Wydział Fizyki i Informatyki Stosowanej

Sprawozdanie

Algorytmy Genetyczne - Projekt

Aleksandra Poręba
nr. indeksu 290514

14 czerwca 2020

1 Wstęp

Celem projektu było napisanie programu znajdującego lokalizację małych płytek na dużej płycie, tak, aby powierzchnia małych płytek była jak największa. Do rozwiązania tego problemu został napisany program wykorzystujący algorytm genetyczny.

Jako dane wejściowe program przyjmuje listę wymiarów małych płyt. Powinna być ona zapisana w pliku *maleplyty.txt*, znajdującym się w tym samym katalogu, co plik wykonywalny programu. Wartości powinny zostać podane w milimetrach. Wymiary dużej płyty są z góry znane i wynoszą $2800x2070[mm]$.

Jako wyjście program zapisuje do pliku *output.txt* pole rozplanowanych płytek, oraz kolejno:

- szerokość i wysokość płytki,
- współrzędne lewego górnego rogu płytki - jeśli nie zostanie ona wycięta wartości te będą równe -1,
- flaga, czy płytka ma zostać obrócona przy cięciu.

Do napisania programu został wykorzystany język $C++$ wraz z biblioteką *GALib* oraz język *python + turtle* do rysowania rozkładu płytek.

2 Algorytm i kodowanie

W algorytmie, po przeprowadzeniu testów, ustawiono 600 pokoleń oraz 1000 osobników. Została włączona również opcja gwarantująca nie ujemność funkcji dostosowania - gdy kary są wyjątkowo wysokie, może przyjmować ona ujemne wartości.

W programie zostało wykorzystane kodowanie rzeczywistoliczbowe, z wartościami całkowitymi. Za lokalizacje jednej płytki odpowiada czwórka: *położenie lewego górnego wierzchołka (x, y), czy płytka jest obrócona, czy płytka ma zostać wycięta.*

Dane te mogą przyjmować następujące wartości:

- $0 \leqslant x \leqslant 2800$,
- $0 \leqslant y \leqslant 2070$,
- $r \in \{0, 1\}$,
- $e \in \{0, 1\}$.

Dzięki takiemu kodowaniu wartości, zgodnie z założeniami, będą całkowite oraz w zadanych obszarach.

3 Funkcja dostosowania

Funkcja dostosowania algorytmu została przedstawiona w następujący sposób:

$$f(S_i) = \sum_j P_{i,j} - a \cdot g(S_i) - b \cdot h(S_i) \tag{1}$$

S_i - położenia płytek dla danego genomu,

$P_{i,j}$ - pole j-tej płytki,

a - parametr skalujący funkcję kary g,

$g(S_i)$ - funkcja kary wychodzenia poza obszar dużej płyty,

b - parametr skalujący funkcję kary h,

$h(S_i)$ - funkcja kary przecinania się z innymi płytkami.

Funkcja kary sprawdzająca wychodzenie poza wyznaczony obszar uruchamiana jest dla każdej płytki. Zwraca ona kwadrat pola, które "wystaje" poza dużą płytę.

Kara za przecięcie z innymi płytkami obliczana jest dla każdej pary płytek. Liczone jest pole wspólnego obszaru obu kształtów, funkcja kary zwraca kwadrat tej wartości.

W trakcie ewolucji parametry kary aktualizowane są trzykrotnie. Na samym początku wartości $a, b = 10$, po upływie 180 pokoleń (0.3 ustawionej liczby pokoleń) wartość ta jest zwiększana do 100, a następnie, po kolejnych 120 pokoleniach do 1000. Zwiększając te współczynniki stopniowo algorytm eliminuje słabe, nieprawidłowe rozwiązania jednocześnie nie odrzucając rozwiązań bardzo dobrych, ale z małym błędem i pozwala im wyewoluować w dobrym kierunku.

W ostatniej aktualizacji, po 0.7 liczby wszystkich pokoleń wyłączane jest karanie w funkcji dostosowania, tak, aby ostatecznie uzyskać jedynie poprawne wyniki. Wtedy, w przypadku przecięcia, funkcja dostosowania jest zerowana.

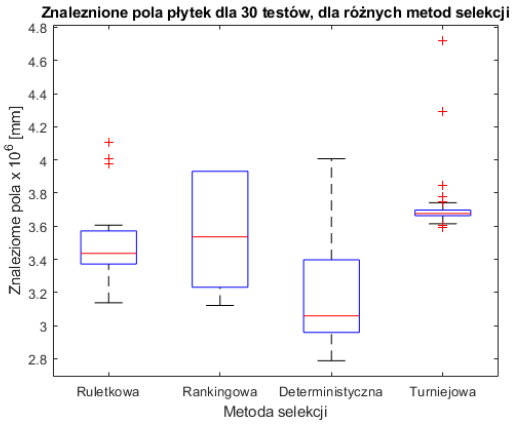
4 Testowanie parametrów

W algorytmie została włączona sukcesja elitarna, dzięki której algorytm uzyskiwał lepsze wyniki. Podobnie jak w przypadku zadania z laboratorium 7 zmiany przy operatorze krzyżowania nie wpływały znacząco na otrzymywane wyniki, więc pozostawiono domyślny operator, a jego prawdopodobieństwo ustawione zostało na wartość 0.3.

Zostały przetestowane następujące operatory selekcji:

- selekcja ruletkowa,
- selekcja deterministyczna,
- selekcja rankingowa,
- selekcja turniejowa.

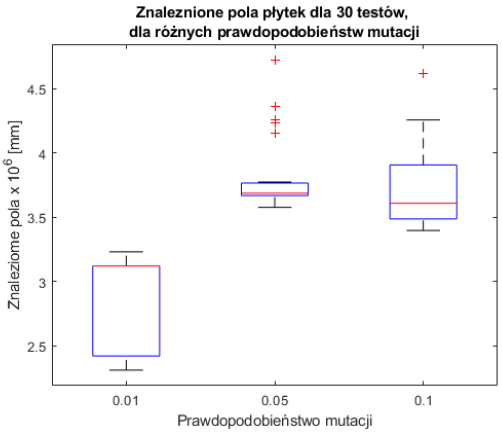
Prawdopodobieństwo mutacji zostało ustawione na 0.05. Uzyskane wyniki dla 30 testów zostały przedstawione poniżej.



Rysunek 1: Wyniki 30 testów różnych parametrów, przedstawione w formie wykresu *pudełka z wąsami*. Najlepsze wyniki osiągała selekcja turniejowa, najgorsze deterministyczna.

Jako najlepsza została wybrana selekcja turniejowa - uzyskujemy dla niej najwyższe wartości pola małych płytek. Średnie wyniki które osiąga skoncentrowane w okolicach $3.7 * 10^6$. Algorytm genetyczny jest uruchamiany wiele razy w czasie pojedynczego uruchomienia programu, więc możemy się spodziewać, że uda się mu odnaleźć wartości odstające, które dla tej selekcji są najwyższe.

Dla wybranej selekcji przetestowano różne wartości prawdopodobieństwa mutacji.



Rysunek 2: Wyniki 30 testów różnych parametrów, przedstawione w formie wykresu *pudełka z wąsami*. Zadowalające wyniki osiągały wartości 0.05 oraz 0.1.

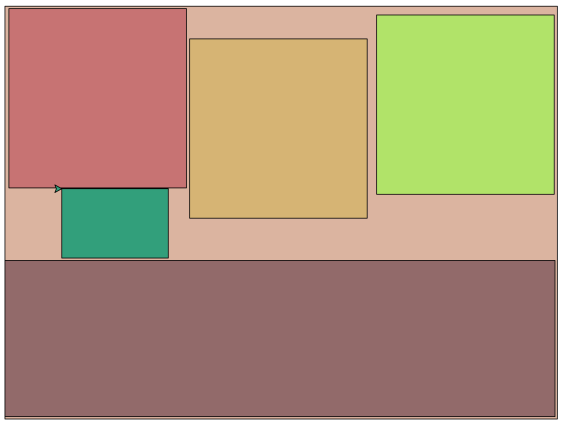
Finalnie została wybrana wartość $p_m = 0.05$.

5 Uruchomienie programu

Aby uruchomić skompilowaną wersję programu należy uruchomić plik *AG_proj.exe* znajdujący się w katalogu *exe*. Plik wejściowy o nazwie *maleplyty.txt* powinien znajdować się w tym samym katalogu.

Jako wynik działania tworzony jest plik *output.txt*, w którym znajduje się suma pól małych płyt, które zostaną wycięte, ich położenia - zgodnie z zadanym formatem. Ten plik może zostać użyty do uruchomienia skryptu $draw_results.py$, który przedstawi wycięcie płytek w sposób graficzny. Aby użyć programu konieczne jest włączenie opcji przekazywania sesji $X11$.

Skrypt do rysowania kolejno zaprezentuje ustawienie płytek, tak jakby były wycinane. Aby wyjść z programu po zakończeniu rysowania należy kliknąć w okno. Wynik zostanie zapisany do pliku *board.eps*. Przykładowy wynik został przedstawiony poniżej.



Rysunek 3: Graficzna reprezentacja przykładowego rozkładu płytek, dla pola równego $4.81 * 10^6$

Aby skompilować program można użyć narzędzia *cmake* i wykorzystać przygotowany plik $CMakeLists.txt$. Znajduje się on w katalogu AG. Wymagane jest posiadanie biblioteki $GALib$.

Została przetestowana poprawność działania funkcji obliczającej karę dla różnych położeń płytek. Przygotowany test znajduje się w katalogu *test*. Aby go uruchomić również można skorzystać z przygotowanego pliku $CMakeLists.txt$, tym razem z katalogu *test*. Testy sprawdzają, czy jest poprawnie obliczane pole przecięcia płytek, oraz czy płytka nie wystaje poza wyznaczony obszar.