Algorytmy optymalizacji
Projekt

Optymalizacja rozmieszczenia bloków 2D

Dokumentacja projektu

Autorzy:	Marcin Kordas, Dariusz Palt
Numer indeksu:	246812, 246808
Prowadzący:	Prof. zw. dr hab. inż. Czesław Smutnicki
Grupa:	Piątek TN 13:15, Środa TN 13:15
Kod grupy:	K05-00c, K05-00e

Wrocław, 2022

Wstęp

Zgodnie z empirycznym prawem Moore'a liczba tranzystorów dostępna w układach scalonych podwaja się co dwa lata. Pomimo znacznego wzrostu mocy obliczeniowej układów scalonych zarówno ich rozmiar, pobór mocy jak i koszt produkcji z biegiem czasu nie ulegał równie drastycznemu zwiększeniu, a wręcz przeciwnie – stopniowo się zmniejszał. Głównym i oczywistym powodem takiego stanu rzeczy jest postęp technologiczny: zdolność masowej produkcji coraz mniejszych i bardziej wydajnych tranzystorów. Istotny wpływ ma jednak także zastosowanie nowszych i lepszych metod projektowania układów scalonych. Ich efektem jest ustalenie przestrzennego rozplanowania poszczególnych elementów układu, czyli w tym przypadku tranzystorów, przy jednoczesnej optymalizacji najważniejszych cech układu takich jak pole powierzchni, pobór mocy czy łączna długość ścieżek przewodzących. Jak istotna jest optymalizacja związana z projektowaniem tego typu układów wskazuje przykład firmy Intel, w której zwiększenie pola powierzchni układu tylko o 1% przełożyło się na zwiększenie kosztów produkcji aż o 63 miliony dolarów. Właśnie tego typu rozważania stały się motywacją do realizacji niniejszego projektu.

1. Opis oraz matematyczny model problemu

Problem projektowania i rozmieszczania bloków tworzących układy scalone będący motywem przewodnim projektu sformułować możemy jako zagadnienie optymalizacji kombinatorycznej. Jego punktem początkowym jest zbiór poszczególnych elementów układu scalonego – w tym przypadku tranzystorów, a konkretniej ich grup. Rozwiązanie natomiast stanowi fizyczne rozmieszczenie elementów i jednocześnie w zależności od przyjętej funkcji celu maksymalizuje lub minimalizuje jej wartość. Uproszczeniem w tym wypadku jest prezentacja opisywanego rozwiązania, która przedstawia kolejne tranzystory za pomocą prostokątów o określonych wymiarach. Rozwiązanie każdemu z nich przypisuje zatem konkretne położenie na płytce drukowanej. Naturalnie, elementy te nie mogą na siebie nachodzić. Problem ten w literaturze określany jest jako Block Packing Problem i jest NP-trudny.

Jak już zostało wspomniane – zadanie optymalizacji polega na znalezieniu takiego rozmieszenia prostokątnych bloków symbolizujących grupy tranzystorów, które w zależności od przyjętej funkcji celu maksymalizuje ją lub minimalizuje. Można zatem zapisać, że:

Posiadamy zbiór $E = \{e_1, e_2, \dots, e_N\}$ będący zbiorem N zadanych prostokątnych elementów, które przeznaczone są do rozmieszczenia. Każdy z tych elementów posiada swoją wysokość w_i oraz szerokość s_i. Poszukiwane jest natomiast rozmieszenie L stanowiące N zestawów liczb postaci (x_i, y_i, o_i), gdzie para (x_i, y_i) oznacza współrzędne lewego górnego rogu elementu i, a o_i jego orientację – pionową lub poziomą.

$A(L)$ jest przyjętą funkcją celu i oznacza pole powierzchni najmniejszego prostokąta, w którym mieści się rozmieszczenie L. L^* to z kolei rozwiązanie optymalne spośród poprawnych rozwiązań L, a więc takie, które minimalizuje funkcję celu $A(L)$.

Rozwiązanie stanowi prostokątny plan uzyskiwany poprzez stopniowe wypełnianie prostokąta reprezentującego płytkę drukowaną mniejszymi prostokątami symbolizującymi kolejne tranzystory przy pomocy wspierającego algorytmu meta heurystycznego.

2. Algorytm generowania rozwiązania

W celu poszukiwania optymalnego rozwiązania zaimplementowane zostały dwa algorytmy. Pierwszy z nich jest to algorytm genetyczny, który odpowiedzialny jest za generowanie oraz oceniać kolejno powstałych rozwiązań. Algorytmy genetyczne należą do grupy algorytmów ewolucyjnych, a te należą do algorytmów metaheurystycznych. Algorytmy genetyczne oparte są na mechanizmach dziedziczności i doboru naturalnego. Łączą w sobie zasadę przeżycia najlepiej przystosowanych jednostek i zasadę losowej wymiany informacji. Wartym dodania faktem, że pomimo opierania się na zjawisku losowości (generowanie losowej populacji) wynik jego nie jest przypadkowy – wykorzystuje doświadczenie z przeszłości do ciągłego zawężania zbioru poszukiwań. Zapis ogólny algorytmu genetycznego prezentuje się następująco:

1) Losowanie populacji początkowej

2) Selekcja, inaczej ocena populacji

3) Wybór najlepszych osobników do procesu reprodukcji

4) Modyfikacja genotypów – krzyżowanie oraz mutacja

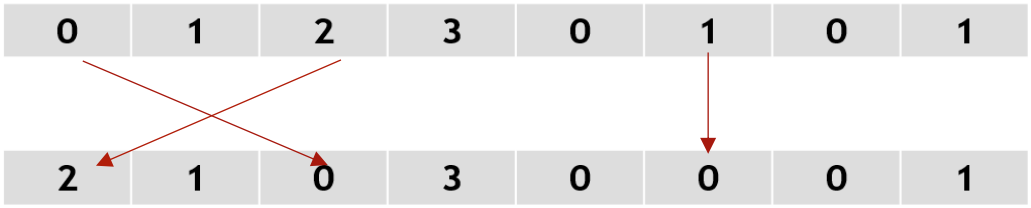
5) Powielenie najlepszych osobników oraz usuwanie najsłabszych osobników (aby utrzymać liczebność populacji) po czym powrót do kroku 2 lub wybór najlepszego osobnika będącego rozwiązaniem problemu.

Na potrzeby opisanego powyżej algorytmu zaimplementowane zostały metoda wspierająca generowanie wstępnej populacji, strategia zastępowania słabszych rodziców, metody selekcji, a także mutacji oraz krzyżowania:

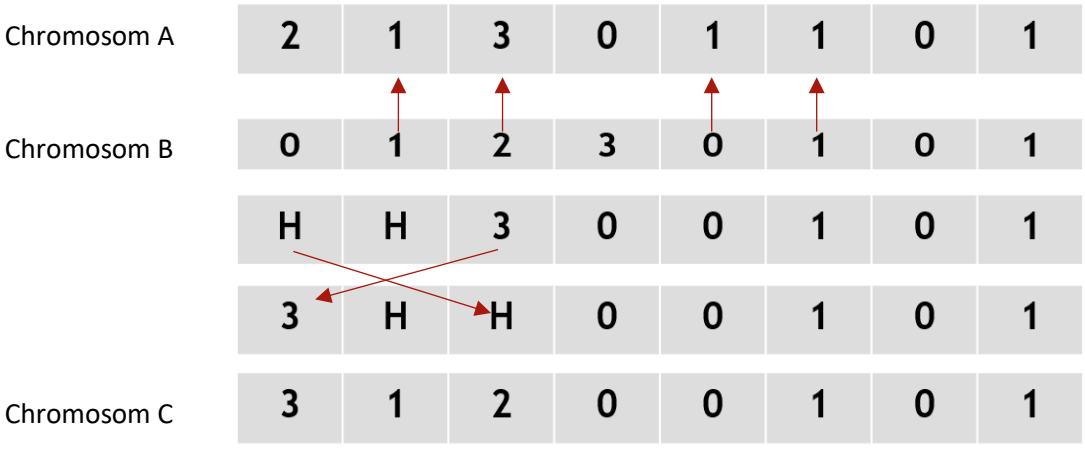
a) Generowanie populacji początkowej dla chromosomów o indeksach od 0 do n – 6 następuje w sposób losowy. Sześć ostatnich chromosomów tworzonych jest natomiast w oparciu o kryteria odpowiednio: pola bloku, szerokości bloku, wysokości bloku, obwodu, najszerszego bloku oraz przekątnej bloku. Bloki dominujące w wybranych kategoriach ustawiane są jako pierwsze w kolejności w chromosomie.

b) Selekcja, a więc metoda wybierania osobników przeznaczonych do reprodukcji została zaimplementowana w dwóch wersjach: rankingowej i ruletki. Pierwsza z nich eliminuje ¼ najgorszych osobników. Druga natomiast przypisuje prawdopodobieństwa wylosowania każdemu z osobników bazując przy tym na wartości funkcji jakości jaką one wyznaczają.

c) Mutacja, a więc drobne, losowe modyfikacje chromosomów, zaimplementowana została w różny sposób w zależności od typu genu, którego dotyczy. Chromosom, który podzielony jest na dwie części, w pierwszej z nich zawiera kolejność ułożenia bloków, a w drugiej orientacje każdego z nich. W pierwszej części mutacja zamienia miejscami ze sobą dwa losowo wybrane indeksy bloków. W części drugiej natomiast zamienia losowo wybraną jedną orientację bloku na przeciwną.



d) Krzyżowanie polega na wymianie genów pomiędzy dwoma chromosomami tworząc w ten sposób trzeci będący kombinacją swoich rodziców. Zaimplementowany został algorytm odpowiednio przesuwający geny przy ich wymianie między chromosomami tak, aby zapobiec ich powtórzeniu się w nowym osobniku. Początkowo algorytm wybiera dwa losowe przedziały. Pierwszy z nich zawiera się w pierwszej połowie chromosomu, drugi natomiast – w drugiej. W przypadku drugiej części chromosomu krzyżowanie nie jest skomplikowane, ponieważ jest to część binarna, w której nie ma ograniczeń co do unikalności zbioru genów. Nowy chromosom (chromosom C) powstaje, poprzez uzupełnienie go genami spoza wylosowanego przedziału z chromosomu A oraz z wylosowanego przedziału odpowiadającymi genami z chromosomu B.



W przypadku krzyżowania pierwszej połowy chromosomu, problem jest bardziej skomplikowany ze względu na wymóg unikalności zbioru genów. W przedstawionej powyżej sytuacji niemożliwe jest zastosowanie tego samego algorytmu co dla drugiej połowy genotypu ze względu na uzyskanie genów: [2,1,2,0]. W celu wygenerowania pierwszej części nowego chromosomu C należało: skopiować geny z chromosomu A; wybrać przedział, który chcemy zastąpić genami z chromosomu B; wystąpienia wybranego przedziału genów chromosomu B zastąpić w chromosomie C literami „H”; przenieść litery „H” w ten sposób, aby pokrywały się indeksami z wybranym

przedziałem; zastąpić przygotowany przedział w chromosomie C, genami z przedziału chromosomu B .

Kluczowym dla powodzenia algorytmu genetycznego jest dobór odpowiedniego sposobu kodowania, który pozwoli na skuteczne przeszukanie przestrzeni rozwiązań. Jak już zostało wspomniane chromosom składa się z dwóch części. Przykładowo może prezentować się następująco:

2	**1**	**3**	**0**	**1**	**1**	**0**	**1**

Informację jaką przenosi dekodujemy jako:

a) Kolejność ułożenia bloków to: 2, 1, 3, 0
b) Orientacje kolejnych bloków to: pionowa, pionowa, pozioma, pionowa

Drugim algorytmem jaki został zaimplementowany w ramach niniejszego projektu jest metaheurystyczny algorytm odpowiedzialny za umieszczanie bloków symbolizujących tranzystory na powierzchni większego prostokąta symbolizującego płytkę drukowaną. Pseudokod wspomnianego algorytmu zaprezentowano poniżej:

1) Zbiór prostokątnych bloków $E_N = \{e_1, e_2, \dots, e_N\}$, $e_i = (id_i, w_i, h_i, o_i)$
2) Panel ograniczający $P = \{w_P, h_P\}$
3) Rozmieszczenie bloków L = []
4) Chromosom $chromosome = [g_{e_1}, g_{e_2}, \dots, g_{e_N}, g_{o_1}, g_{o_2}, \dots, g_{o_N}]$
5) Szerokości $widths = []$
6) usedWidht = 0, usedHeight=0, currentHeight = 0, currentWidht =0
7) Dla i = 1 do i = N:
    a. currentGen = chromosome[i]
    b. wantedOrientation = chromosome[i+N]
    c. currentBlock = E[currentGen]
    d. jeżeli currentBlock.o != wantedOrientation:
        i. block.flip()
    e. jeżeli currentBlock.width + currentWidth $\leq w_P$
        i. currentBlock.setPosition(currentWidth, currentHeight)
        ii. usedWidth = usedWidth + currentBlock.width
        iii. jeżeli currentBlock.height > usedHeight:
            1. usedHeight = currentBlock.height
        iv. currentWidth = curremtWidth + currentBlock.width
    f. w.p.p:
        i. currentHeight = currentHeight + usedHeight
        ii. usedHeight = 0
        iii. widths.append(currentWidth)
        iv. currentWidth = 0
        v. currentBlock.setPosition(currentWidth, currentHeight)
        vi. usedWidth = usedWidth + currentBlock.height
        vii. jeżeli currentBlock.height > usedHeight:
            1. usedHeight = currentBlock.height

viii. currentWidth = currentWidth + currentBlock.width

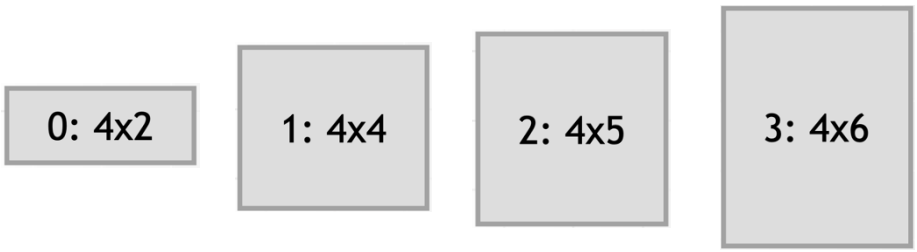
g. L.append(currentBlock)

8) Widths.append(currentWidth)

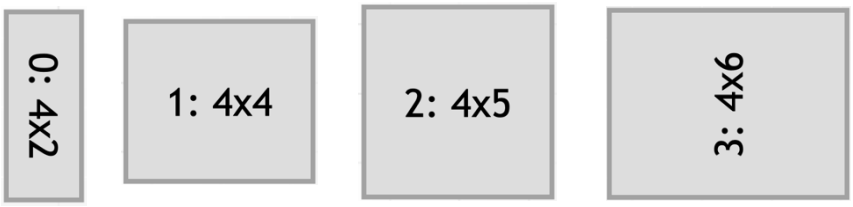
9) Funkcja przystosowania $A(L) = max(widths) * (currentHeight + usedHeight)$

Przykładowe działanie w oparciu o chromosom zaproponowany przy okazji omawiania algorytmu genetycznego prezentuje się następująco:

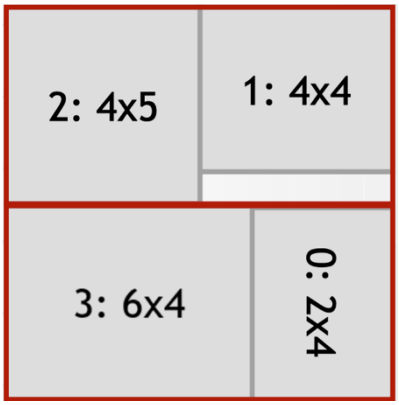
1) Posiadamy bloki o przykładowych wymiarach:



2) Zmieniamy ich położenie uwzględniając zakodowaną orientację



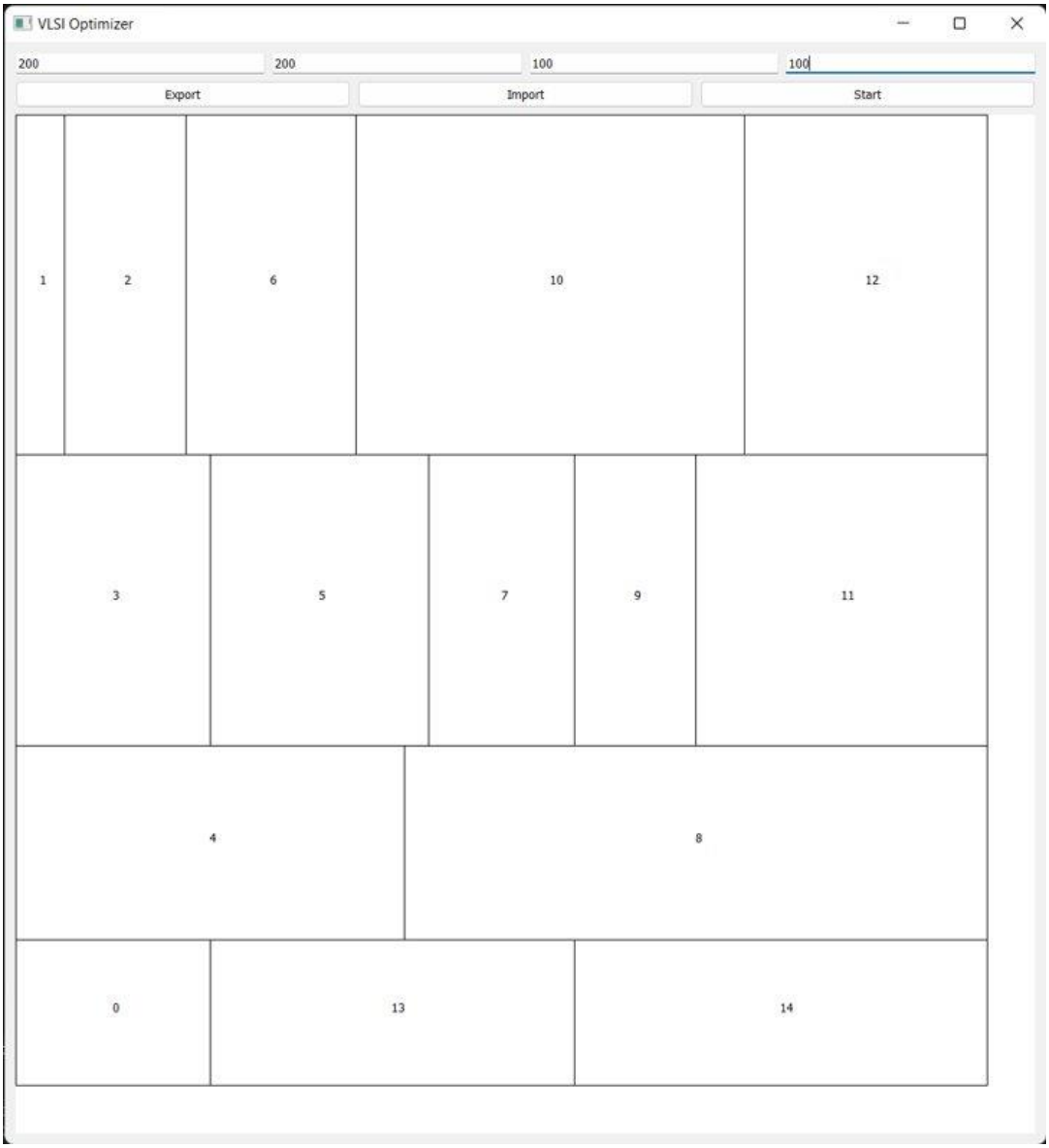
3) Układamy od lewej strony mając na uwadze zadaną wielkość płytki (w tym przypadku 8x9) oraz zakodowaną kolejność ułożenia bloków: 2, 1, 3, 0



Tak powstałe rozmieszczenie bloków staje się rozwiązaniem zadanego problemu, naturalnie, jeśli akurat to ten chromosom osiąga najmniejszą wartość funkcji celu spośród całej populacji.

3. Implementacja

Program rozwiązujący problem zadany w ramach niniejszego projektu zaimplementowany został z wykorzystaniem języka Python. Jego kod źródłowy załączony został wraz z niniejszą dokumentacją projektu. Użytkownik programu ma możliwość skorzystania z graficznego interfejsu w celu wygenerowania rozwiązania oraz zwizualizowania go. Przed rozpoczęciem optymalizacji ma możliwość określenia populacji epok oraz rozmiarów płytki, na której chciałby umieścić tranzystory. Dodatkowo, w celu ułatwienia wprowadzania danych ma on także możliwość wczytania zestawu bloków (ich indeksów wraz z wymiarami) z pliku w formacie .xlsx. Rozwiązanie prezentowane jest w formie graficznej wizualizacji rozmieszczenia bloków i możliwe jest jego zapisanie również w pliku o formacie .xlsx. Zrzut ekranu z przykładowego funkcjonowania programu zaprezentowano poniżej:



4. Wnioski

Zrealizowany projekt pozwolił oswoić się z założeniami algorytmu genetycznego. Zwrócił on uwagę na jego zalety takie jak zdolność do przeglądu dużej liczby dostępnych konfiguracji, ale także przypomniał o jego cesze charakterystycznej, a więc uzależnieniu od wszechobecnej losowości. Algorytm przy omawianym problemie wykazywał się dużą wrażliwością na jakość populacji początkowych i przejawiał tendencję do utykania w optimach lokalnych. Polem do poprawy niewątpliwie w tym zakresie są możliwość lepszego dostrojenia parametrów algorytmu lub modyfikacja i potencjalne usprawnienie metod krzyżowania a także mutacji chromosomów.

5. Bibliografia

1) W. Bożejko, J. Pempera: „Optymalizacja dyskretna w informatyce, automatyce i robotyce” (2012). Oficyna Wydawnicza PWr.
2) P. R. Fernando: "Genetic Algorithm Based Design and Optimization of VLSI ASICs and Reconfigurable Hardware" (2008). USF Tampa Graduate Theses and Dissertations.