów koloru piksela na obrazie i piksela w nakładanym w procesie mutacji kształcie.

3.3.3 Funkcja oceny

Funkcja oceny ma na celu przyznanie każdemu z osobników wartości liczbowej, która, odpowiednio wyliczona, pozwoli określić przystosowanie osobnika (jakość jego genotypu, czyli, w języku genetyki, zbioru genów) i później wyznaczyć, które osobniki mogą zostać uznane za najmocniejsze, które za średnio przystosowane, a które zaliczone zostaną do zbioru osobników słabo przystosowanych.

W prezentowanym w pracy algorytmie funkcja oceny wylicza różnicę pomiędzy wartościami na każdym z kanałów dla każdego piksela obrazu oryginalnego oraz ocenianego osobnika i sumuje wartości bezwzględne wyników. Osobnik, dla którego wartość funkcji oceny jest najmniejsza, będzie uznany za osobnik najmocniejszy, ponieważ najmniej *różni się* od obrazu oryginalnego. Analogicznie - osobnik, dla którego wartość funkcji oceny będzie największa zostanie sklasyfikowany jako osobnik najsłabszy, ponieważ najbardziej różni się od obrazu oryginalnego.

3.3.4 Wybór osobników do kolejnej iteracji algorytmu

W omawianej implementacji algorytmu służącego do replikacji obrazów możliwe są trzy rodzaje wyboru osobników do kolejnej populacji.

Pierwszym rodzajem wyboru jest wybór losowy. Można zauważyć, że jest to metoda najbardziej zbliżona do naturalnego procesu, w wyniku którego organizmy wymieniają między sobą swój materiał genetyczny.

[1] Słowo losowy w odniesieniu do kształtu zawsze będzie oznaczać trójkąt lub prostokąt o losowym kolorze, wymiarach i losowym położniu na obrazie

Pozwala on na wprowadzenie różnorodności w genach potomstwa i dopuszcza dziedziczenie cech zarówno powodujących lepsze przystosowanie organizmu względem pozostałych, jak i również konsekwencją stosowania tego sposobu wyboru jest szansa na przejście do kolejnej iteracji osobników gorzej przystosowanych [2].

Drugim rodzajem wyboru jest wybór do kolejnej iteracji jedynie osobników najmocniejszych. Może to zapewnić szybsze zbieganie algorytmu to poszukiwanego rozwiązania, jak również powoduje mniejszą różnorodność w materiale genetycznym potomstwa.

Trzecim rodzajem wyboru jest wybieranie zawsze pewnej puli osobników najmocniejszych, najsłabszych oraz tych, które oceniane są jako średnio przystosowane. Pozwala to zwiększyć różnorodność wśród potomstwa i eliminuje problem, który mógł się pojawić w przypadku wyboru losowego. Wówczas mogło bowiem dojść do sytuacji, gdy do następnej iteracji została wybrana pula osobników najsłabszych w danym pokoleniu.

3.3.5 Operator krzyżowania

Celem operatora krzyżowania jest wymiana informacji genetycznej pomiędzy dwoma osobnikami. Podobnie, jak w przypadku operatora mutacji, można w łatwy sposób wskazać analogię tej operacji do naturalnego procesu krzyżowania (rozmnażania się organizmów), podczas którego dochodzi do wymiany materiału genetycznego rodziców i stworzenia nowego genotypu potomka.

Operator krzyżowania w przypadku opisywanego algorytmu replikacji wybiera pewien prostokątny obszar na jednym z dwóch osobników przeznaczonych do krzyżowania. Następnie materiał genetyczny jednego osobnika jest wymieniany z drugim, tj. wybrany obszar z pierwszego obrazu jest kopiowany w dokładnie to samo miejsce na obrazie drugim i w ten sam sposób wybrany obszar z drugiego osobnika jest kopiowany w odpowiadające miejsce na osobniku pierwszym. Osobniki początkowe można określić mianem rodziców, a osobniki powstałe w wyniku krzyżowania określa się osobnikami potomnymi.

3.3.6 Współbieżność

W implementacji algorytmu replikacji zastosowano model przetwarzania równoległego w celu przyspieszenia obliczeń. Współbieżnie wykonywane są etapy algorytmu, podczas których działa się na osobniki operatorem mutacji i poddaje ocenie oraz proces krzyżowania osobników wybranych do nowej populacji. Synchronizacja pomiędzy wątkami dokonywana jest w momencie tworzenia nowej populacji. Proces odbywa się w ten sam sposób, jaki opisano w rozdziale niniejszej pracy poświęconym analizie problemu, a szczegóły implementacyjne, w raz z odpowiednim komentarze, dostępne są w załączonym do pracy kodzie.

[2] *Przystosowanie osobników* lub *dopasowanie* należy rozumieć jako to, jak bardzo dany osobnik jest odległy od rozwiązania wzorcowego

Instrukcja użytkownika

Analiza wyników

Analiza alternatywnych rozwiązań i narzędzi

Podsumowanie

W podsumowaniu zostaną zawarte wnioski wyciągnięte z pracy (oraz procesu jej tworzenia).

Bibliografia

[1] *Dokumentacja języka `Go` - biblioteka `image`.* [Dostęp: 16.11.2019r.].

[2] *Dokumentacja języka `Go` - biblioteka `sync`.* [Dostęp: 12.11.2019r.].

[3] *Dokumentacja języka `Go` - channels.* [Dostęp: 12.11.2019r.].

[4] *Dokumentacja języka `Go` - gorutines.* [Dostęp: 12.11.2019r.].

[5] Specyfikacja przestrzeni barw rgba. URL: https://www.w3.org/TR/PNG-DataRep.html. [Dostęp: 16.11.2019r.].

[6] Słownik języka polskiego. URL: https://sjp.pwn.pl/szukaj/heurystyka.html. [Dostęp: 12.11.2019].

[7] Z. J. Czech. *Wprowadzenie do obliczeń równoległych.* Wydawnictwo Naukowe PWN, 2013.

[8] M. L. F. Glover. *Tabu Search.*

[9] M. S. L. Davis. *Genetic Algorithms and Simulated Annealing: An Overview.*

[10] S. Luke. Essentials of metaheuristics, 2015. A Set of Undergraduate Lecture Notes.

[11] G. D. C. M. Dorigo. *The Ant Colony Optimization meta-heuristic.*

[12] Z. Michalewicz. *Algorytmy genetyczne + struktury danych = programy ewolucyjne.* Wydawnictwo Naukowo-Techniczne, 2003.

[13] E. H. L. A. P. J. M. van Laarhoven. *Simulated Annealing: Theory and Applications.*

[14] C. H. Papadimitriou. *Złożoność obliczeniowa.* Wydawnictwo Naukowo-Techniczne, 2002.

Zawartość płyty CD

W tym rozdziale należy krótko omówić zawartość dołączonej płyty CD.