



WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

**PRACA MAGISTERSKA**

*Optymalizacja podziału dyskretnego modelu  
symulacyjnego na potrzeby symulacji równoległej*

*Optimization of the discrete simulation model division for parallel  
simulation*

Autorzy: *Jakub Ziarko*

Kierunek studiów: *Informatyka*

Typ studiów: *Stacjonarne*

Opiekun pracy: *dr hab. inż. Wojciech Turek*

Kraków, 2021

# Spis treści

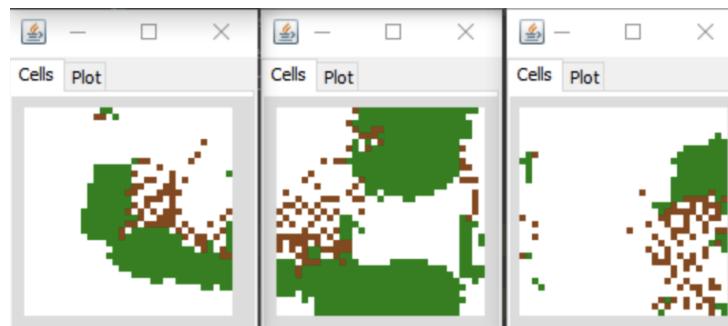
<b>1 Wstęp</b>	<b>3</b>
1.1 Motywacja . . . . .	3
1.2 Cel pracy . . . . .	4
1.3 Struktura pracy . . . . .	4
<b>2 Istniejące metody partycjonowania grafów</b>	<b>5</b>
2.1 Metody spektralne . . . . .	5
2.2 Metody rekursywne . . . . .	6
2.3 Metody geometryczne . . . . .	7
2.4 Metody wielopoziomowe . . . . .	8
2.5 Porównanie istniejących metod wielopoziomowych . . . . .	9
<b>3 Opis algorytmu</b>	<b>11</b>
3.1 Podział siatki na $m \cdot k$ obszarów . . . . .	12
3.1.1 Konwersja pliku graficznego na graf i redukcja obszarów niepodzielnych . . . . .	12
3.1.2 Zmniejszanie i partycjonowanie grafu za pomocą algorytmu LAM .	15
3.1.3 Szczegóły modyfikacji algorytmu LAM . . . . .	20
3.1.4 Przywracanie grafu do początkowego rozmiaru wraz z ulepszeniem podziału . . . . .	23
3.1.5 Używanie Helpful Sets w celu zmniejszenia długości granic . . . . .	26
3.1.6 Szczegóły budowania zbioru helpful . . . . .	33
3.1.7 Szukanie zbioru $k$ -helpful gdzie $k \geq limit$ . . . . .	35
3.1.8 Szczegóły budowania zbioru balancing . . . . .	36
3.1.9 Balansowanie rozmiarów obszarów . . . . .	40
3.1.10 Usuwanie obszarów rozproszonych . . . . .	42
3.2 Podział siatki na $m$ obszarów . . . . .	44
<b>4 Wyniki</b>	<b>46</b>
4.1 Wyniki dla podziału na $m \cdot k$ obszarów . . . . .	46
4.1.1 Wyniki dla podziału siatek bez obszarów niepodzielnych i wyłącznych z obliczeń . . . . .	47
4.1.2 Wyniki dla podziału siatek z obszarami niepodzielnymi i wyłącznymi z obliczeń . . . . .	48
4.2 Wyniki dla podziału na $m$ obszarów . . . . .	59
<b>5 Podsumowanie i kierunek rozwoju</b>	<b>62</b>

# 1. Wstęp

## 1.1. Motywacja

Partycjonowanie siatek jest bardzo ważnym podproblemem dla wielu dziedzin. Ma między innymi zastosowanie dla sieci komputerowych, a także dla obliczeń równoległych, gdzie zależy nam, aby problem podzielić na równe części pomiędzy węzły. Jednym z takich problemów jest symulacja 2D, dla której definiujemy zasady jej przebiegu oraz siatkę, na której ma się odbywać. Przykładowo celem takiej symulacji może być zbadanie ruchu pieszego na mapie galerii handlowej, a następnie wyodrębnienie miejsc zatłoczonych, które wymagają poprawek w projekcie. Innym przykładem symulacji jest symulacja królików i kapusty przedstawiona na rysunku 1.

Z racji na wielkość mapy taka symulacja często jest bardzo wymagająca obliczeniowo, dla tego dążymy do jej zrównoleglenia. Jedną z metod zrównoleglenia tego typu obliczeń jest podział siatki wejściowej pomiędzy węzły obliczeniowe, tak by każdy węzeł zajmował się swoim obszarem. Węzły mogą otrzymywać obszary równe pod względem wielkości pola, jeśli są to węzły homogeniczne, lub proporcjonalne do ich możliwości obliczeniowych, jeśli są heterogeniczne. Problemem, który towarzyszy zrównolegleniom jest narzut komunikacji. Siatka pomimo podziału na części funkcjonuje jako całość, więc węzły muszą wiedzieć co dzieje się na granicach obszarów sąsiednich, aby symulacja odbywała się prawidłowo. W celu zminimalizowania narzutu komunikacyjnego należy więc zminimalizować długość granic pomiędzy obszarami.



Rysunek 1: Fragment symulacji królików i kapusty. Króliki zjadają kapustę zwiększając swoją populację. Ze względu na zwiększoną populację królików i spadającą ilość pożywienia populacja królików spada, z czasem pozwalając populacji kapusty po raz kolejny się odrodzić. Każde okno obsługiwane jest przez osobny rdzeń procesora.

Patrząc na symulację jak ta przedstawioną na rysunku 1, łatwo zaproponować bardzo dobry podział, ponieważ siatka jest prostokątna. Podział otrzymamy rysując pionowe oraz poziome separatory na siatce. Jeśli przyjmiemy właściwy sposób ich rysowania otrzymamy idealny podział zarówno pod względem wielkości pól, jak i długości granic pomiędzy obszarami. Problem przestaje być trywialny, jeśli siatka nie jest prostokątem lub chcemy, aby wyodrębnione obszary siatki nie zostały podzielone między partycje, to znaczy należały zawsze do jednej partycji. Często również z różnych powodów obliczenia nigdy nie odbędą się na pewnych obszarach siatki. Mogą to być na przykład ściany na mapie galerii handlowej lub pomieszczenia, do których nie wchodzą klienci. Chcemy aby tego typu

obszary zostały w taki sposób ujęte w podziale siatki, aby uniknąć sytuacji, kiedy jeden węzeł otrzyma do symulowania część siatki z obszarem, na którym symulacja nie będzie się odbywać. Nie będzie on wtedy wykorzystywany do symulacji.

## 1.2. Cel pracy

Celem niniejszej pracy jest skonstruowanie algorytmu tworzącego podział siatki na partycje. Podział będzie się odbywał dla węzłów homogenicznych, a więc takich gdzie każdy rdzeń ma identyczną moc obliczeniową. Partycje muszą więc być równe pod względem wielkości pola. Dodatkową cechą takiego partycjonowania jest zminimalizowanie długości granic między partycjami (ang. edge-cut), tak by narzut komunikacyjny był jak najniższy. Początkowa siatka powinna pozwalać na zdefiniowanie obszarów niepodzielnych oraz obszarów wyłączonych z obliczeń. Obszar niepodzielny nie może znajdować się w więcej niż jednej partycji, natomiast obszar wyłączony z obliczeń powinien być uwzględniany podczas partycjonowania, tak by wszystkie węzły obliczeniowe obsługiwały równe pod względem pola obszary siatki, na których ma szansę odbyć się symulacja.

W celu wykonania podziału siatka zostanie zamieniona na odpowiadający jej graf, który następnie przy wykorzystaniu proponowanych przez literaturę oraz zmodyfikowanych przez mnie algorytmów grafowych zostanie poddany partycjonowaniu. Partycjonowanie będzie się odbywało dwukrotnie. Jeśli bowiem mamy  $m$  węzłów obliczeniowych, każdy po  $k$  rdzeni, to siatka najpierw zostanie podzielona na  $m \cdot k$  możliwie równych pod względem pola obszarów, a następnie  $m \cdot k$  obszarów zostanie podzielone na  $m$  obszarów, każdy po  $k$  podobszarów. Tak działający algorytm został opisany w niniejszej pracy.

## 1.3. Struktura pracy

W rozdziale drugim zebrano przegląd istniejących rozwiązań w zakresie partycjonowania grafów. Następnie spośród przeanalizowanych rozwiązań wybrano to, które najlepiej prognozowało dla problemu niniejszej pracy. W rozdziale trzecim zaproponowano i opisano rozwiązanie problemu partycjonowania siatki dla  $m$  węzłów homogenicznych. Rozdział czwarty prezentuje otrzymane rezultaty. Rozdział piąty zawiera podsumowanie oraz dalsze kierunki rozwoju.

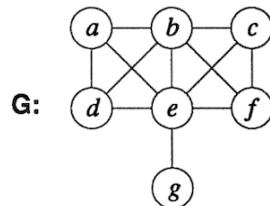
## 2. Istniejące metody partycjonowania grafów

Rozbudowany podział metod został zaproponowany przez autorów artykułu - [16] oraz [24]. Zgodnie z ich analizą metody dzielimy na:

- spektralne,
- rekursywne,
- geometryczne,
- wielopoziomowe.

### 2.1. Metody spektralne

Partycjonowanie spektralne daje dobre rezultaty i jest metodą w miarę często używaną [26, 27, 18]. Podzbiór wierzchołków  $S \subset V$  grafu  $G$  nazywamy separatorem jeśli dwa wierzchołki w tym samym komponencie grafu  $G$  są w dwóch różnych komponentach w grafie  $G \setminus S$ .



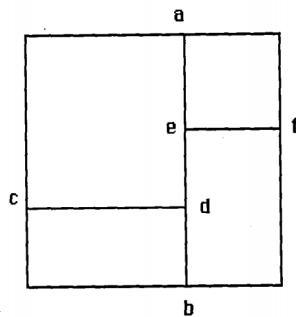
Rysunek 2: Separator wierzchołków  $d$  oraz  $c$  to  $\{b, e\}$ . Źródło: [22].

Metoda [26], będąca algorytmem spektralnym pokazuje algebraiczne podejście do obliczania separatorów wierzchołków. Artykuł [27] opisuje algorytm spectral nested dissection (SND). Powyższe dwa algorytmy za pomocą znajomości spektralnych właściwości macierzy Laplaciana obliczają separatory wierzchołków w grafie, które tworzą partycjonowanie grafu. Artykuł [18] mówi o nowatorskim rozwinięciu metody spektralnej pod kątem umożliwienia podziału obliczeń na cztery bądź osiem części na każdym etapie rekursywnej dekompozycji. Są to jednak metody kosztowne z racji na obliczanie wektorowa własnego odpowiadającego drugiej najmniejszej wartości własnej (Fielder wektor). Istnieją udane próby skrócenia czasu wykonania tych metod, które polegają na liczeniu Fielder wektora poprzez algorytm wielopoziomowy - MSB [1]. Jednak nawet one wciąż charakteryzują się wysoką złożonością.

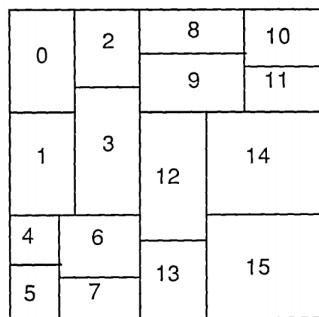
## 2.2. Metody rekursywne

Często są to metody prostsze w implementacji, jednak nie sprawdzają się tak dobrze w kontekście bardziej skomplikowanych problemów, głównie ze względu na zachłanną naturę. Metoda [2] zakłada, że dzielimy siatkę na liczbę obszarów, która jest równa potęce liczby dwa. Metoda ta potrafi także dzielić siatkę wedle możliwości obliczeniowych poszczególnych rdzeni procesora. Czasami metody rekursywne są implementowane jako faza metod bisekcji spektralnej [26]. Przykłady działania partycjonowania rekursywnego przedstawione są na rysunkach 3 oraz 4.

Specyfika tych metod, polegająca na dzieleniu grafu na coraz mniejsze części sprawia, że nie jesteśmy w stanie uwzględnić części niepodzielnych lub rozwiązań tego problemu byłoby skomplikowane.



Rysunek 3: Partycjonowanie rekursywne na głębokości wynoszącej 2. Linia partycjonowania a-b, która została stworzona przez partycjonowanie poziomu 1 jest podzielona na 3 segmenty przez dwie linie partycjonowania poziomu drugiego: c-d, e-f. Źródło: [2]



Rysunek 4: Metoda rekursywna - binarna dekompozycja dla 16 procesorów. Najpierw tworzone jest wertykalne cięcie, które gwarantuje, że prawy i lewy obszar zawiera połowę pracy do wykonania (lub takie, które jest jak najbliżej takiego podziału). Jeśli dostępne są 4 procesory to każdy z dwóch segmentów partycjonowany jest horyzontalną linią, która spełnia te same założenia jak ta dla pierwszego wertykalnego cięcia. Procedura jest kontynuowana wykorzystując na zmianę wertykalne i horyzontalne cięcie, aż do otrzymania podziału na oczekiwanaą liczbę obszarów. Źródło: [2]

### 2.3. Metody geometryczne

Inną klasą metod są metody geometryczne [20, 29, 21, 22, 25]. Używają danych geometrycznych o grafie w celu znalezienia dobrego partycjonowania. Ich cechą charakterystyczną jest relatywnie krótki czas wykonania przy gorszych rezultatach podziału niż w przypadku metod spektralnych. Najlepsze wyniki prezentują metody [21, 22]. W artykule [21] zaproponowano klasę grafów zwaną  $k$ -overlap graphs. Dla grafów  $k$ -overlap osadzonych w  $d$  wymiarach [33] udowadniają istnieje separatora wielkości  $O(k^{1/d}D^{d-1/d})$ . Oznacza to, że dla grafu o  $N$  wierzchołkach istnieje podzbiór wierzchołków o wcześniej wymienionej wielkości, który po usunięciu rozłącza graf na dwie podobnej wielkości części. Wyniki w tym artykule unifikują kilka wcześniejszych wyników dla obliczania separatorów. Ponadto autorzy proponują rozwiązanie, które oblicza separatory w czasie liniowym. W artykule [22] opisano efektywny sposób partycjonowania siatek niestrukturalnych, który występuje w metodach elementów skończonych i różnic skończonych. Podejście to wykorzystuje strukturę geometryczną danej siatki i znajduje dobre partycjonowanie w czasie  $O(n)$ . Można je aplikować do siatek w dwóch i trzech wymiarach. Ma zastosowanie w wydajnych algorytmach sekwencyjnych i równoległych do rozwiązywania rozbudowanych problemów w obliczeniach naukowych. Charakterystyczną cechą metod geometrycznych, wynikającą z ich losowej natury, jest konieczność wielokrotnego użycie algorytmu (od 5 do 50 razy), aby uzyskać wynik porównywalny z metodami spektralnymi. Wielokrotnie wywołanie wydłuża czas uzyskania rezultatu, jednak jest on wciąż niższy od metod spektralnych.

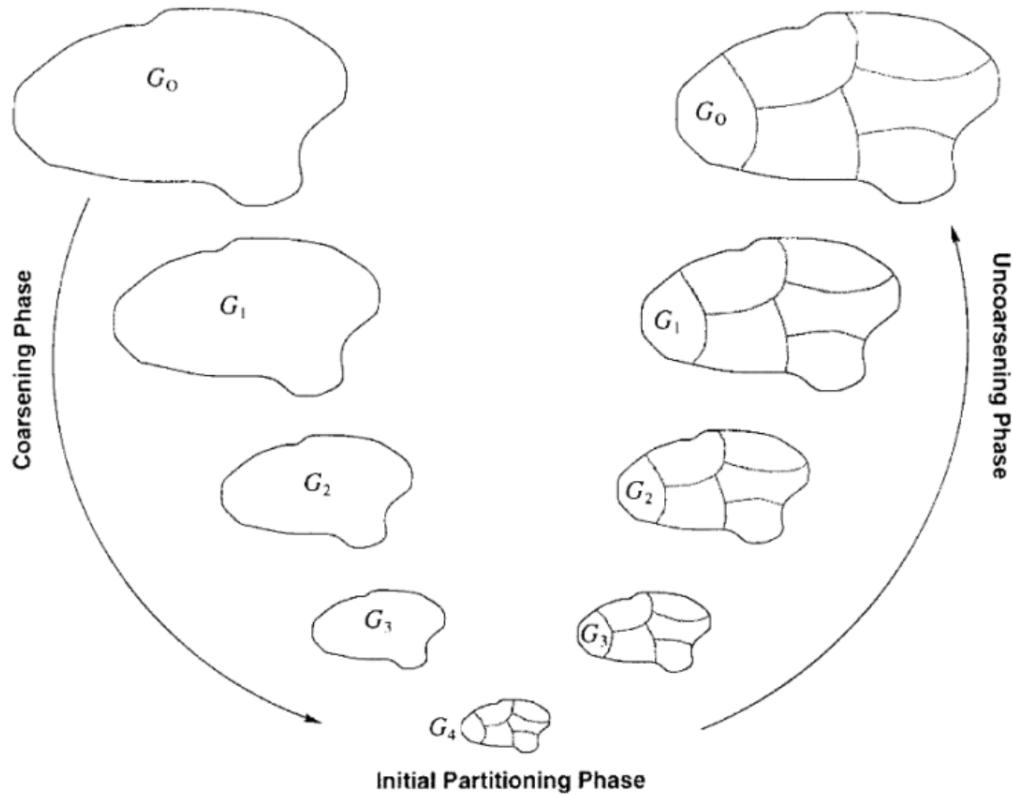
Metody geometryczne są możliwe do zastosowania tylko w przypadku, gdy dostępne są współrzędne wszystkich wierzchołków w grafie. Dla wielu dziedzin problemów (programowanie liniowe, VLSI) nie otrzymujemy współrzędnych wraz z grafem. Istnieją algorytmy, które są w stanie obliczyć współrzędne dla wierzchołków grafu [4] wykorzystując metody spektralne, są one jednak bardzo kosztowne i dominują czas potrzebny na partycjonowanie grafu. Implementacje tego typu metod nie znalazły się wśród algorytmów state-of-the-art dla problemu partycjonowania grafów, więc nie były brane pod uwagę w niniejszej pracy.



Rysunek 5: Rekursywne partycjonowanie mapy USA - pierwszy obrazek od góry - za pomocą algorytmu z artykułu [22]. Dwa następne obrazki prezentują rezultat. Dane geometryczne używane są do obliczenia separatorów.

## 2.4. Metody wielopoziomowe

Istnieje wiele implementacji metod wielopoziomowych: [16, 32, 3, 5, 10, 9, 11, 8, 19]. Cechą charakterystyczną tego podejścia jest redukcja wielkości grafu, poprzez łączenie wierzchołków i krawędzi, następnie dzielenie zmniejszonego grafu na partycje. Ostatnią fazą jest przywrócenie początkowego grafu z zachowaniem podziału. Często graf jest zmniejszany aż liczba wierzchołków nie osiągnie liczby partycji, którą chcemy otrzymać [24]. Fazie przywracania grafu do początkowej wielkości towarzyszy algorytm, którego celem jest ulepszanie podziału [6, 7]. Algorytm ten, bazując na zmniejszonym grafie, niesie ze sobą niższy koszt obliczeniowy. Jego działanie polega na zmniejszaniu długości granic pomiędzy partycjami z jednoczesnym zachowaniem wielkości obszarów. Na tym etapie może także zostać dodana faza balansowania, która stopniowo zmniejsza różnice w wielkości pól poszczególnych obszarów. Metody te zostały stworzone z myślą o zmniejszeniu czasu patrycjonowania kosztem jego jakości. Obecnie dają jednak bardzo dobre rezultaty również co do jakości podziału. Późniejsze prace w dziedzinie tych algorytmów pokazały, że dają one lepsze rezultaty niż metody spektralne [16]. Biblioteki jak Party [24], Metis [16], Jostle [32], Chaco [11], dające state-of-the-art wyniki w kwestii jakości partycjonowania najczęściej bazują na schemacie wielopoziomowym [11].

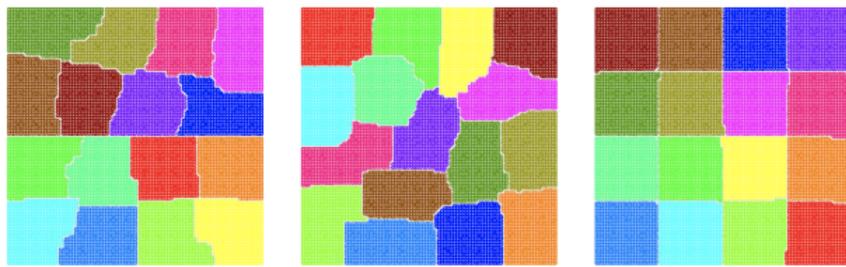


Rysunek 6: Wielopoziomowe partycjonowanie grafu, przedstawiające fazę zmniejszania grafu, następnie przypisanie partycji na zmniejszonym grafie, na końcu przywrócenie grafu do początkowej wielkości. Źródło: [14].

## 2.5. Porównanie istniejących metod wielopoziomowych

Wszystkie wyżej wymienione metody nie biorą pod uwagę problemu obszarów niepodzielnych oraz obszarów wyłączonych z obliczeń. W związku z tym celem niniejszej pracy stało się znalezienie metody dającej możliwie najlepsze rezultaty w zakresie partycjonowania grafów oraz dostosowanie jej do wyżej wymienionych rozszerzeń problemu partycjonowania.

Metody wielopoziomowe gwarantują najlepsze wyniki partycjonowania. Przykładów tych jest bardzo wiele [16, 32, 3, 5, 10, 9, 11, 8, 19], jednak skupiłem się głównie na tych, które dawały wyniki state-of-the-art. Były to biblioteki: Party [24], Metis [16], Jostle [32], Chaco [11].



Rysunek 7: Partycjonowanie siatki 100x100 na 16 obszarów. Od lewej - pmetis [16] używa edge-cut wynoszący 688, następnie Jostle [32] z wynikiem 695 oraz Party [24] z wynikiem 615. Źródło: [24].

Do zmniejszenia grafu stosowane są różne warianty matching algorithm (algorytm do budowania skojarzeń w grafie). Porównanie większości z nich można znaleźć w [13]. Przykładowo biblioteka Jones oraz Bui [3] używa random weighted matching, natomiast Party [24] stosuje LAM matching [28]. Ze względu na wysoką złożoność obliczeniową wszystkie metody używają heurystyk. Wszystkie z tych metod zmniejszają graf, jednak tylko Jostle i Party zmniejsza graf, aż do otrzymania liczby wierzchołków równej liczbie partycji, na które chcemy podzielić wejściowy graf. Dzięki temu metoda partycjonowania, działająca na najmniejszym możliwym grafie, jest dużo prostsza niż w przypadku pozostałych metod. Kiedy graf jest zmniejszony, wierzchołki są przyporządkowywane do partycji, a informacja na temat partycji jest przenoszona do wierzchołków na wyższych poziomach, zgodnie z partycją ich reprezentantów na najniższym poziomie. Ten proces prowadzi do partycjonowania początkowego grafu. Faza zmniejszania grafu jest ponadto stosunkowo łatwa do zrównoleglenia [15]. Fazą, która nie podlega zrównolegleniu jest faza ulepszania istniejącego podziału. Najczęściej bazuje ona na metodzie Fiduccia-Mattheysesa [7], która jest zoptymalizowaną pod kątem czasu działania heurystyką Kerninghan-Lin (KL) [17]. Artykuł [17] podejmuje problem partycjonowania wierzchołków grafu, którego krawędzie mają przyporządkowane wagi - koszt. Jego celem jest ustalenie podziału na obszary o wybranej wielkości wraz ze zminimalizowaniem sumy kosztów na wszystkich granicach. Artykuł [7] przedstawia heurystykę do poprawiania partycjonowania sieci. Algorytm ten przesuwa pojedyncze komórki pomiędzy blokami wraz z utrzymaniem oczekiwanej rozmiaru bloków. W przeciwnieństwie do Metis i Jostle faza ulepszania partycjonowania używana przez bibliotekę Party, bazuje na metodzie Helpful-Sets (HS). Heurystyka Helpful-Set wywodzi

się z obserwacji teoretycznych wykorzystywanych do znajdowania górnych granic szerokości bisekcji grafów regularnych [12, 23]. Szerokość bisekcji to minimalna liczba krawędzi, która musi zostać usunięta w celu podzielenia grafu na dwie równej wielkości części lub różniące się wielkością o maksymalnie jeden wierzchołek. Party daje bardzo dobre wyniki stosując tę metodę [30], uzyskując często mniejszą długość granicy pomiędzy obszarami niż Metis czy Jostle. Jednocześnie jest tylko trochę bardziej kosztowna obliczeniowo. Heurystyka Helpful-Sets jest metodą bazującą na wyszukiwaniu lokalnym. Zaczynając od początkowej bisekcji  $\pi$  dąży do zminimalizowania długości granicy za pomocą lokalnie wprowadzanych zmian. Najważniejsza różnica względem metody KL polega na tym, że KL przemieszcza tylko pojedyncze wierzchołki, natomiast HS zbiory wierzchołków. Zarówno Party, Metis, jak i Jostle pozwalały na małe różnice wielkości obszarów, co przekłada się na mniejsze długości granic, szczególnie na głębszych poziomach, kiedy przemieszczane są wierzchołki o dużych wagach. Ciężko wtedy o otrzymanie idealnie równych obszarów. Biblioteka Party [24] okazała się dawać najlepsze rezultaty w porównaniu do innych bibliotek dających wyniki state-of-the-art. Szczególną własnością Party była znacznie niższa długość granic w porównaniu do pozostałych bibliotek (Rysunek 7), co było bardzo ważne dla mojej pracy. Dlatego to właśnie metodę z biblioteki Party zdecydowałem się wybrać jako podstawę rozwiązania, prezentowanego w niniejszej pracy.

### 3. Opis algorytmu

W tym rozdziale zostanie szczegółowo opisany algorytm podziału siatki, który został zaimplementowany na potrzeby niniejszej pracy. W ramach tego algorytmu, zakładając  $m$  node'ów, każdy zawierający  $k$  rdzeni, wyróżniam dwa następujące etapy:

1. Podział siatki na  $m \cdot k$  obszarów.
2. Podział siatki podzielonej na  $m \cdot k$  obszarów na  $m$  obszarów.

W ramach etapu podział siatki na  $m \cdot k$  obszarów wyróżniam następujące podetapy:

1. mapowanie wejściowego pliku graficznego przedstawiającego początkową siatkę na graf,
2. zmniejszanie grafu algorytmem LAM [28] do liczby wierzchołków równej liczbie partycji, na które chcemy podzielić wejściową siatkę,
3. przypisanie numerów partycji do wierzchołków w zmniejszonym grafie,
4. stopniowe przywracanie grafu z jednoczesnym wyrównaniem krawędzi [30] oraz balansowaniem pól obszarów (mniejsze obszary powiększają się kosztem większych),
5. usunięcie obszarów rozproszonych.

W ramach etapu podziału siatki na  $m$  obszarów wyróżniam następujące podetapy:

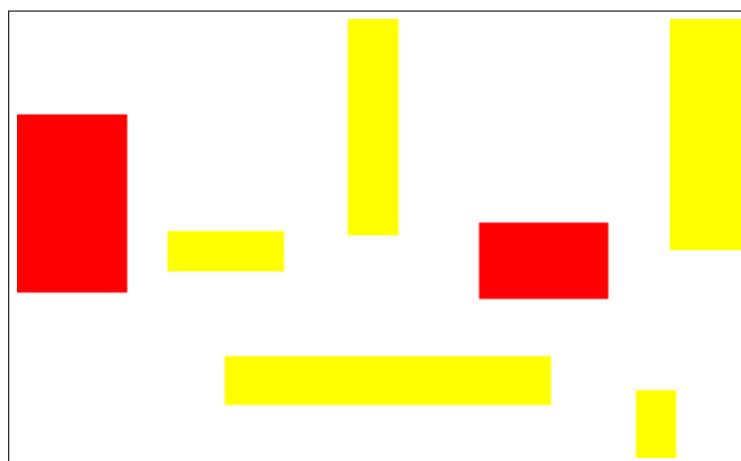
1. zmniejszanie grafu algorytmem LAM [28] do liczby wierzchołków równej liczbie partycji, na które chcemy podzielić wejściową siatkę,
2. przypisanie numerów partycji do wierzchołków w zmniejszonym grafie,
3. udoskonalenie podziału.

### 3.1. Podział siatki na $m \cdot k$ obszarów

W ramach etapu podziału siatki na  $m \cdot k$  obszarów wyróżniam konwersję pliku graficznego na graf, redukcję rozmiaru grafu, podział zredukowanego grafu oraz przywrócenie grafu do początkowego rozmiaru wraz z wprowadzeniem optymalizacji podziału (wyrównanie powierzchni pól i zmniejszenie długości granic pomiędzy partycjami).

#### 3.1.1. Konwersja pliku graficznego na graf i redukcja obszarów niepodzielnych

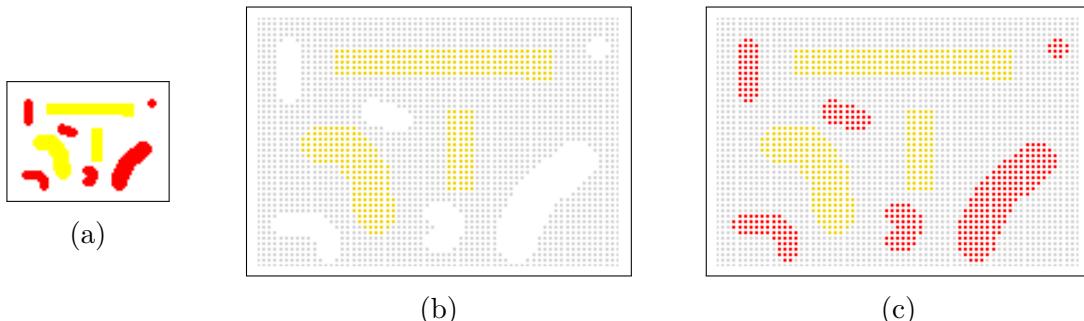
Pierwszą częścią algorytmu jest tworzenie grafu na podstawie siatki, która dostarczana jest jako dana wejściowa. Siatka jest w formie pliku graficznego. Przykład takiego pliku przedstawiony jest na rysunku 8. Jeden piksel na siatce reprezentuje jeden wierzchołek w grafie, na którym dokonywane będzie partycjonowanie. Kolor piksela decyduje o tym, jakiego typu wierzchołkiem będzie dany piksel oraz do jakiego typu obszaru należy. Wierzchołki, które są tego samego koloru oraz sąsiadują ze sobą, są przyporządkowywane do tego samego obszaru. Żółte piksele interpretowane są jako wierzchołki obszarów niepodzielnych, natomiast czerwone piksele jako wierzchołki obszarów wyłączonych z obliczeń, białe piksele to wierzchołki zwyczajne. Obszary wyłączone z obliczeń mogą być używane do zaznaczenia fragmentów siatki, na których symulacja nie będzie miała miejsca. Mogą być to na przykład ściany budynku lub pomieszczenia, które są wyłączone z symulacji. Obszary niepodzielne to obszary, które z jakiś powodów nie powinny być podzielone na partycje, na przykład wejścia do budynków. Przynależność wierzchołków do konkretnych grup wpływa na ich parametry w grafie, takie jak waga. Może również spowodować, że nie pojawią się na grafie. Ta część algorytmu, choć stosunkowo mało skomplikowana, jest kluczową dla niniejszej pracy. W związku z tym, że autorzy artykułu [24] nie zawarli informacji w jakiej formie dostarczany był wejściowy graf, ta część algorytmu została zrealizowana w całości przeze mnie. Efekt zamiany siatki na graf widoczny jest rysunku 9.



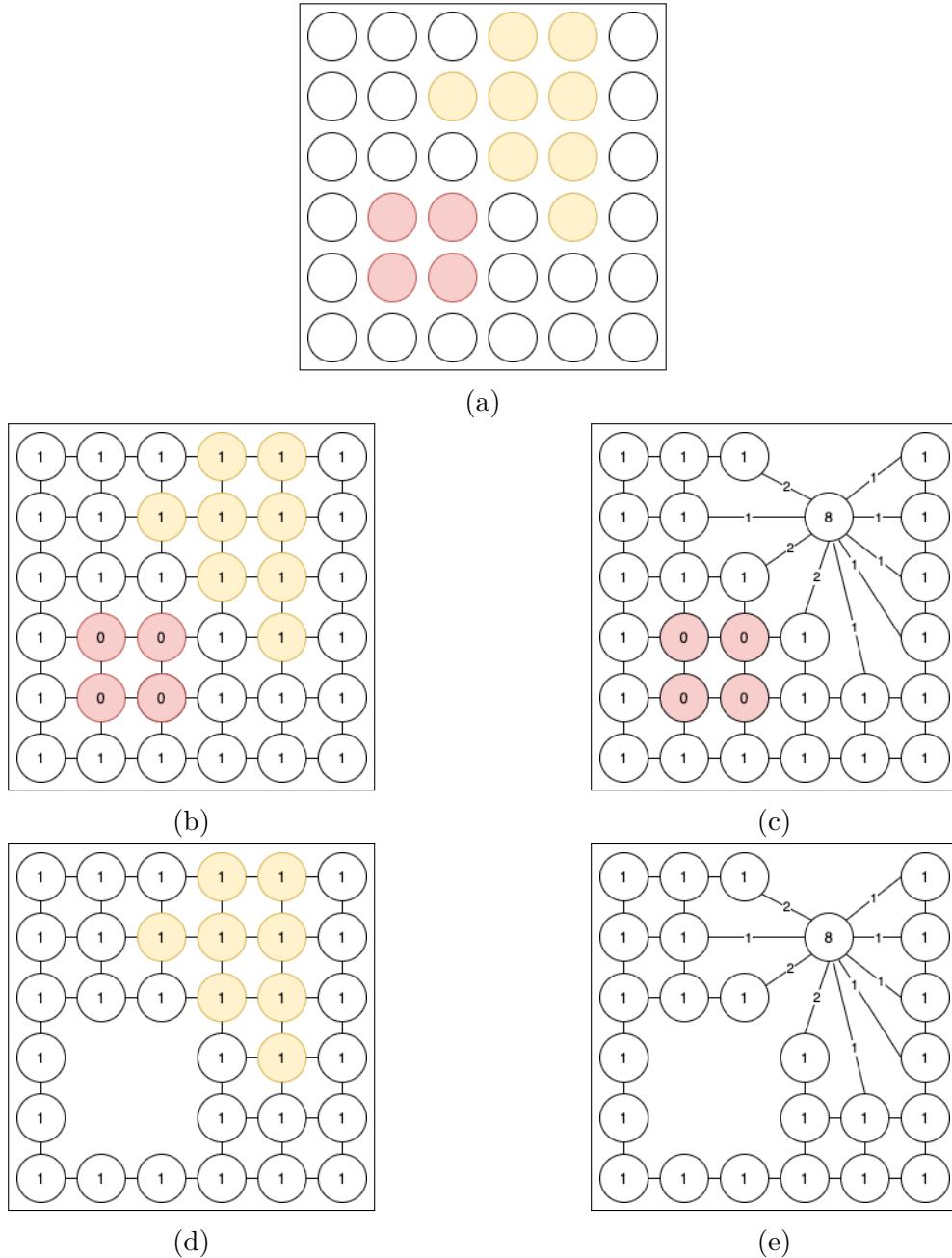
Rysunek 8: Obrazek, który reprezentuje strukturę siatki do podziału. Żółte obszary to obszary niepodzielne, czerwone to te wyłączone z obliczeń, białe to zwykłe przez które przechodzić będą granice podziału.

Algorytm iteruje po pikselach obrazka wejściowego i na podstawie ich koloru tworzy wierzchołki grafu oraz buduje zbiory obszarów niepodzielnych oraz wyłączonych z obliczeń. W zbiorach przechowywane są numery wierzchołków. Na początku każda krawędź pomiędzy wierzchołkami, a także wierzchołki zwykłe oraz wierzchołki obszarów niepodzielnych otrzymują wagę 1. Wierzchołki obszarów wyłączonych z obliczeń mogą mieć przyporządkowaną wagę 0 lub nie być mapowane na wierzchołki w grafie, tworząc puste przestrzenie między wierzchołkami. Każdy wierzchołek w grafie ma parametr, który określa, jakim jest typem obszaru. Kolejnym etapem jest redukcja obszarów niepodzielnych do pojedynczych wierzchołków. Każdy obszar niepodzielny zamieniany jest na wierzchołek o wadze równej sumie wag wierzchołków tego obszaru. Od teraz wierzchołki, które reprezentują obszary niepodzielne, nie są rozróżniane jako obszary niepodzielne - stają się zwykłymi wierzchołkami. Ten proces przedstawiony jest na rysunku 10.

W dalszych częściach algorytmu następuje partycjonowanie oraz przemieszczanie wierzchołków między partycjami. Wszystkie te operacje będą następowały na grafie z co najmniej zredukowanymi obszarami niepodzielnymi. W ten sposób obszary niepodzielne są zawsze traktowane jako całość, a pojedyncze wierzchołki nigdy nie zostaną od nich odłączone. Wierzchołki wyłączone z obliczeń mają wagę 0, ponieważ algorytmy wykorzystywane później do partycjonowania grafu, operują głównie na parametrze wagi wierzchołków. Waga wierzchołka w zredukowanym grafie reprezentuje liczbę wierzchołków z grafu początkowego, co ma wpływ na balansowanie pól obszarów. Chcemy, aby partie były równe pod względem liczby wierzchołków, na których będą odbywać się obliczenia. Waga 0 wierzchołka oznacza, że nie jest on brany pod uwagę w wyrównywaniu pól obszarów, nie ma znaczenia dla algorytmu i może być dowolnie przemieszczany pomiędzy partycjami.



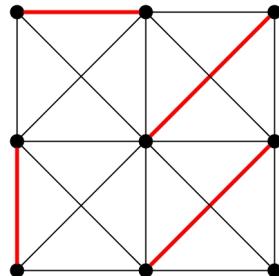
Rysunek 9: Obrazek (a) przedstawia obrazek wejściowy w rzeczywistym rozmiarze. Obrazki (b) oraz (c) przedstawiają powstały graf. Obrazek (b) przedstawia sposób podejścia do obszarów wyłączonych z obliczeń, gdzie nie tworzone są odwzorowujące je wierzchołki, obrazek (c) pokazuje podejście gdzie w ich miejscu tworzone są wierzchołki z wagą 0. Dla czytelności wierzchołki zwykłe zostały oznaczone kolorem szarym.



Rysunek 10: Obrazek (a) to wejściowa siatka. Obszary niepodzielne są żółte, natomiast wyłączone z obliczeń czerwone. Obrazki (b) oraz (d) pokazują siatkę tuż po konwersji na graf na dwa sposoby. Sposób (b) nadaje wierzchołkom na obszarach wyłączonych z obliczeń wagę 0, natomiast sposób (d) usuwa wierzchołki obszarów wyłączonych z obliczeń. Na wierzchołkach zaznaczona jest waga. Można zaobserwować wagę 1 dla wierzchołków zwykłych i tych budujących obszary niepodzielne. Na tym etapie wszystkie krawędzie mają wagę 1. Obrazek (c) oraz (e) przedstawiają ten sam graf po redukcji obszarów niepodzielnych, odpowiednio z kroku (b) oraz (d). Wierzchołek, do którego zostało zredukowane obszar niepodzielny otrzymuje powiększoną wagę, równą sumie wag wierzchołków, które redukuje. Każda krawędź, która zastąpiła dwie krawędzie, zyskuje sumę wag tych krawędzi równą 2.

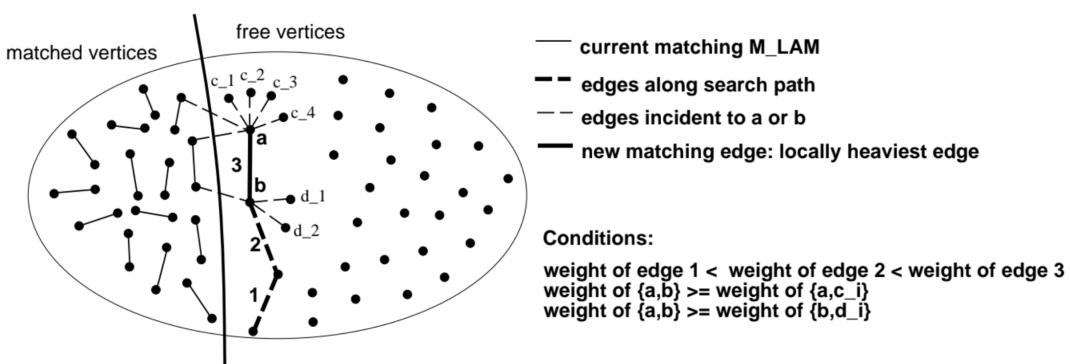
### 3.1.2. Zmniejszanie i partycjonowanie grafu za pomocą algorytmu LAM

Aby stworzyć mniejszy odpowiednik wejściowego grafu stosowany jest algorytm tworzący skojarzenia (ang. matching). Skojarzenie to podzbiór krawędzi grafu (ozn.  $M$ ) o tej właściwości, że każdy wierzchołek jest końcem co najwyżej jednej krawędzi z  $M$ . Pary wierzchołków połączone bezpośrednio krawędzią należącą do  $M$  są skojarzone przez  $M$  [34].



Rysunek 11: Skojarzenie największe, którego liczba krawędzi wynosi 4. Źródło: [34].

W celu zmniejszenia grafu stosowana jest heurystyka, która bazuje na metodzie [28]. Algorytm ten oblicza 2-approximation dla problemu maximum weighted matching w czasie liniowym. Startuje on od arbitralnej krawędzi i sprawdza sąsiednie krawędzie. Tak dugo, jak jest w stanie znaleźć sąsiednią krawędź z wyższą wagą, wywołuje się na niej i powtarza tę procedurę, aż znajdzie krawędź z lokalnie najwyższą wagą. Ta krawędź łączy dwa wierzchołki, które zostaną skojarzone. Skonstruowany w ten sposób algorytm dąży do budowania możliwie "zwartych" obszarów, co oznacza to, że granice między obszarami są możliwie krótkie. Krawędzie z wysokimi wagami łączą wierzchołki, reprezentujące zbiory wierzchołków, które w początkowym grafie miały między sobą długą granicę. Działanie algorytmu pokazany jest na rysunku 15.



Rysunek 12: Fragment przebiegu algorytmu LAM. Startując od wybranej arbitralnie krawędzi numer 1, ścieżka buduje się przez krawędzie z lokalnie wyższymi wagami - krawędź numer 2 oraz 3 - aż do znalezienia krawędzi  $\{a, b\}$  z lokalnie największą wagą. (Pokazywane są tylko krawędzie z aktualnego wywołania algorytmu LAM, krawędzi znajdujące się na ścieżce oraz krawędzie sąsiadujące do  $\{a, b\}$ ). Źródło: [28].

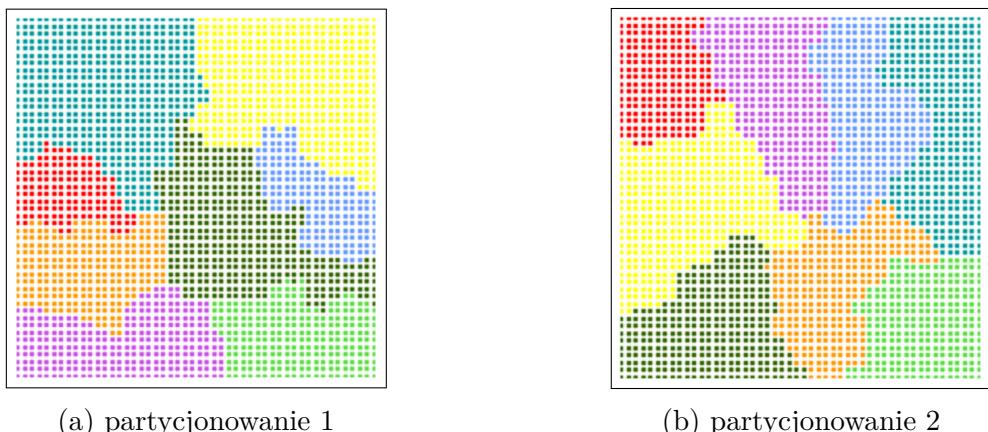
```

1 PROCEDURE shrink graph ( $G$ )
2    $t = 0$  /* number of episodes */
3   WHILE  $|V_G| \neq$  number of partitions we want to achieve
4     matched_vertices = run LAM-Algorithm on  $G$ 
5     shrink  $G$  based on the matched_vertices
6      $t = t + 1$ 
7   ENDWHILE
8
9
10 LAM-Algorithm
11    $M_{LAM} := \emptyset$ ; /* empty matching at the start */
12    $U := E$ ; /* all edges are unchecked at the start */
13   WHILE ( $U \neq \emptyset$ )
14     take arbitrary edge  $\{a, b\} \in U$ ;
15     try match ( $\{a, b\}$ );
16   ENDWHILE
17
18
19 PROCEDURE try match ( $\{a, b\}$ )
20    $C_{\{a,b\}}(a) := \emptyset$ ;  $C_{\{a,b\}}(b) := \emptyset$ ; /* empty local sets of checked edges at the
21   start */
22   WHILE ( $a$  is free AND  $b$  is free AND ( $\exists \{a, c\} \in U$  OR  $\exists \{b, d\} \in U$ ))
23     IF ( $a$  is free AND  $\exists \{a, c\} \in U$ )
24       move  $\{a, c\}$  from  $U$  to  $C_{\{a,b\}}(a)$ ; /* move from  $U$  to  $C$  */
25       IF ( $w(\{a, c\}) > w(\{a, b\})$ )
26         try match ( $\{a, c\}$ ); /* call heavier edge */
27       ENDIF
28     ENDIF
29     IF ( $b$  is free AND  $\exists \{b, d\} \in U$ )
30       move  $\{b, d\}$  from  $U$  to  $C_{\{a,b\}}(b)$ ; /* move from  $U$  to  $C$  */
31       IF ( $w(\{b, d\}) > w(\{a, b\})$ )
32         try match ( $\{b, d\}$ ); /* call heavier edge */
33       ENDIF
34     ENDIF
35   ENDWHILE
36   IF ( $a$  is free AND  $b$  is free AND  $\{a, b\}$  can be matched)
37     add  $\{a, b\}$  to  $M_{LAM}$  /* new matching edge  $\{a, b\}$  */
38   ELSE IF ( $a$  is matched AND  $b$  is free)
39     move edges  $\{\{b, d\} \in C_{\{a,b\}}(b) \mid d$  is free $\}$  back to  $U$ ;
40   ELSE IF ( $b$  is matched AND  $a$  is free)
41     move edges  $\{\{a, c\} \in C_{\{a,b\}}(a) \mid c$  is free $\}$  back to  $U$ ;

```

Algorytm 1: Kod przedstawiający ulepszony algorytm LAM. Wprowadzone przeze mnie zmiany dotyczyły warunku skojarzenia wierzchołków oraz końcowych instrukcji warunkowych, które mogły zostać uproszczone z racji na brak potrzeby tworzenia zbioru krawędzi do usunięcia -  $R$ . Niezmodyfikowany kod algorytmu można znaleźć w artykule [28].

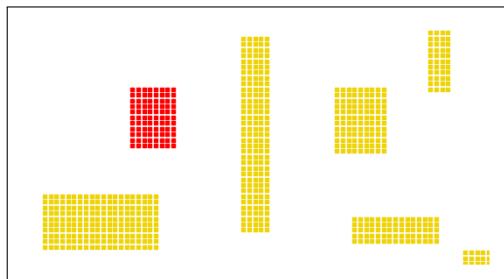
Algorytm 1 (opisany w [28]) przedstawia procedury służące do zmniejszania grafu. Końcowe efekty jego działania można obserwować na rysunku 13 oraz 14. Został ulepszony o bardziej rozbudowany warunek łączenia wierzchołków oraz uproszczony o usunięcie zbioru  $R$ , odpowiedzialny za przechowywanie krawędzi do usunięcia. Startuje z pustym zbiorem skojarzeń  $M_{LAM}$ . Zbiór  $U$  przechowuje nieodwiedzone krawędzie ( $U = E$  na startie). Główną częścią algorytmu jest pętla WHILE (linia numer 13), która wywołuje procedurę 'try match' z arbitralnie wybraną, nieodwiedzoną krawędzią  $\{a, b\}$ . Ta krawędź nie jest dodawana do zbioru skojarzeń, dopóki wszystkie sąsiednie krawędzie prowadzące do wolnych wierzchołków (ang. free vertices) nie są sprawdzone w zakresie poszukiwania nowej krawędzi z wyższą wagą - następuje to w pętli WHILE w linii numer 21. Każde wywołanie procedury 'try match' ( $\{a, b\}$ ) przechowuje swoje własne zbiory lokalnie odwiedzonych krawędzi  $C_{\{a,b\}}(a)$  oraz  $C_{\{a,b\}}(b)$ , w zależności od tego, czy nieodwiedzone krawędzie są sąsiadujące z wierzchołkiem  $a$  czy z  $b$ . Jeśli wierzchołki  $a$  i  $b$  są wolne oraz co najmniej jeden z nich jest wierzchołkiem należącym do jednej z nieodwiedzonych krawędzi, nieodwiedzona krawędź jest sprawdzana dalszymi instrukcjami. Jeśli ma wyższą wagę od krawędzi  $\{a, b\}$ , 'try match' wywołuje się na niej rekursywnie. Rekursywne wywołania są powtarzane aż do znalezienia krawędzi z lokalnie najwyższą wagą. Następnie krawędź dodawana jest do  $M_{LAM}$ , a algorytm kończy aktualne wywołanie 'try match' i kontynuuje pętlę WHILE sprawdzając kolejne sąsiadujące krawędzie. Pętla WHILE kończy się wtedy, gdy  $a$  oraz/lub  $b$  zostaną skojarzone w rekursywnym wywołaniu, lub jeśli nie mają więcej sąsiednich, nieodwiedzonych krawędzi. Następnie w końcowej części składającej się z instrukcji warunkowych następuje sprawdzenie czy  $a$  oraz  $b$  są wolne. Jeśli tak, to  $\{a, b\}$  jest dodawane do  $M_{LAM}$ . W tej części, w oryginalnym algorytmie uzupełniany jest zbiór wierzchołków do usunięcia [28] -  $R$ . Ponadto jeśli  $a$  lub  $b$  są wolne, to wszystkie krawędzie, które zostały odwiedzone w aktualnym wywołaniu, zawierające w sobie wierzchołek  $a$  lub  $b$ , zostają oznaczone jako nieodwiedzone - są przenoszone do zbioru  $U$ .



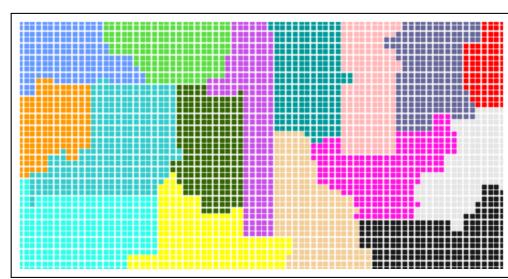
Rysunek 13: Dwa losowo wybrane partycjonowania siatki  $50 \times 50$  na 8 obszarów za pomocą samego algorytmu LAM - bez fazy ulepszania podziału. Granice podziałów nie są optymalne, pola obszarów nie są równe, lecz bliskie bycia równym pod względem wielkości pól partycji, co jest oczekiwana i dobrą bazą do dalszych operacji. Algorytm ma charakter losowy, co znaczy, że każde wywołanie będzie dawać inny rezultat. Zawsze jednak zachowane będzie dążenie do równych i zwartych obszarów.

Pętla WHILE w 21 linii sprawdza krawędzie sąsiadujące z wierzchołkiem  $a$  oraz  $b$  tylko jeśli obydwa wierzchołki są wolne. Ta cecha jest używana przez autorów artykułu do udowodnienia liniowej złożoności algorytmu.

Nieodwiedzone krawędzie zawarte w  $U$  mają taką właściwość, że obydwa należące do nich wierzchołki są wolne. Nowe krawędzie są przenoszone ponownie do  $U$  w końcowej części z instrukcjami warunkowymi, ale tylko jeśli obydwa ich wierzchołki są wolne. Tak skonstruowany algorytm jest uruchamiany wielokrotnie przez procedurę 'shrink graph', aż do momentu, gdy liczba wierzchołków w grafie osiągnie liczbę partycji, na które chcemy podzielić wejściową siatkę. Po każdym wywołaniu algorytmu LAM uruchamiana jest procedura, która zamienia skojarzone pary wierzchołków na pojedyncze wierzchołki wraz ze zmianą wagi. Wyjątkiem jest ostatnie wywołanie, kiedy liczba wierzchołków w grafie osiąga wartość docelową. Wtedy łączonych jest pierwszych  $n$  par wierzchołków, tak by otrzymać wymaganą liczbę wierzchołków. Waga wierzchołka powstała ze skojarzenia jest sumą wag wierzchołków kojarzonych. Jeśli na skutek skojarzenia powstaje krawędź, która zastępuje kilka krawędzi to jej waga jest sumą wag tych krawędzi. Następnie do wierzchołków przyporządkowywane są numery partycji. Zwykle jedno wywołanie algorytmu LAM tworzy liczbę skojarzeń niewiele mniejszą niż połowa liczby wierzchołków w grafie. W ten sposób graf, szczególnie w początkowych wywołaniach, zmniejszany jest o około połowę po każdym wywołaniu algorytmu LAM.

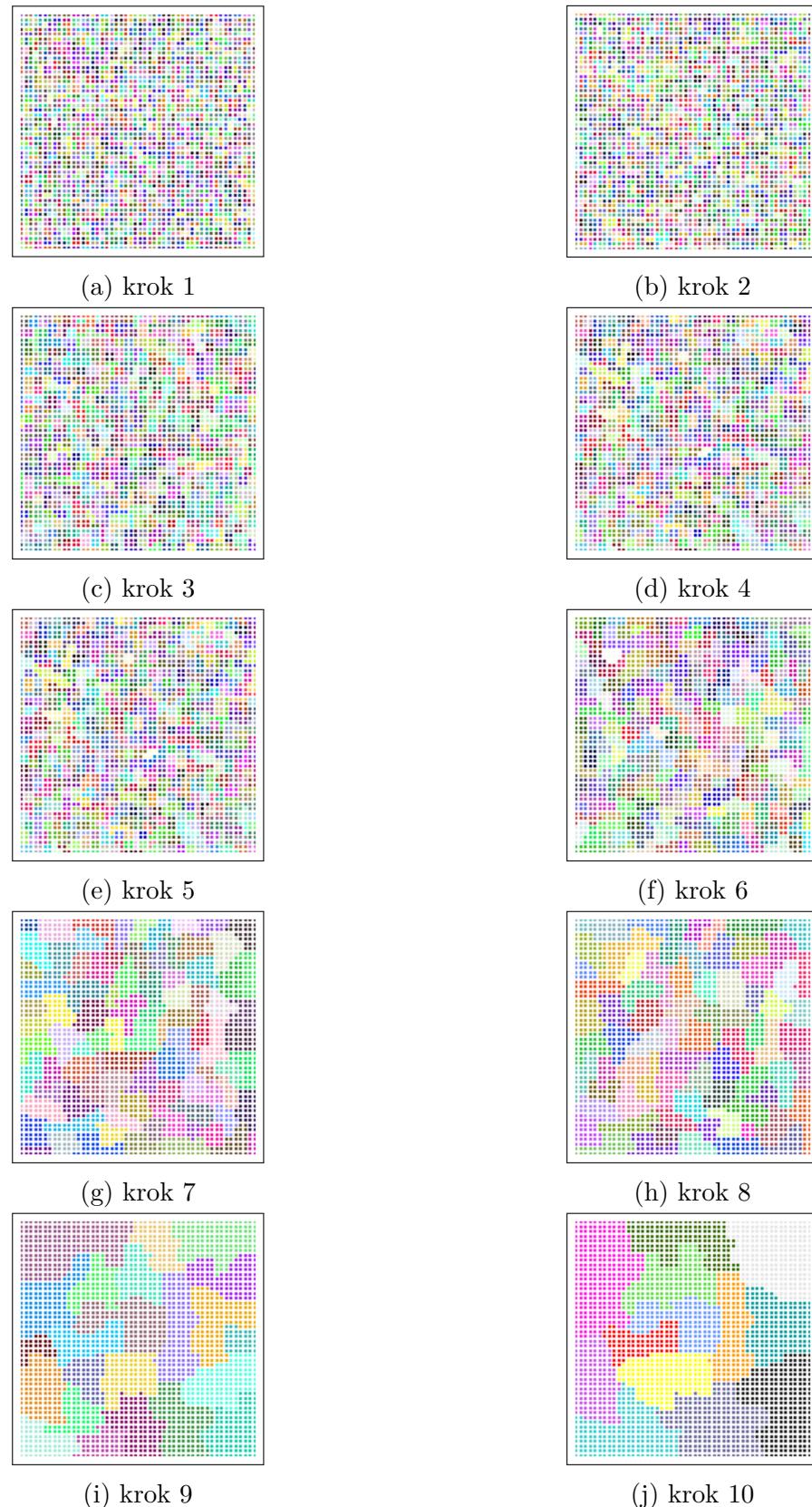


(a) siatka do partycjonowania



(b) partycjonowanie

Rysunek 14: Obrazek (b) przedstawia partycjonowanie siatki (a) samym algorytmem LAM. Widoczne jest uwzględnianie obszarów niepodzielnych, które na siatce (a) zaznaczone są jako obszary żółte.



Rysunek 15: Obrazki przedstawiają wybrane, kolejne kroki algorytmu LAM aż do otrzymania podziału siatki  $50 \times 50$  na 13 obszarów. Przedstawione zostało co siódme wywołanie algorytmu oraz efekt końcowy. Widoczne jest równomierne łączenie obszarów.

### 3.1.3. Szczegóły modyfikacji algorytmu LAM

Algorytm ten bez dodatkowego warunku na skojarzenie wierzchołków (linia 35) zwykle tworzyłby grafy typu gwiazda, co oznacza, że powstawałby część wierzchołków o bardzo wysokim stopniu. To powodowałoby, że zmniejszony graf nie przypominałby początkowego grafu. Warunek ten został dodany przez autorów artykułu [28] i mówi, że skojarzenie wierzchołka  $a$  z wierzchołkiem  $b$  może zaistnieć tylko wtedy, jeśli ich sumaryczna waga nie przekracza dwukrotności najmniejszej wagi wierzchołka w grafie, zsumowanej z największą wagą wierzchołka w grafie.

$$w_a + w_b \leq w_{\text{highest}} + 2 \cdot w_{\text{lowest}} \quad (1)$$

Warunek 1 powoduje, że nawet jeśli rozpatrujemy skojarzenie wierzchołka o relatywnie wysokiej wadze, to ma on szansę zostać skojarzony jedynie z wierzchołkiem o wadze niższej, co prowadzi do równiejszych wag w grafie. W dalszych etapach zmniejszania grafu trudniej spełnić warunek na skojarzenie wierzchołków - pojawia się więcej różnorodności w zakresie wag wierzchołków grafu, a co za tym idzie, spada liczba udanych skojarzeń przypadających na wywołanie. Kiedy zbyt mało par jest kojarzonych podczas pojedynczego wywołania, warunek ten jest osłabiany poprzez zwiększenie dopuszczalnej sumarycznej wagi wierzchołka  $a$  oraz  $b$ .

Warunek stworzony przez autorów artykułu [28] okazał się nie działać w kontekście obszarów niepodzielnych. W tym celu stworzony został nowy, zmodyfikowany warunek 3, bazujący na warunku 1 rozszerzonym o *discount* (wzór 2).

$$\text{discount} = \frac{t}{T \cdot \frac{w_{\text{highest}}}{w_{\text{lowest}} + 1} \cdot \log(\text{number\_of\_partitions}) + 1} \quad (2)$$

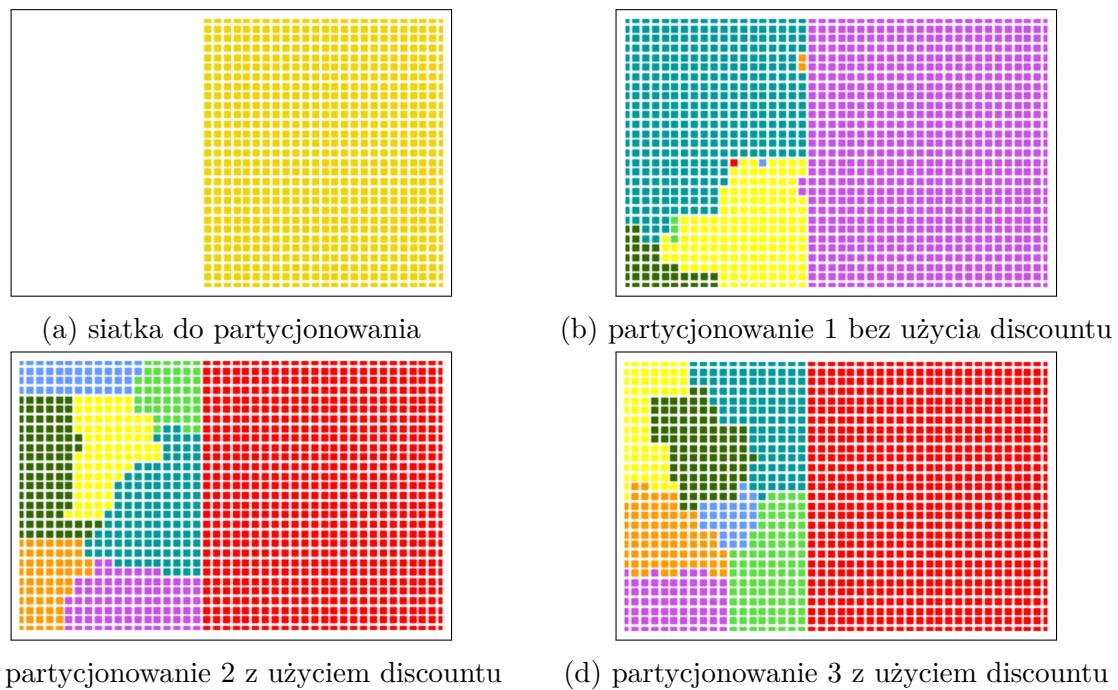
Jeśli  $\text{discount} > 1$  wtedy ma przypisywaną wartość 1.  $T$  to przewidywana liczba wywołań algorytmu LAM. Obliczana jest jako liczba dzieliń liczby wierzchołków grafu przez liczbę dwa, potrzebną do uzyskania liczby wierzchołków mniejszej lub równej docelowej liczbie partycji.  $t$  to licznik wywołań algorytmu LAM (linia 2). Z użyciem *discountu* powstał zmodyfikowany przeze mnie warunek na skojarzenie wierzchołków  $a$  i  $b$ :

$$w_a + w_b \leq \text{discount} \cdot (w_{\text{highest}} + 2 \cdot w_{\text{lowest}}) \quad (3)$$

Niedziałanie poprzedniego warunku jest spowodowane faktem, iż w początkowej fazie algorytmu, tuż po zamianie wejściowej siatki na graf, wszystkie obszary niepodzielne zamieniane są na pojedyncze wierzchołki. Waga dla wierzchołków reprezentujących obszary niepodzielne jest równa sumie wag wierzchołków, które do nich należą. Oryginalny algorytm LAM został zaprojektowany z założeniem równomiernego budowania skojarzeń ze względu na wagę wierzchołków w grafie od samego początku. W warunku na skojarzenie wierzchołków bierze pod uwagę wierzchołek z aktualnie największą wagą oraz aktualnie najmniejszą wagą w grafie. Warunek ten funkcjonuje poprawnie, jeśli graf jest zmniejszany równomiernie od samego początku, kiedy wagi wszystkich wierzchołków w grafie mają taką samą lub niemal taką samą wartość. W momencie, w którym na samym początku pojawiają się wierzchołki o dużych wagach (z powodu zamiany wierzchołków należących

do obszarów niepodzielnych na pojedyncze wierzchołki o wyższych wagach) warunek 1 zaczyna działać niepoprawnie. W szczególnym przypadku, jeśli taki obszar zajmuje więcej niż połowę całej siatki (rysunek 16) niemal każde skojarzenie dwóch wierzchołków, z których żaden nie jest reprezentującym obszar niepodzielny, będzie dozwolone. Każdy z takich wierzchołków ma bowiem wagę mniejszą niż suma podwojonej najmniejszej i największej wagi wierzchołka w grafie. Warunek 1 jest więc niemal zawsze spełniony. W tego typu przypadkach realny podział odbywa się tak naprawdę poza obszarem niepodzielnym, ponieważ wiemy, że będzie on na końcu jedną partycją, a jako że jest największym wierzchołkiem w grafie - nie będzie mógł tworzyć wielu skojarzeń. Prowadzi to do kojarzenia obszarów bez zachowania równomiernych pól jak na rysunku 16(b). W efekcie w skład wielu partycji wchodzi tylko jeden wierzchołek.

Zaproponowana przeze mnie modyfikacja (wzór 2), polegająca na dodaniu *discountu*, wzmacnia warunek i powoduje bardziej równomierne (ze względu na wagi) łączenie się wierzchołków. *discount* zależny jest od liczby iteracji - osłabia się w czasie. Im późniejsza iteracja tym bliższy jest wartości 1, a więc zmierza do oryginalnego warunku 1. Ważne jest jednak, że w kluczowych, początkowych wywołaniach algorytmu LAM pozwala na równomierne skojarzenia. Efekt działania warunku widoczny jest na rysunku 16.



Rysunek 16: Obrazek (a) przedstawia siatkę wejściową, której więcej niż połowę zajmuje oznaczony kolorem żółtym obszar niepodzielny. Obrazek (a), (b) oraz (c) przedstawia partycjonowanie siatki na 8 części wykonane algorytmem LAM bez fazy ulepszania podziału. Obrazek (b) pokazuje partycjonowanie wykonane z użyciem warunku 1 na skojarzenie wierzchołków. Obraz (c) oraz (d) prezentują partycjonowanie wykonane z użyciem warunku 3.

Negatywnym efektem stosowania *discountu* jest większa liczba wywołań algorytmu LAM. Czasami zmienna  $t$  musi urosnąć do konkretnej wartości, umożliwiającej dokonanie kolejnych, blokowanych obecnie skojarzeń, a żadne inne skojarzenia, które spełniałyby warunek nie są w danej chwili możliwe. Rysunek 16 pokazuje pozytywne działanie modyfikacji na jakość podziału wraz z porównaniem do poprzedniej wersji warunku.

Skonstruowanie *discountu* polegało na odpowiednim dobraniu jego wartości względem parametrów dzielonej siatki, tak by nie działał zbyt agresywnie, nadmiernie przedłużając obliczenia, a jednocześnie był na tyle agresywny, aby pozwalał tylko na kojarzenie wierzchołków prowadzące do możliwie równych podziałów. Został skonstruowany na bazie podstawowego współczynnika  $\frac{t}{T}$ , który na podstawie wielu prób dostosował dodatkowymi zmiennymi w poszukiwaniu wcześniej opisywanego balansu. W momencie, gdy partycjonujemy obszar, w którym nie ma dużych obszarów niepodzielnych, nie musimy stosować *discountu*. W takim wypadku jedynym jego efektem będzie większa liczba wywołań algorytmu LAM.

### 3.1.4. Przywracanie grafu do początkowego rozmiaru wraz z ulepszeniem podziału

Kolejną częścią algorytmu jest faza przywracania grafu do początkowej wielkości. Tej fazie towarzyszy poprawianie podziału, które polega na zmniejszaniu długości granic oraz na wyrównywaniu wielkości pól pomiędzy obszarami. Ten etap nadaje podziałowi ostateczny kształt. W tym celu zaimplementowana została zmodyfikowana heurystyka Helpful Sets (HS) [30]. Efekt jej działania widoczny jest na rysunku 17.



Rysunek 17: Obrazek (a) przedstawia partycjonowanie siatki (a) samym algorytmem LAM. Obrazek (b) przedstawia ulepszenie podziału z obrazka (a). Zmniejszona została długość granic oraz wyrównane zostały pola.

Etap pierwszy to heurystyka HS do poprawiania podziału. Jest oparta na wyszukiwaniu lokalnym - próbuje poprawić aktualny podział poprzez wielokrotne wprowadzanie lokalnych zmian. Wierzchołki klasyfikowane są pod kątem liczby krawędzi łączących je z wierzchołkami obszaru do którego należą (ang. internal edges) oraz z wierzchołkami pozostałych obszarów (ang. external edges). Wedle tej klasyfikacji wierzchołki przy granicach wymieniane są między obszarami. Niech  $A$  i  $B$  będą partycjami na siatce. Algorytm dzieli się na dwie fazy. Pierwsza buduje zbiór wierzchołków  $C$  na wierzchołkach partycji  $A$ , a następnie przenosi go z partycji  $A$  do partycji  $B$ . Wierzchołki są dobierane tak, aby zmniejszyć długość granicy między obszarami. Druga faza to szukanie zbioru balansującego  $D$ . Jest budowany na zbiorze  $B$ . Jego celem jest zwrócenie zbioru wierzchołków o tej samej wielkości jak zbiór  $C$  z partycji  $B$  do partycji  $A$ , przy jednoczesnym możliwym utrzymaniu ulepszenia w długości granicy po wcześniejszej fazie. Zbiór  $C$  nazywamy **k-helpful set**, natomiast zbiór  $D$  nazywamy **balancing set**. Po etapie ulepszenia granicy następuje etap drugi mający na celu wyrównywanie powierzchni pól. Używając tej samej klasyfikacji, wierzchołki przy granicach przenoszone są z obszarów mniejszych do większych. Te dwa etapy powtarzane są co jakiś czas podczas przywracania grafu do początkowej wielkości. Najpierw więc pracują na grafie o mniejszej liczbie wierzchołków, gdzie każdy wierzchołek reprezentuje zbiór wierzchołków. W ten sposób efektywnie przenoszone są duże obszary w formie pojedynczych wierzchołków o dużych wagach. Im dalej w procesie przywracania grafu, tym algorytm pracuje na większym grafie z wierzchołkami o mniejszych wagach. Podział jest już ulepszony w poprzednich wywołaniach więc może dokonywać bardziej lokalnych poprawek.

```

1 PROCEDURE RestoreGraphWithPartitionsImprovements
2   number_of_restoration_steps_done ← 0
3   number_of_all_reductions ← reductions.length
4   WHILE number_of_all_reductions – number_of_restoration_steps_done > 0
5     reduction = reductions.pop()
6     RestoreReduction(reduction)
7     number_of_restoration_steps_done ← number_of_restoration_steps_done + 1
8     IF number_of_restoration_steps_done / number_of_all_reductions > 0.9 AND
9       floor(number_of_all_reductions mod (number_of_all_reductions · 0.03)) == 0
10    ImprovePartitioning
11  ENDIF
12 ENDWHILE
13 ImprovePartitioning
14 WHILE there are any indivisible areas:
15   area ← pop one of them
16   RestoreReduction(area)
17 ENDWHILE

```

Algorytm 2: Algorytm przedstawia główną procedurę optymalizującą długość granic.

Na algorytmie 2 przedstawiona jest główna procedura odpowiedzialna za przywrócenie grafu do początkowej wielkości wraz z wprowadzaniem optymalizacji granic. Autorzy artykułu [24], na podstawie którego stworzona została niniejsza praca, wyspecyfikowali, że do optymalizacji granic używany jest algorytm HelpfulSets, który wywoływany jest wielokrotnie na różnych etapach przywracania grafu. Nie było natomiast informacji jak często oraz w jakich stadiach przywracania grafu wywoływany jest algorytm. W związku z powyższym zaproponowałem wyżej opisane rozwiązanie. Dodatkową modyfikacją algorytmu przywracania grafu było również uwzględnienie obszarów niepodzielnych. Pierwsza pętla WHILE (linia 4) odpowiedzialna jest za przywrócenie zwykłych redukcji, to znaczy tych, które zostały stworzone przez algorytm LAM oraz wprowadzenie ulepszeń. Inne redukcje to te, które wykonywane są na obszarach niepodzielnych zaraz po algorytmie, który zamienia obrazek przedstawiający siatkę na graf. Redukcje te przywracane są na samym końcu, po przeprowadzeniu optymalizacji granic.

Procedura `ImprovePartitioning` za każdym razem aktualizuje informacje o sąsiadujących obszarach, ponieważ te mogą ulec zmianie na skutek kolejnych wywołań algorytmu, a następnie iteruje po wszystkich parach sąsiadujących obszarów i wywołuje dla nich procedurę `HelpfulSet`. Pseudokod procedury `HelpfulSet` został przedstawiony na algorytmie 3. Jak wynika z kodu (linia 8) proces optymalizacji długości granic oraz pól obszarów zaczyna się dopiero, gdy graf przywrócony jest do początkowej wielkości przynajmniej w 90%. Tą wartością można manipulować dowolnie, jednak powinna być ona wysoka. W momencie, kiedy pozwolimy algorytmowi optymalizować bardzo zredukowane grafy, to przesuwane na raz będą bardzo duże pule wierzchołków. Algorytm do optymalizacji granic został stworzony do wprowadzania małych, lokalnych zmiany, a nie przesuwania dużych obszarów. Dlatego więc przy optymalizowaniu zredukowanych grafów niszczyc podziały, zamiast go poprawiać. Efektem tego jest powstawanie obszarów podzielonych na dwie części. Zjawisko to przedstawione jest na rysunku 28 i będzie dokładnej opisane

w dalszych częściach pracy. W linii 9 widzimy także, że optymalizacja przeprowadzana jest co pewną liczbę przywróconych redukcji. Bowiem przywrócenie jednej redukcji, to zamiana jednego wierzchołka na dwa wierzchołki, które zastąpiła. Jest więc to zmiana zbyt mała, żeby ponownie włączać optymalizację dla całego grafu. Na podstawie przeprowadzonych prób można stwierdzić, że współczynniki powinny być ustawione tak, aby wywoływać algorytm optymalizacji granic 5-10 razy podczas przywracania grafu. Takie wartości gwarantują dobry wynik przy niezbyt długim czasie wykonania. Więcej wywołań niekoniecznie da lepszy rezultat. Ta funkcjonalność została zaimplementowana przy użyciu operacji wyliczania reszty z dzielenia. W linii 13 następuje ostatnie wywołanie algorytmu optymalizacji granic, już na niemal całkowicie przywróconym grafie (nie licząc obszarów niepodzielnych). Ostatnim krokiem jest przywrócenie obszarów niepodzielnych, co dzieje się w drugiej pętli WHILE w linii 14.

### 3.1.5. Używanie Helpful Sets w celu zmniejszenia długości granic

Heurystyka Helpful Sets startuje od początkowego podziału i za pomocą lokalnie wprowadzanych zmian zmniejsza długość granicy. Algorytm startuje od poszukiwania zbioru **k-helpful**, to jest podzbioru wierzchołków ze zbioru  $V_1$  lub  $V_2$ , który zmniejsza długość granicy o  $k$ , jeśli zostanie przeniesiony do drugiej partycji. Jeśli taki zbiór zostaje znaleziony w jednej z dwóch partycji, na przykład w  $V_1$ , to jest przenoszony do  $V_2$ . Następnie algorytm zaczyna szukać w części  $V_2$ , która jest aktualnie wystarczająco duża, aby zbalansować podział i zwiększyć długość granicy o maksymalnie ( $k - 1$ ) krawędzi. Jeśli taki **balancing set** zostaje znaleziony, to przenoszony jest do zbioru  $V_1$  i proces zaczyna się od początku. Działanie algorytmu dobrze widać na rysunku 19 oraz 20. Zanim algorytm zostanie opisany dokładniej, potrzebne są definicje zbiorów helpful oraz balancing:

**Definicja 1** (*diff-value wierzchołka*)

Dla wierzchołka  $v \subset V$  wartość *diff-value* definiujemy jako  $diff(v) = ext(v) - int(v)$ .

**Definicja 2** (*Helpful Set*)

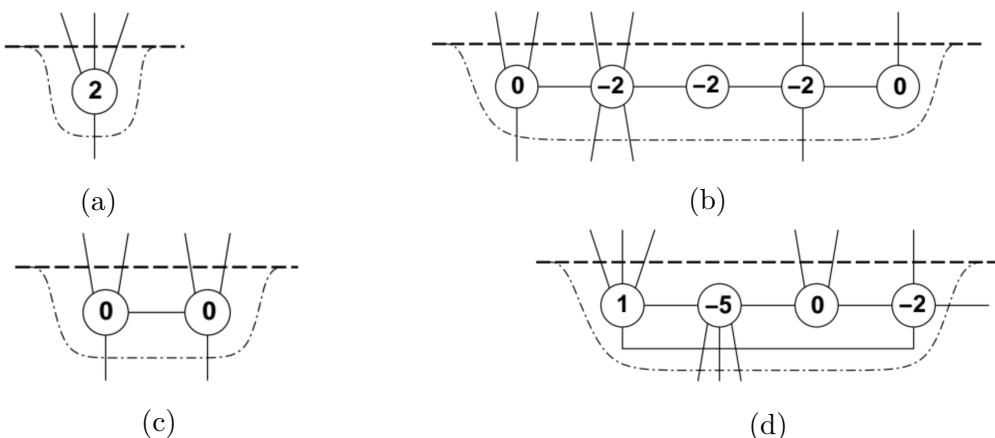
Niech  $S \subset V_i, i \in \{1, 2\}$  będzie podzioborem wierzchołków partycji. Dla każdego  $v \in S$  definiujemy  $int_s(v) = |\{w \in S; \{v, w\} \in E\}|$  - zbiór krawędzi wewnętrznych  $S$ .

$$H(S) = \sum_{v \in S} (ext(v) - int(v) + int_s(v)) \quad (4)$$

to **helpfulness** zbioru  $S$ . Zbiór  $S$  nazywany jest  $H(S)$ -**helpful**.

Przeniesienie zbioru  $S$  do drugiej partycji zmniejsza długość granicy o  $H(S)$ . Przykłady zbiorów 2-helpful pokazane są na rysunku 18.

Autorzy artykułu [6] zamienne używają pojęć diff-value oraz helpfulness. W niniejszej pracy używana jest nazwa helpfulness zarówno dla wierzchołków, jak i dla zbiorów.

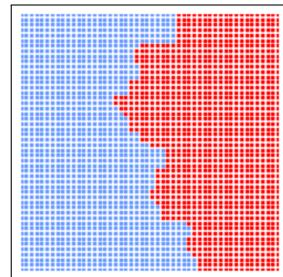


Rysunek 18: Zbiory 2-helpful. Na górze każdego z 4 przykładów jest zbiór zewnętrzny (ang. external set). Wyliczanie wartości helpfulness na podstawie wzoru 4: (a):  $3 - 1 + 0 = 2$ , (b):  $6 - 8 + 4 = 2$ , (c):  $4 - 3 + 1 = 2$  oraz (d):  $6 - 8 + 4 = 2$ . Na każdym wierzchołku zaznaczona jest jego wartość helpfulness. Źródło: [6].

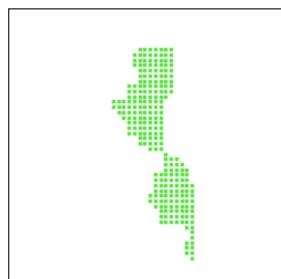
**Definicja 3 (Balancing Set)**

Niech  $S \in V_i$  będzie zbiorem  $k$ -helpful. Zbiór  $\bar{S} \subset V_j \cup S$ ,  $J \neq i$  nazywany jest **balancing set** zbioru  $S$  jeśli  $|\bar{S}| = |S|$  and  $\bar{S}$  jest przynajmniej  $(-k+1)$ -helpful.

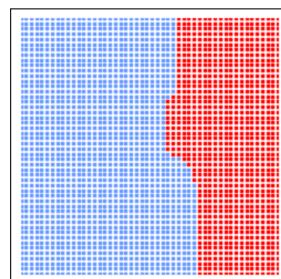
Jedną z cech algorytmu jest, że długość granicy zwiększa się o nie więcej niż  $(k - 1)$  jeśli zbiór  $\bar{S}$  przenoszony jest z jednej partycji do drugiej.



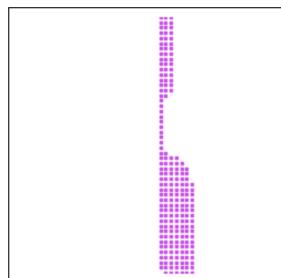
(a) początkowy podział



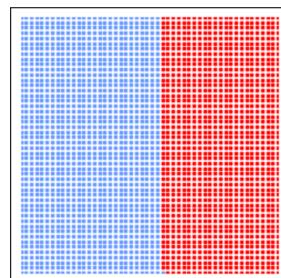
(b) krok 1 - budowanie zbioru helpful



(c) krok 2 - przeniesienie zbioru helpful



(d) krok 3 - budowanie zbioru balancing



(e) krok 4 - przeniesienie zbioru balancing

Rysunek 19: Obrazki pokazują początkowy podział, a następnie jedno wywołanie algorytmu Helpful Sets. Rozmiar partycji pozostaje niemal identyczny, zmniejsza się długość granicy.

```

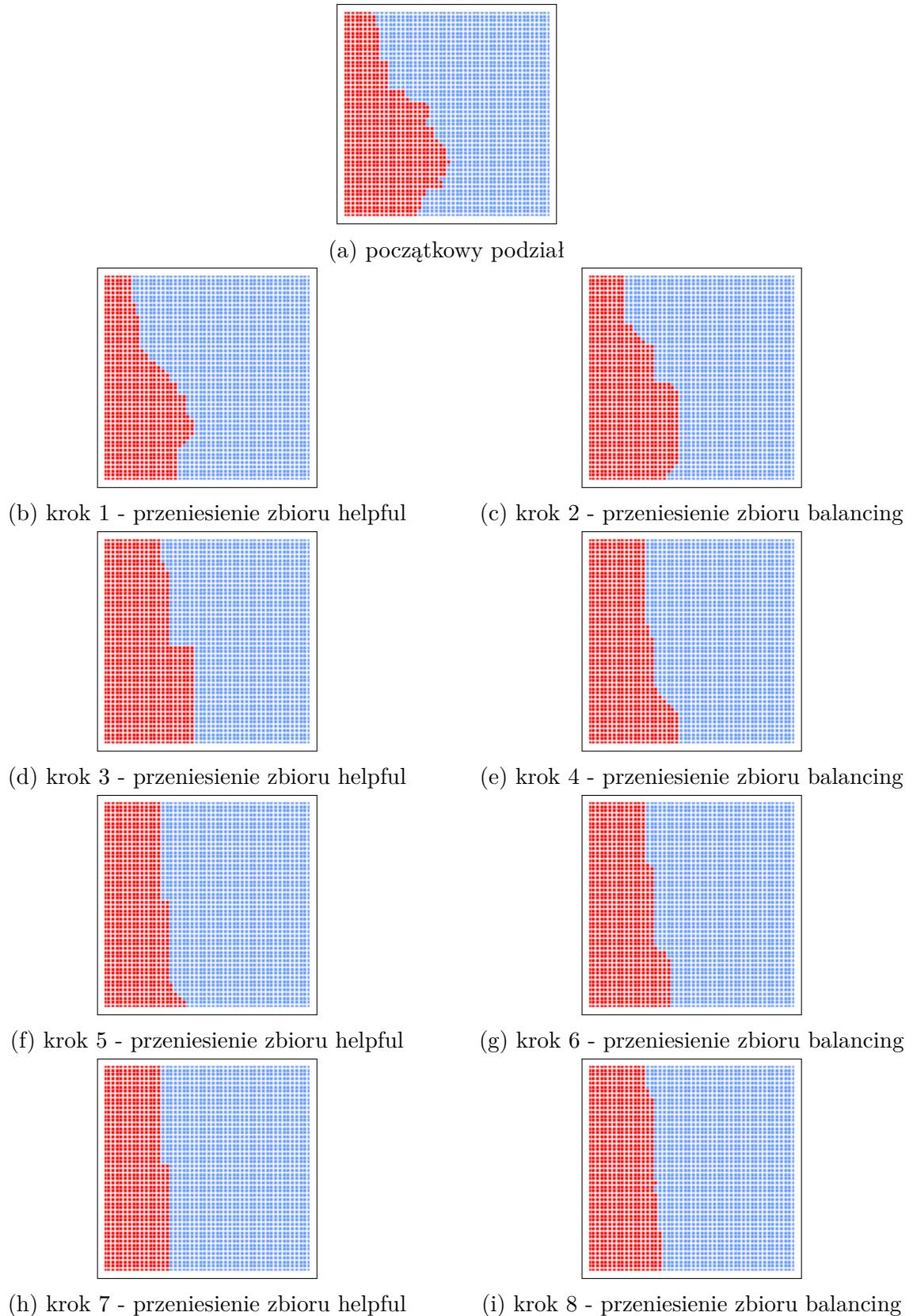
1 HelpfulSet( $A, B$ )
2   IF  $\text{cut\_size}_{A-B} < 0$ 
3     RETURN;
4   ENDIF
5    $l_A \leftarrow l_B \leftarrow \text{cut}/2$ ; /* Initialize the limits */
6    $s_{\max} = (|A| + |B|)/2 \cdot 0.2$ ; /* Initialize max size of HS */
7   WHILE  $l_A + l_B \geq 1$ 
8     IF  $l_A = 0$  OR  $2 \cdot l_A \leq l_B$  /* Choose the more promising partition */
9       Swap( $A, B$ );
10       $S_A = \text{BuildHS}(A, l_A, s_{\max})$ ;
11    ENDIF
12    IF  $h(S_A) \leq l_A$  /* If the helpfulness of  $S_A$  is smaller than wanted ...
13       $l_A \leftarrow b(S_A)$ ; ... adjust the limit for the next search */
14      IF  $l_B > h(S_A)$ 
15         $S_B = \text{BuildHS}(B, l_B, s_{\max})$ ;
16        IF  $h(S_B) \geq h(S_A)$  /* Name the partition with the better set A ...
17          Swap( $A, B$ );
18        ELSE
19           $l_B \leftarrow b(S_B)$ ; ... and reduce the limit of the other partition */
20        ENDIF
21      ENDIF
22      UndoBuild( $S_B$ );
23    ENDIF
24    IF  $\text{len}(S_A) == 0$ 
25       $l_A = 0$ ;
26      CONTINUE;
27    ENDIF
28     $l_A \leftarrow \min(l_A, h(S_A))$ ; /* Adjust the limit for the next search */
29    MoveSet( $S_A$ ) /* Move the helpful set */
30     $\min, \max \leftarrow \text{DetermineMaxAndMin}(w(S_A))$ ;
31     $S_B = \text{BuildBS}(B, 1 - h(S_A), \min, \max)$ ;
32    IF  $w_l \leq w(S_B) \leq w_h$  AND  $h(S_B) > -h(S_A)$  /* Check, if the BS is ok */
33      MoveSet( $S_B$ ); /* Yes: Move the BS */
34       $l_A \leftarrow l_A + \log(l_A)$ ; /* Increase the limits */
35       $l_B \leftarrow l_B + 1$ ;
36    ELSE
37      UndoBuild( $S_B$ ); /* No: Undo the build operation and
38      UndoMove( $S_A$ ); the movement of the helpful set */
39       $l_A \leftarrow l_A/4$ ; /* Reduce the limits */
40       $l_B \leftarrow l_B/2$ ;
41    ENDIF
42  ENDWHILE
43  IF  $\text{cut\_size}_{A-B} > 0.1 \cdot \text{longer\_edge\_of\_a\_grid}$ 
44    Balance( $A, B$ );
45 ENDIF

```

Algorytm 3: Kod przedstawiający zmodyfikowany na potrzeby niniejszej pracy algorytm Helpful-Set 2-partitioning.

Kluczowym elementem algorytmu jest ustalenie wartości *helpfulness*  $l$  zbiorów, których szukamy. Jeśli wartość  $l$  jest zbyt mała, to obiecujące zbiory są przeoczone, natomiast jeśli  $l$  jest za duże, to czas wykonywania algorytmu jest dłuższy, natomiast znalezione przy tym zbiory nie są lepsze. W celu uzyskania  $l$ , Party używa techniki zwanej **adaptive limitation** [6]. Początkowe  $l$  ustawiane jest na wartość  $cut/2$ . Wartość  $l$  jest zmniejszana o połowę albo ustawiana na najlepszą znalezioną wartość *helpfulness*, jeśli zbiór  $l$ -helpful nie może zostać znaleziony w żadnym z dwóch optymalizowanych zbiorów oraz podważana, jeśli poszukiwanie zakończy się sukcesem. Autorzy [24] zaproponowali kilka modyfikacji do algorytmu. Po pierwsze, zamiast używać pojedynczego limitu  $l$  użyto dwóch oddzielnych limitów dla zbioru  $A$  oraz zbioru  $B$ . Po drugie, dopuszczone do pojawiania się małych różnic wielkości obszarów. Jest to szczególnie ważne, kiedy podział optymalizowany jest wielokrotnie, na niecałkowicie przywróconym do początkowego rozmiaru grafie. Przenoszone wówczas wierzchołki mają często wagę  $\geq 1$ , ponadto istnieją różnice pomiędzy wagami wierzchołków, co przekłada się na trudności w idealnym zbalansowaniu optymalizowanych obszarów. To podejście zostało również zaimplementowane w bibliotekach Jostle and Metis. Udowodniono, że wpływa ono pozytywnie na optymalizację długości granic, ponieważ dopuszczenie niewielkich różnic w powierzchni pól pozwala zwykle na lepsze zoptymalizowanie długości granic. Po trzecie algorytm balansowania obszarów, który zachłannie wyrównuje pola obszarów został przeniesiony na koniec.

Algorytm 3 przedstawia zmodyfikowaną przez mnie procedurę. Pierwszą moją modyfikacją jest sprawdzanie, czy balansowane obszary wciąż ze sobą graniczą (linia 2). Może się zdarzyć, że na skutek balansowania innych obszarów granica ta zostanie przerwana. Jak w poprzedniej implementacji, limity  $l_A$  oraz  $l_B$  ustawiane są na połowę aktualnej długości granicy pomiędzy obszarami  $A$  i  $B$  (linia 5). W rozwiązaniu zaproponowanym przez autorów [24], podczas wyszukiwania, wartość *helpful* zbioru *helpful* nie może stać się mniejsza niż  $-d/2$ , nie może również przekroczyć wagi  $s_{max}$ . Wartość  $d$  to średni stopień wierzchołka w grafie. Dla mojej implementacji mógłbym założyć, że  $d$  jest zawsze równe 4 - wynika to ze sposobu, w jaki buduję graf z wejściowej siatki (rysunek 10). Jednak ta zasada nie działa. Algorytm budowania zbioru *helpful* jest bowiem zachłanny i często budował zbiory, które spełniały kryterium nieprzekraczania wagi  $s_{max}$  oraz miały ujemną wartość *helpful*, większą niż  $-2$ . W kodzie oryginalnego algorytmu później sprawdzane było, czy zbiór nie ma ujemnej wartości *helpful* i jeśli miał to budowanie zaczynało się od początku. Przez to algorytm wywoływał się wielokrotnie bez zakończenia się sukcesem. Moim rozwiązaniem była zmiana tej instrukcji na warunek na wielkość zbioru  $S_A$  (linia 24) oraz na pozwolenie na budowanie zbiorów o wartości *helpful* większej bądź równej 0 (linia 10 oraz 15). Natomiast  $s_{max}$ , autorzy [6], ustawiali na wartość 128. Ja uzależniłem  $s_{max}$  od rozmiaru optymalizowanych partycji (linia 6). Użyty do tego współczynnik 0.2 może zostać dowolnie zmieniony. Jednak doświadczenie wskazuje, że powinien być on raczej mniejszy niż 0.4, aby zbiór *helpful* nie był zbyt duży. Zwiększa to czas partycjonowania, ale nie poprawia wyników. Korzystniejsze jest, gdy jest on nieco za mały, niż za duży. Po udanych znalezieniu małego zbioru *helpful* oraz zbioru *balancing*, limit i tak zostanie zwiększony, a zaczynanie od bardzo dużego zbioru *helpful* często kończy się nieudanym poszukiwaniem zbioru *balancing* i powrotem do początkowego podziału. Poszukiwanie dużego zbioru jest bardziej kosztowne.



Rysunek 20: Obrazki przedstawiają kolejne kroki działania algorytmu Helpful Sets na siatce 50x50 podzielonej na 2 obszary przez algorytm LAM. W pierwszym kroku tworzony jest stosunkowo mały zbiór helpful, a w kolejnym kroku odpowiednio mały zbiór balancing. W momencie kiedy to się udaje algorytm zwiększa limit liczby wierzchołków, dlatego kolejne kroki wnoszą więcej zmian. Po 4 cyklach budowania i przenoszenia zbioru helpful oraz zbioru balancing granica między obszarami jest znacznie mniejsza, a wielkość obszarów zachowana. Na potrzeby przykładu balansowanie pól jest wyłączone.

Pętla WHILE, która znajduje się między liniami 7 i 42 jest wykonywana, dopóki suma limitów jest większa od 1. W oryginalnej implementacji wartość ta wynosiła 0, jednak wtedy algorytm wywoływał się bardzo długo, niekoniecznie poprawiając jakość partycjonowania. Jeśli limit  $l_A$  jest dużo mniejszy od limitu  $l_B$  zbiory  $A$  i  $B$  są ze sobą zamieniane (linia 9). Ten warunek powoduje, że zbiór helpful szukany jest najpierw w bardziej obiecującej partycji. Jeśli zbiór  $l_A$ -helpful ( $S_A$ ) (z wagą  $w(S_A)$  oraz wartością helpfulness wynoszącą  $h(S_A)$ ) nie może zostać znaleziony (linia 12), limit jest redukowany do najlepszej znalezionej wartości helpfulness ( $b(S_A)$ ). Dalej podejmowana jest decyzja czy nastąpi wyszukiwanie w partycji  $B$  (linia 14). W jej wyniku ustalony może zostać zbiór  $S_B$  (linia 15). Dalej zbiór z większą wartością helpfulness jest nazywany  $S_A$  (linia 17) lub limit drugiej partycji jest redukowany (linia 19), a zbiór  $S_B$  usuwany (linia 22). Jeśli wielkość zbioru helpful jest równa 0, to limit ustawiany jest na 0 (linia 25), a algorytm powtarzany. W przeciwnym wypadku do  $l_A$  przypisywana jest nowa wartość (linia 28),  $S_A$  przenoszone jest do  $B$  (linia 29), a wywołanie wkracza w fazę szukania zbioru balancing. W tym celu obliczany jest przedział  $[min, max]$  dla zbioru balancing (linia 30). W implementacji zaproponowanej w [24] wartości  $min$  oraz  $max$  były obliczane w następujący sposób:

```

1  $min \leftarrow (w(B) - w_{max}(B) - grace)^+$ 
2  $max \leftarrow (w(B) - w_{min}(B) + grace)^+$ 
```

#### Algorytm 4

gdzie  $grace$  jest połową wagi najczępszego wierzchołka. Jednak ten sposób wyznaczania wartości  $min$  oraz  $max$  nie sprawdzał się w moich testach.  $max$  i  $min$  pozwalają na budowanie zbioru balancing o przedziale wag  $[min, max]$ . Rzadko jednak algorytm nie był w stanie osiągnąć wartości  $max$ . Najczęściej była to wartość bliska lub równa  $max$ . Ponadto na początku algorytmu znajduje się kod odpowiedzialny za wybieranie lepszego zbioru do budowania zbioru helpful. Testy pokazały, że podczas wielokrotnych wywołań, pomimo ciągłych wymian wierzchołków między obszarami, zwykle to ten sam obszar jest wybierany jako ten lepszy do budowania zbioru helpful. W efekcie jeśli zbiór balancing był za każdym razem bliski  $max$  (czyli był nieco większy od zbioru helpful) oraz był on budowany na tym samym obszarze, to jeden obszar sukcesywnie rósł kosztem drugiego. W związku z tym zaproponowałem poniższe rozwiązańe, które losowo decyduje, czy wartość  $max$  ma być trochę mniejsza, czy trochę większa od zbioru helpful:

```

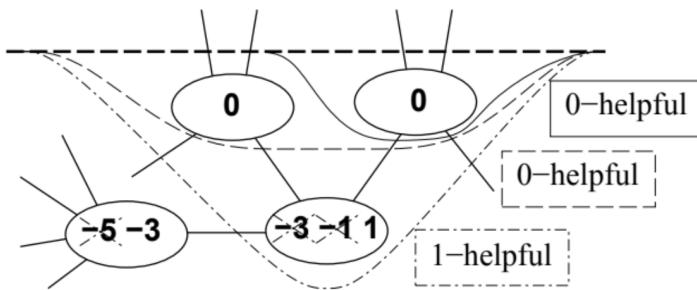
1 DetermineMaxAndMin( $w(S_A)$ )
2    $rand = RandValue(0,1)$     /* draws either 0 or 1 */
3   IF  $rand == 1$ 
4      $min = |w(S_A) - 0.1 \cdot w(S_A)|$ 
5      $max = |w(S_A) + 0.1 \cdot w(S_A)|$ 
6   ELSE
7      $min = |w(S_A) - 0.2 \cdot w(S_A)|$ 
8      $max = |w(S_A) - 0.1 \cdot w(S_A)|$ 
9   ENDIF
10  RETURN  $min, max$ 
```

#### Algorytm 5

Za pomocą tych granic algorytm wyszukuje zbiór balancing  $S_B$ , który nie zwiększy długości granicy pomiędzy  $A$  i  $B$  o więcej niż  $1 - h(S_A)$  (linia 31). Jeśli szukanie takiego zbioru zakończy się sukcesem, to jest on przenoszony do zbioru  $A$  i limity dla obydwu zbiorów ( $l_A$  oraz  $l_B$ ) są zwiększane (linie 33-35). W przeciwnym wypadku  $S_B$  jest usuwany,  $S_A$  przenoszone z powrotem do  $A$ , a limity  $l_A$  oraz  $l_B$  zmniejszane (linie 37-40). Jeśli algorytm nie może już poprawiać więcej partycjonowania sprawdzany i w razie potrzeby poprawiany jest balans pól (linia 43). Dodatkową modyfikacją wprowadzoną przeze mnie był warunek na uruchomienie procedury 'Balance', który jest omawiany dokładniej w dalszych częściach pracy. Jego celem jest uniemożliwienie balansowania pól między obszarami o bardzo krótkiej granicy.

### 3.1.6. Szczegóły budowania zbioru helpful

Główna funkcja tej heurystyki działa podobnie do algorytmu Kruskala do wyliczania najmniejszego drzewa rozpinającego [31]. Startuje od pustego zbioru (o wartości helpfulness równej 0) i buduje go za pomocą wierzchołków z jednej strony granicy. W każdym kroku wybiera wierzchołek z wartością helpfulness z pewnego przedziału, przenosi go do zbioru (helpfulness zbioru zwiększa się wtedy o helpfulness wierzchołka), następnie zwiększa wartość helpfulness wierzchołków, które są w tej samej partycji i są sąsiednie do przeniesionego wierzchołka. Te wierzchołki zyskują jedną krawędź zewnętrzną kosztem jednej krawędzi wewnętrznej, w związku z powyższym ich wartość helpfulness rośnie o 2.

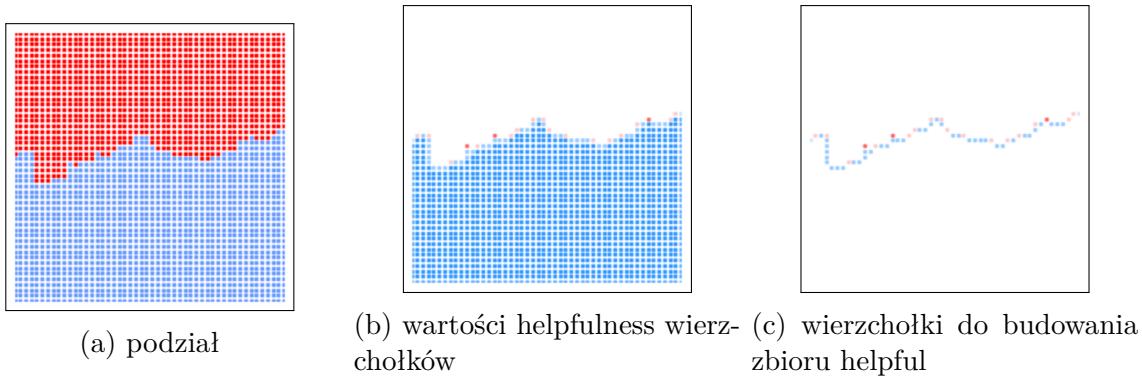


Rysunek 21: Zmiana wartości helpfulness dla sąsiadów. Źródło: [6].

Branie pod uwagę tylko wierzchołków z jednej strony granicy pozwala na budowanie zbioru z dwóch stron podziału jednocześnie. Jeśli znaleziony zbiór wierzchołków jest przemieszczany między partycjami, to wartości helpfulness dla wierzchołków docelowej partycji muszą również zostać zaktualizowane. Podczas budowania zbioru jego wielkość zwiększa się w każdym kroku. Zbiór helpful przestaje być budowany w następujących przypadkach:

- wartość helpfulness dla zbioru osiągnie wartość limitu,
- brane są pod uwagę tylko wierzchołki o pewnej wartości helpfulness, a te się skończą,
- wielkość zbioru osiągnie wartość limitu.

Wałą obserwacją jest fakt, że jeśli brane pod uwagę są jedynie wierzchołki z wartością helpfulness  $\geq 0$ , to helpfulness zbioru albo zachowuje tę samą wartość, albo rośnie, ale nigdy nie maleje. Podobieństwo tego algorytmu do algorytmu BFS objawia się w sposobie, w jakim wierzchołki przypisywane są do zbioru. Wierzchołek może zostać wybrany, jeśli jego wartość helpfulness jest większa od ustalonej wartości. To następuje albo jeśli wierzchołek ma od początku odpowiednią wartość helpfulness, albo jeśli urosnie ona do odpowiedniej wartości podczas budowania zbioru (poprzez dodanie do zbioru jego sąsiadów).



Rysunek 22: Obrazki przedstawiają jak wygląda zbiór wierzchołków, na bazie którego budowany jest zbiór helpful. Obrazek (a) przedstawia partycjonowanie. Zbiór helpful będzie budowany na niebieskiej partycji. Obrazek (b) przedstawia wszystkie wierzchołki zbioru (a), gdzie kolor oznacza wartość helpful. Skala rozpoczyna się od koloru ciemnoniebieskiego dla wierzchołków z wartością helpfulness wynoszącą  $-4$  i przechodzi do czerwonego dla wierzchołków z wartością helpfulness  $4$ . Widoczna jest większa wartość helpfulness dla wierzchołków przy granicy. Obrazek (c) przedstawia wierzchołki, które brane są pod uwagę przez algorytm. Są one filtrowane na etapie budowania zbioru, tak by zawsze wybierane były wierzchołki znajdujące się na granicy.

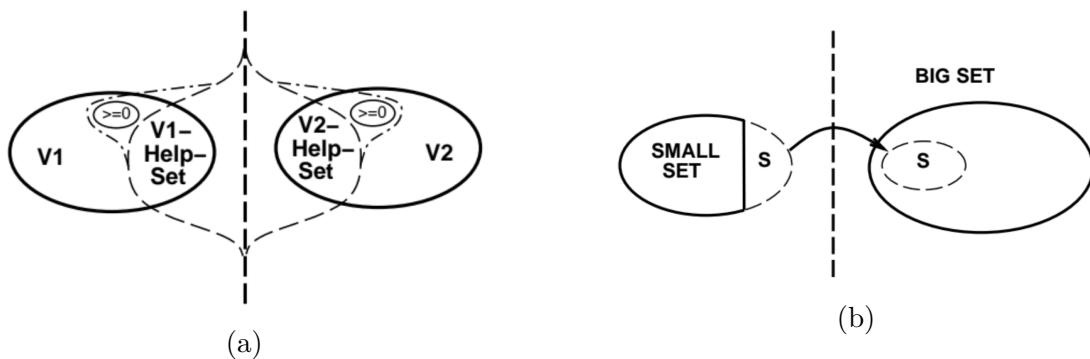
Dodatkową modyfikacją, którą wprowadziłem, było filtrowanie wierzchołków, tak by algorytm wybierał wierzchołek z największą wartością helpfulness tylko spośród wierzchołków granicznych. Została ona zaimplementowana w związku z powstającymi obszarami rozproszonymi (to negatywne zjawisko opisuje w późniejszych częściach pracy). Chciałem aby istniała pewność, że wybierany wierzchołek jest zawsze wierzchołkiem granicznym. Rysunek 22 przedstawia moje rozwiązanie.

### 3.1.7. Szukanie zbioru $k$ -helpful gdzie $k \geq \text{limit}$

Proces budowania zbiorów helpful pokazany jest na rysunku 23(a). Te zbiory nazywane są zbiorami Help. Kardynalność tych zbiorów rośnie dynamicznie podczas szukania. Dzięki temu, że podczas budowania zbioru Help aktualizujemy tylko wartości helpfulness wierzchołków z tej samej partycji, obydwa zbiory Help mogą być budowane niezależnie i równolegle. Jednak w mojej implementacji (rysunek 3) budowane są one jeden po drugim (jeśli warunek pozwoli na zbudowanie drugiego).

Podczas tego procesu brane pod uwagę są tylko wierzchołki z wartością helpfulness  $\geq 0$ . Może to prowadzić do przedwczesnego zakończenia wywołania algorytmu, jeśli żadne wierzchołki z taką wartością helpfulness nie będą już dostępne. To także gwarantuje, że wartość helpfulness dla zbioru Help nigdy nie będzie maleć. Wyszukiwanie zbioru Help kończy się, gdy jego wartość helpfulness osiąga limit lub jeśli nie ma więcej wierzchołków z wartością helpfulness  $\geq 0$ .

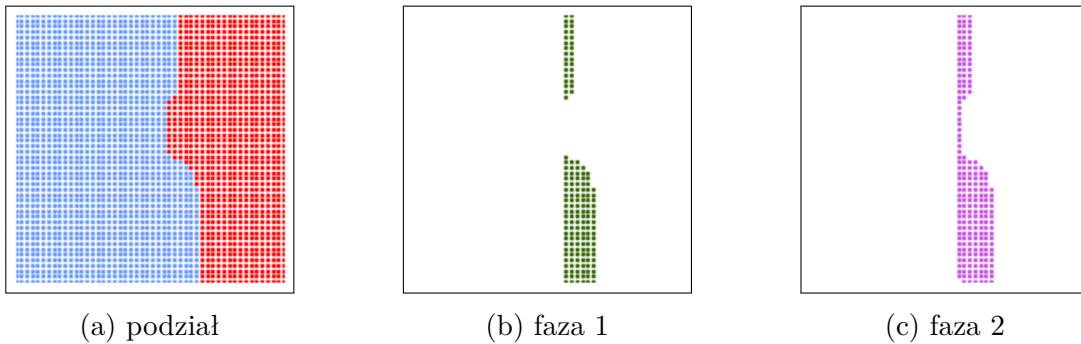
Po udanym wywołaniu, zbiór  $S$ , który jest zbiorem Help z większą wartością helpfulness przenoszony jest na drugą stronę podziału. Drugi zbiór Help jest usuwany i nie jest dalej brany pod uwagę. Na rysunku 23(b) BIG SET reprezentuje  $V_1$  lub  $V_2$  połączone z  $S$ . SMALL SET to pozostała część, która została zredukowana o  $S$ .



Rysunek 23: Szukanie zbioru  $k$ -helpful poprzez przenoszenie wierzchołków z wartością helpfulness  $\geq 0$  do zbiorów Help (a). (b) to przenoszenie zbioru  $S$  na drugą stronę podziału. Źródło: [6].

### 3.1.8. Szczegóły budowania zbioru balancing

Znalezienie zbioru balancing jest trudniejsze, z racji na dodatkowe obostrzenia związane z jego rozmiarem oraz wartością helpfulness. Zbiór  $\bar{S}$  musi mieć ten sam rozmiar co zbiór  $S$  oraz jego wartość helpfulness musi być tak duża jak to możliwe, lecz nie mniejsza niż  $-k+1$ -helpful, jeśli  $S$  był  $k$ -helpful. Idea budowania zbioru balancing polega na rozpoczęciu od pustego zbioru  $\bar{S}$  i wybraniu podzbioru ze zbioru BIG SET, który zwiększy jego rozmiar tak bardzo jak to możliwe oraz jednocześnie zmniejszy jego wartość helpfulness tak mało jak to możliwe. Algorytm podzielony jest na trzy fazy. Pierwsza faza dodaje do zbioru wierzchołki, których wartości helpfulness są  $\geq 0$ . Druga faza próbuje znaleźć podzbiory wierzchołków, które dodane do  $\bar{S}$ , pozostawią jego helpfulness na jednym poziomie - szuka zbiorów 0-helpful. Każda z tych faz kończy swoje wywołanie jeśli  $\bar{S}$  jest wystarczająco duży. Jeśli dwie pierwsze fazy są niewystarczające, uruchamiana jest trzecia faza, która oparta jest na zachłannym dobieraniu wierzchołków, w celu dokończenia budowania zbioru balancing.



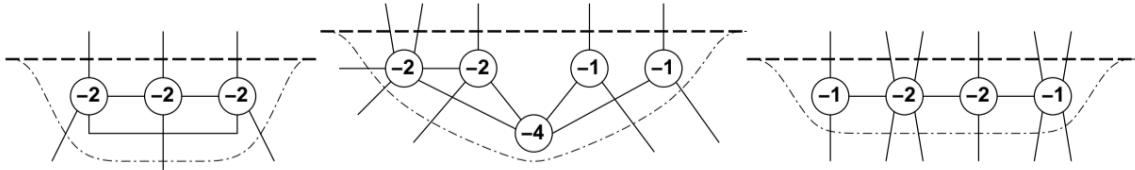
Rysunek 24: Obrazki przedstawiają fazę pierwszą oraz fazę drugą budowania zbioru balancing. Ten sam przypadek podziału można znaleźć na rysunku 19.

#### Faza 1

Pierwsza faza jest najmniej skomplikowana. Dopóki w zbiorze BIG SET są wierzchołki z wartościami helpfulness  $\geq 0$ , wierzchołek z największą wartością helpfulness dodawany jest do zbioru  $\bar{S}$ . Te wierzchołki uznawane są jako wierzchołki, które należą już do drugiej partycji, więc wartości helpfulness ich sąsiadów z poprzedniej partycji muszą wzrosnąć o 2. Podczas tej fazy wielkość  $\bar{S}$  wzrasta, a wartość  $H(\bar{S})$  pozostaje taka sama lub się powiększa. Faza pierwsza kończy się jeśli utworzony zostanie zakładany zbiór balancing lub nie ma więcej wierzchołków w zbiorze BIG SET, których wartości helpfulness są przynajmniej równe 0.

#### Faza 2

Ta faza próbuje znaleźć zbiory 0-helpful, to znaczy podzbiory zbioru BIG SET, które przeniesione do  $\bar{S}$ , nie zmniejszą jego wartości helpfulness. Koncepcja tej fazy polega na przeprowadzeniu kilku wyszukiwań, rozpoczynając od pojedynczego wierzchołka  $-1$ -helpful lub  $-2$ -helpful. Każde z wyszukiwań próbuje dopełnić swój podzbiór wierzchołków do zbioru 0-helpful. Ważną obserwacją jest, że na początku tego etapu BIG SET nie zawiera żadnych wierzchołków z wartością helpfulness większą bądź równą zero.



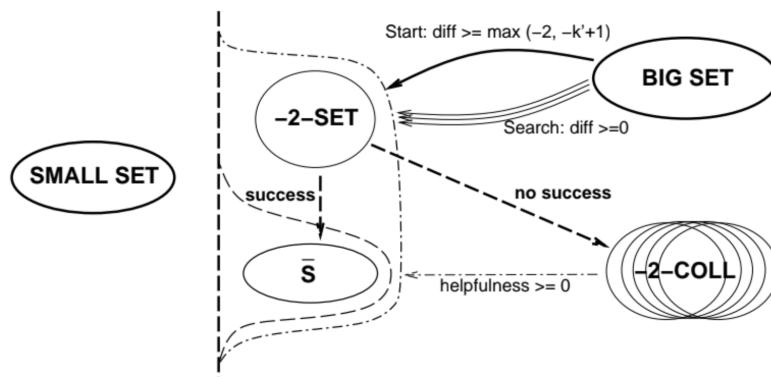
Rysunek 25: Przykłady zbiorów 0-helpful. Źródło: [6].

Rysunek 25 pokazuje przykładowe zbiory 0-helpful. Nie ma jednego przepisu na zbiór 0-helpful, mogą być to różne konstrukcje wierzchołków, a intencją drugiej fazy jest znalezienie jak największej liczby tego typu zbiorów.

Rysunek 26 pokazuje drugą fazę. Tak jak w fazie pierwszej, wykorzystywana jest ta sama metoda budowania zbioru (sekcja 3.1.6). Jednak zamiast przenosić wierzchołki bezpośrednio do  $\bar{S}$ , są one przechowywane w zbiorze o nazwie  $-2\text{-SET}$ .  $-2\text{-SET}$  zawiera aktualny podzbiór wierzchołków, który faza druga próbuje dopełnić do zbioru 0-helpful. Niech  $k' = k + H(\bar{S})$  będzie poprawą długości granicy, spowodowaną przez przeniesienie  $S$  i  $\bar{S}$ , każdy na odpowiednią stronę podziału. Na początku każdego poszukiwania zbioru  $-2\text{-SET}$  rozpatrywane są jedynie wierzchołki z wartością helpfulness, wynoszącą  $\max(-2, -k' + 1)$ . Następnie tylko wierzchołki, które są przynajmniej 0-helpful mogą być umieszczone w zbiorze  $-2\text{-SET}$ . To gwarantuje, że dowolny  $-2\text{-SET}$  może zostać przeniesiony do  $\bar{S}$  na każdym etapie algorytmu, bez znaczącego zmniejszania jego wartości helpfulness. Jest to szczególnie istotne jeśli  $-2\text{-SET}$  jest wystarczająco duży, aby zakończyć balansowanie. Podsumowując:

- tylko wierzchołki  $-1\text{-helpful}$  oraz  $-2\text{-helpful}$  ze zbioru BIG SET są rozważane na początku budowania  $-2\text{-SET}$ ,
- wartość helpfulness zbioru  $-2\text{-SET}$  jest co najmniej równa  $-2$  i nigdy nie maleje,
- jeśli  $k'$  wynosi 2 tylko wierzchołki  $-1\text{-helpful}$  mogą zostać wybrane oraz
- jeśli  $k'$  wynosi 1 faza druga w ogóle się nie rozpoczyna.

Jeśli wyszukiwanie zbioru  $-2\text{-SET}$  zakończy się i jego wartość helpfulness jest mniejsza od 0 to jest przechowywany do dalszego użycia.



Rysunek 26: Obrazek przedstawiający fazę 2. Źródło: [6].

```

1 PROCEDURE Phase_2
2   WHILE ( $|\bar{S}| < |S|$ ) AND BIG SET contains nodes with helpfulness  $\geq$ 
3     max( $-2, -k + 1 - H(\bar{S})$ )
4     start building a -2-SET with the max. helpful node;
5
6   WHILE  $H(-2\text{-SET}) < 0$  AND ( $|-2\text{-SET}| + |\bar{S}| < |S|$ ) AND
7     BIG SET has node with helpfulness  $\geq 0$ 
8     move node with the highest helpfulness to -2-SET;
9   ENDWHILE
10
11  IF  $H(-2\text{-SET}) \geq 0$ 
12    move -2-SET to  $\bar{S}$ ;
13    WHILE ( $|\bar{S}| < |S|$ ) AND -2-COLL contains sets  $S'$  with  $H(S') \geq 0$ 
14      take set  $S' \in -2\text{-COLL}$  with max. helpfulness;
15      IF  $|S'| + |\bar{S}| < |S|$ 
16        move  $S'$  to  $\bar{S}$ ;
17      ELSE
18        reduce  $S'$  to  $|S| - |\bar{S}|$  and move it to  $|\bar{S}|$ ;
19      ENDIF
20    ELSE IF  $|-2\text{-SET}| + |\bar{S}| = |S|$ 
21      move -2-SET to  $\bar{S}$ ;
22    ELSE
23      move -2-SET to -2-COLL;
24    ENDIF
ENDWHILE

```

Algorytm 6: Algorytm przedstawiający fazę drugą. Źródło: [6].

Kolekcja zbiorów -2-SET jest nazywana -2-COLL. Zawiera ona podzbiory wierzchołków, które są co najmniej -2-helpful. Wiele z nich może zostać zbiorami 0-helpful, jeśli inne zbiory -2-SET zostaną przeniesione do  $\bar{S}$ . Aktualizacja wartości helpfulness sąsiadów podczas budowania zbioru -2-SET następuje tylko dla wierzchołków, które są częścią zbioru BIG SET. Zbiory -2-SET będące w zbiorze -2-COLL nie są uznawane za zbiory, które zmieniły stronę partycjonowania. W związku z tym, jeśli -2-SET nie może zostać uzupełniony wierzchołkami, aby być 0-helpful i jest przenoszony do zbioru -2-COLL, to wartości helpfulness wierzchołków w zbiorze BIG SET, które były jego sąsiadami, są przywracane do poprzednich wartości. Na tym etapie algorytmu podczas budowania zbioru -2-SET, wartości helpfulness sąsiadnych wierzchołków w zbiorze -2-COLL nie są zmieniane, ponieważ wierzchołki w zbiorze -2-COLL nie są brane pod uwagę przy budowaniu zbiorów -2-SET. Tylko jeśli zbiór -2-SET zostanie 0-helpful i jest przenoszony do  $\bar{S}$  to aktualizowane są wartości helpfulness jego sąsiadów będących w zbiorze -2-COLL.

Algorytm 6 przedstawia fazę drugą. Szukanie zbioru -2-SET kończy się w następujących przypadkach:

1. Wartość helpfulness zbioru -2-SET staje się większa bądź równa 0. W tym przypadku cały zbiór -2-SET przenoszony jest do zbioru  $\bar{S}$ . Wartości helpfulness sąsiadnych wierzchołków znajdujących się w zbiorze -2-COLL są aktualizowane. Każda taka aktualizacja wartości helpfulness dla wierzchołka w zbiorze -2-COLL skutkuje

tym, że wartość helpfulness jednego znajdującego się tam zbioru  $-2\text{-SET}$  podnosi się o 2. W związku z tym wiele z nich może zostać co najmniej 0-helpful, a następnie zostać przeniesione do  $\bar{S}$ . Algorytm wybiera zawsze zbiór  $-2\text{-SET}$  z największą wartością helpfulness i przenosi go do  $\bar{S}$ , tak długo jak w  $-2\text{-COLL}$  są takie zbiory.

Jeśli, któryś z tych zbiorów jest większy niż  $|S| - |\bar{S}|$  to jest redukowany do odpowiedniego rozmiaru, poprzez usuwanie ostatnio dodanych elementów. Rezultatem takiej redukcji jest  $|\bar{S}| = |S|$  oraz zakończenie wykonania procedury.

2. Partycja osiąga odpowiedni rozmiar, to znaczy  $|-2\text{-SET}| + |\bar{S}| = |S|$ . W tym przypadku wartość helpfulness zbioru  $-2\text{-SET}$  wciąż wynosi co najmniej  $\max(-2, -k' + 1)$  (co jest zagwarantowane procesem jego budowania) i może on zostać przeniesiony do zbioru  $\bar{S}$ . Partycja osiąga w tym wypadku odpowiedni rozmiar i następuje zakończenie procedury.
3. Nie ma więcej wierzchołków w zbiorze BIG SET z wartością helpfulness większą bądź równą 0. W tym wypadku aktualnie budowany  $-2\text{-SET}$  nie może zostać powiększony o kolejne wierzchołki, w związku z tym jest przenoszony do zbioru  $-2\text{-COLL}$ , gdzie oczekuje na późniejsze wykorzystanie.

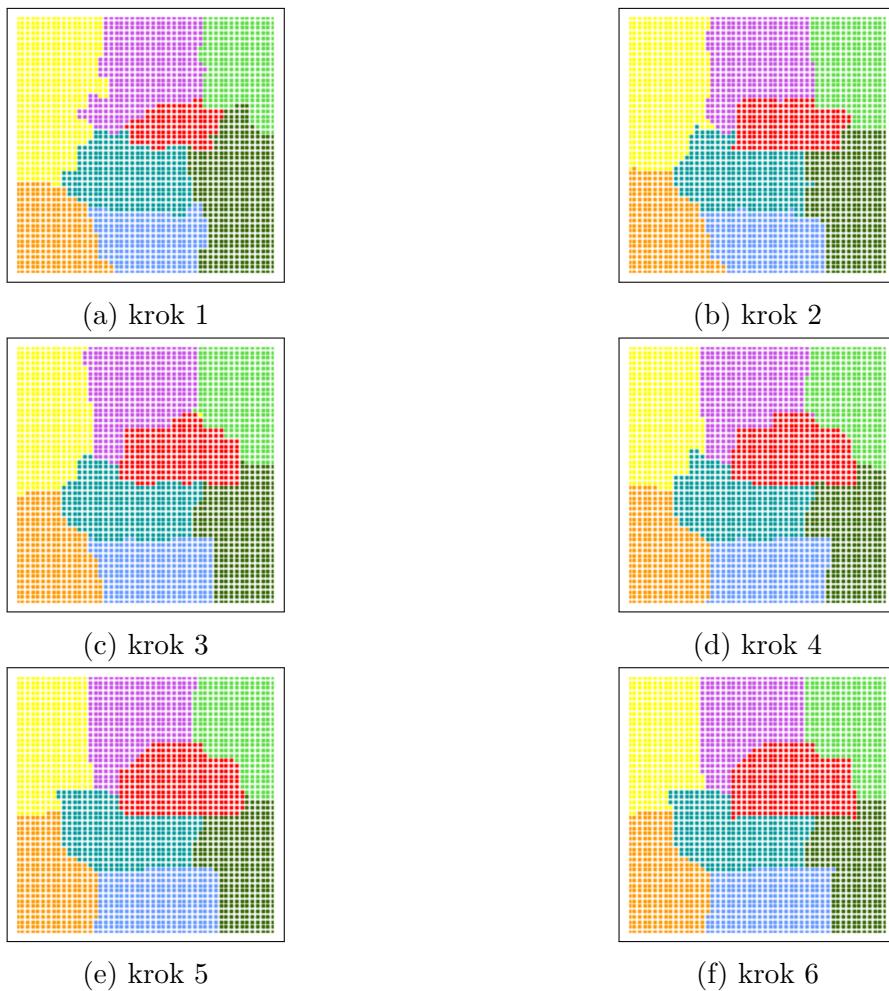
### Faza 3

Ta faza wybiera wierzchołki ze zbioru BIG SET oraz zbiory  $-2\text{-SET}$  ze zbioru  $-2\text{-COLL}$  w celu przeniesienia do  $\bar{S}$ . Wybiera wierzchołek lub zbiór z największą wartością helpfulness tak długo, aż przeniesienie go nie obniży wartości helpfulness zbioru  $\bar{S}$  poniżej wartości  $-k + 1$ . Jeśli jakiś zbiór  $-2\text{-SET}$  ze zbioru  $-2\text{-COLL}$  jest użyteczny, ale jego rozmiar jest większy niż  $|S| - |\bar{S}|$ , to jego rozmiar jest redukowany. Faza 3 kończy swoje wywołanie, jeśli przeniesienie jakiegokolwiek zbioru  $-2\text{-SET}$  lub wierzchołka, spowoduje obniżenie wartości helpfulness zbioru  $\bar{S}$  poniżej  $-k + 1$  lub jeśli uda się wypełnić zbiór  $\bar{S}$  do rozmiaru zbioru  $S$ .

### 3.1.9. Balansowanie rozmiarów obszarów

Ten etap algorytmu wywoływany jest zawsze po procedurze optymalizacji długości granic. Odpowiedzialny jest za wyrównanie wielkości pól, które nie są idealnie równe po wywołaniu algorytmu LAM. Przez autorów artykułu [24] został określony jako algorytm zachłanny, lecz nie opisano szczegółów jego działania.

W związku z powyższym zaimplementowałem następujące rozwiązanie: algorytm korzysta również z mechanizmu do budowania zbiorów helpful i balancing, to jest z obliczania wartości helpfulness dla wierzchołków. Zachłannie wybierane są wierzchołki z największą wartością helpfulness, dopóki pola obszarów nie zostaną zbalansowane do oczekiwanej wartości różnicy wielkości pól. Obszary balansują się zabierając wierzchołki obszarów sąsiednich. Oznacza to, że jeśli wywoływany jest algorytm optymalizacji granic między obszarami  $A$  oraz  $B$ , gdzie obszar  $A$  jest większy od obszaru  $B$ , to po wyrównaniu wielkości pól obszar  $A$  odda część wierzchołków obszarowi  $B$ .



Rysunek 27: Obrazki pokazują działania algorytmu optymalizacji granic oraz balansowania rozmiarów pól. Jak widać, z wywołania na wywołanie, czerwony obszar, który był na początku najmniejszy, stopniowo rośnie odbierając wierzchołki sąsiednim obszarom.  $p$  wynosi 0.1.

Dla działania algorytmu ustalany jest współczynnik  $p$ , który decyduje o tym ile wierzchołków zostanie wymienionych.  $p$  osiąga wartości od 0 do 0.5 i oznacza jaka część różnicę pól balansowanych obszarów będzie przeniesiona do mniejszego obszaru.  $V_b$  to wierzchołki wytypowane przez algorytm balansowania pól, które zostaną przeniesione do mniejszego obszaru.

$$|V_b| = p \cdot |A| - |B| \quad (5)$$

Im  $p$  będzie mniejsze, tym małe obszary będą rosły wolniej, stopniowo rozbudowując się w kierunku wszystkich sąsiadów. Im  $p$  będzie większe tym proces będzie bardziej gwałtowny, co może bardziej naruszyć podział, na przykład przemieszczając obszar w jakimś kierunku. Im  $p$  mniejsze, tym więcej wywołań algorytmu optymalizacji jest potrzebne, aby zrównać wielkości pól obszarów. Zgodnie z moimi obserwacjami, dla współczynników decydujących o częstotliwości wywołań algorytmu do optymalizacji granic (algorytm 2), dobrze sprawdza się  $p$  wynoszące około 0.1. Widoczne jest to również na rysunku 27, gdzie pokazane są wszystkie faktyczne fazy balansowania. Ze współczynnikiem 0.1 algorytm miał wystarczająco dużo czasu, aby wyrównać powierzchnie pól.

Dodatkową modyfikacją wprowadzoną przeze mnie, był warunek uruchamiania tej części algorytmu, która nie jest konieczna, ale odgrywa ważną rolę, gdy pojawiają się obszary niepodzielne. Czasami zdarza się, że balansowane obszary ze względu na szczególne ułożenie obszarów niepodzielnych na siatce, mają ze sobą bardzo krótką granicę. Taka sama sytuacja może pojawić się, gdy rozpatrywany jest klasyczny przypadek bez obszarów niepodzielnych, choć jest ona wtedy znacznie rzadziej spotykana. W takim przypadku, dodatkowo jeśli  $p$  oraz różnica pól balansowanych obszarów jest stosunkowo duża, może dojść do niekorzystnego balansowania, w którym obszar mniejszy zwiększa swoje pole o dużą liczbę wierzchołków, poprzez stosunkowo krótką granicę. Aby nie dopuścić do takiego przypadku, stosowany jest prosty warunek, który uniemożliwia balansowania między obszarami, jeśli długość ich granicy nie przekracza 0.1 dлиności dłuższego boku siatki.

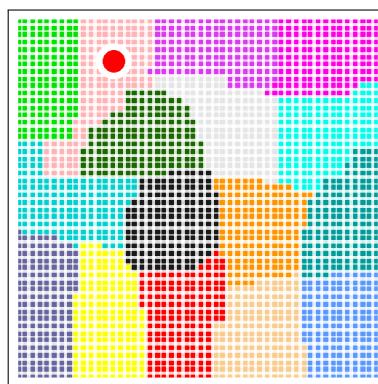
### 3.1.10. Usuwanie obszarów rozproszonych

Występowanie obszarów rozproszonych w kontekście partycjonowania oznacza, że na podzielonej siatce ten sam obszar występuje w dwóch lub większej liczbie oddzielnych części. Problem ten wystąpił, pomimo że autorzy biblioteki Party nie wspominali o takiej możliwości. Przykłady obszarów rozproszonych przedstawione są na rysunku 28.



Rysunek 28: Na rysunku (a) widać, że niebieski obszar jest w dwóch miejscach - na dole oraz w postaci bardzo małego obszaru na środku. Ten sam efekt występuje dla obszaru zielonego na obrazku (b).

Problem spowodowany jest etapem optymalizacji podziału. Dzieje się tak, ponieważ obszary wymieniają się ze sobą granicznymi wierzchołkami w celu wyrównania granic. Przykładem podziału, na którym taki problem może się pojawić jest rysunek 29. W lewym rogu widać obszar zaznaczony czerwonym kółkiem, znajdujący się pomiędzy dwoma zielonymi obszarami. Wąska, dolna odnoga tego obszaru jest potencjalnym miejscem, gdzie może nastąpić odcięcie i przedzielenie tego obszaru na dwie części.



Rysunek 29: Na rysunku widać obraz siatki podatnej na pojawienie się obszarów rozproszonych. Czerwoną kropką zaznaczony jest obszar podatny na podzielenie na dwie części.

Algorytm optymalizacji granic jest szczególnie narażony na tego typu zachowania w swoich początkowych fazach, kiedy graf jest zredukowany i pod pojedynczymi wierzchołkami przenoszone są tak naprawdę całe ich zbiory. Wtedy często wystarczy przenieść jeden wierzchołek, aby przedzielić inny obszar na dwie części. Dlatego też algorytm optyma-

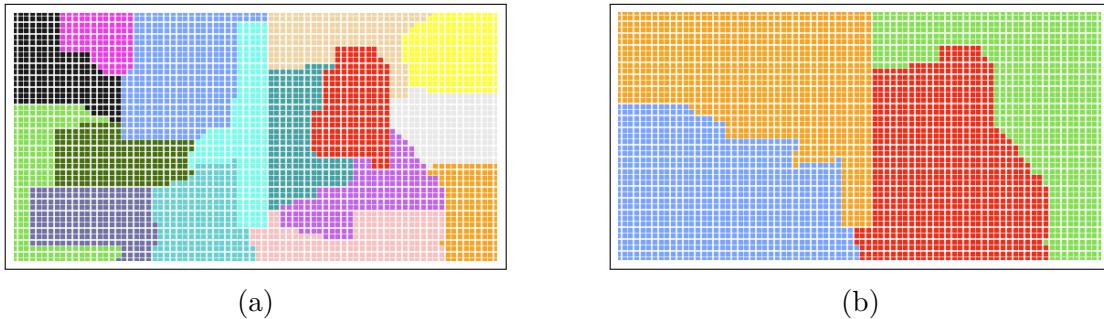
lizacji podziału aktywowany jest dopiero na grafie przywróconym w 85 – 90% do pierwotnego rozmiaru. Wtedy szansa na powstanie obszarów rozproszonych znacznie spada. Kolejnym czynnikiem ryzyka są obszary niepodzielne. Reprezentują je wierzchołki, które zostają zastąpione początkowym zbiorem wierzchołków dopiero po zakończeniu optymalizacji. Przeniesienie takiego obszaru do innej partycji w fazie optymalizacji, tak samo jak w poprzednim przypadku, może skutkować powstaniem obszarów rozproszonych. Bardzo często obszary rozproszone pojawiają się również w początkowych wywołaniach optymalizacji granic, ale są niwelowane przez kolejne wywołania algorytmu optymalizacji. Istotne jest, że algorytm LAM ze względu na swoją charakterystykę dąży do budowania obszarów równych pod względem pola oraz "zwartych", więc taka sytuacja nie zdarza się często. Ponadto, jak wynika z moich obserwacji, pojawiające się obszary rozproszone są zawsze bardzo małe w porównaniu do pozostałych obszarów.

Rozwiążanie tego problemu, które prezentuję w niniejszej pracy, polega na pojedynczym przeiterowaniu po grafie w celu znalezienia takich obszarów, a następnie na przyporządkowaniu wszystkich obszarów rozproszonych do jednego z przylegających do nich obszarów. Algorytm usuwania obszarów rozproszonych jest wywoływany raz, na grafie przywróconym do początkowych rozmiarów, po wszystkich wywołaniach algorytmu optymalizacji granic i wielkości pól.

Zdecydowałem się na takie rozwiązanie z racji na rzadkość występowania tego problemu oraz w związku z faktem, że zwykle powstałe obszary rozproszone są bardzo małe. Według mnie takie rozwiązanie jest dużo tańsze obliczeniowo i bardziej racjonalne przy skali tego zjawiska niż próba wykluczenia wystąpienia takich sytuacji dodatkowymi modyfikacjami algorytmu.

### 3.2. Podział siatki na $m$ obszarów

W związku z tym, że korzystamy z  $m$  węzłów obliczeniowych typu homogenicznego, gdzie każdy węzeł posiada  $k$  rdzeni, siatkę podzieloną na  $m \cdot k$  części należy podzielić na  $m$  obszarów, każdy składający się z  $k$  podobszarów. Bardzo ważną obserwacją jest to, że w przeciwieństwie do poprzedniej części partycjonowania grafu na  $m \cdot k$  obszarów, gdzie dopuszczalne były nierówności w wielkości pól, tutaj bardzo ważne jest, aby każdy obszar składał się z takiej samej liczby podobszarów. Jest to poważne utrudnienie tego problemu.



Rysunek 30: Obrazek (a) podział na  $m \cdot k$  obszarów. Obrazek (b) przedstawia podział  $m \cdot k$  obszarów na  $m$  obszarów po  $k$  podobszarów.  $m = 4$  oraz  $k = 4$ .

Ta część algorytmu jest realizowana przez algorytm weighted matching z dodatkiem części zachłannej. Daną wejściową do algorytmu jest siatka z podziałem na  $m \cdot k$  obszarów, jak na rysunku 30(a). Najpierw wszystkie partycje redukowane są do pojedynczych wierzchołków. Oznacza to, że mamy  $m \cdot k$  wierzchołków. Następnie graf składający się z  $m \cdot k$  wierzchołków redukowany jest do  $m$  wierzchołków za pomocą algorytmu weighted matching, identycznego jak na rysunku 1. Po zredukowaniu do  $m$  wierzchołków, do każdego wierzchołka przypisywana jest inna partycja. Następnie graf jest odtwarzany do rozmiaru  $m \cdot k$  z zachowaniem nowych partycji. W tym momencie otrzymujemy podział  $m \cdot k$  wierzchołków na  $m$  obszarów.

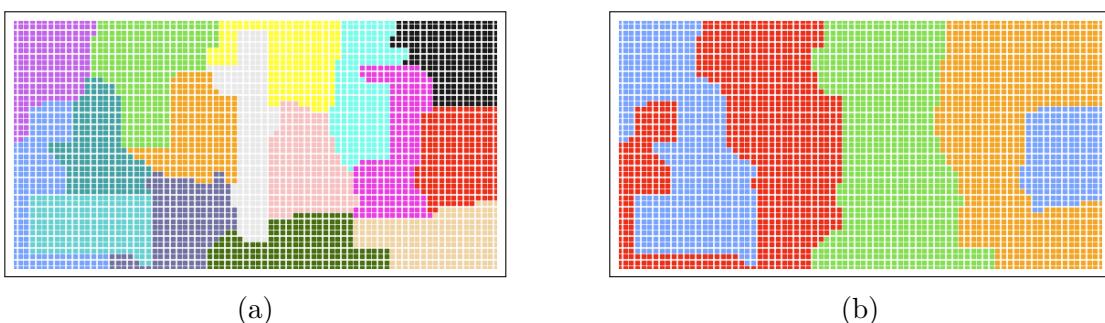
Charakterystyczną cechą algorytmu weighted matching jest to, że od idealnie równych podziałów bardziej ceni sobie obszary "zwarte", czyli takie, które mają możliwie krótkie granice. Jest to istotnym elementem jego charakterystyki, którego nie da się wyeliminować. W związku z powyższym, większość wywołań algorytmu partycjonowania nie kończy się podziałem  $m$  obszarów po  $k$  podobszarów każdy, natomiast podziałem bliskim temu podziałowi, bo choć LAM nie gwarantuje podziałów idealnie równych, gwarantuje powstanie obszarów o zbliżonej liczbie wierzchołków.

Czasami jednak na tym etapie otrzymujemy już  $m$  obszarów po  $k$  podobszarów każdy. Jeśli tak nie jest, to uruchamiana jest zachłanna część algorytmu, która wyrównuje liczbę podobszarów pomiędzy obszarami. Składa się ona z dwóch części. Najpierw algorytm sprawdza jakie obszary leżą obok siebie i wyrównuje pola sąsiadów. Przykładowo, jeśli leżą obok siebie dwa obszary  $A$  oraz  $B$ , z czego  $A$  zawiera  $k - 1$  podobszarów, natomiast  $B$   $k + 1$  podobszarów, to jeden z podobszarów granicznych przenoszony jest z  $B$  do  $A$  w celu wyrównania liczby podobszarów do  $k$  dla każdego z nich. Jeśli  $B$  zawierałby  $k + 3$  obszary, natomiast  $A$   $k - 1$ , to z  $B$  przeniesione zostaną dwa obszary do  $A$ . W ten sposób podobszary z dużych obszarów mają sposobność "rozproszyć" się po siatce. W kolejnej

turze nadmiarowe obszary z  $A$  mają szansę zostać przeniesione gdzie indziej. Ta część algorytmu wywoływana jest dopóki nie przestanie przynosić nowych zmian. Ponieważ operujemy na wierzchołkach, które mają duże wagi, to już na tym etapie przesunięcia obszarów powodują często tworzenie się obszarów rozproszonych.

Jeśli po tym etapie nadal nie mamy równych obszarów, następuje ostatni etap algorytmu. Ten etap działa dopóki wszystkie obszary nie będą miały takiej samej liczby wierzchołków. Wybierany jest zawsze obszar o największej i najmniejszej liczbie podobszarów. Następnie z obszaru o większej liczbie podobszarów losowne są podobszary, które są przenoszone do obszaru o mniejszej liczbie podobszarów, dopóki wybrana para nie będzie miała równego rozmiaru.

Zaletą tego algorytmu jest niski koszt obliczeniowy oraz prostota - im bardziej udany podział zwrócony przez algorytm weighted matching, tym większa szansa na dobry podział końcowy. Liczby wierzchołków, na których operuje algorytm, są bardzo małe, więc można wielokrotnie wywoływać cały algorytm w poszukiwaniu najlepszego rezultatu. Jego prostota oraz zachłanny charakter sprawiają jednak, że jakość podziałów pod względem długości granic może nie być najwyższa - rysunek 31.



Rysunek 31: Obrazek (a) podział na  $m \cdot k$  obszarów. Obrazek (b) przedstawia podział  $m \cdot k$  obszarów na  $m$  obszarów po  $k$  podobszarów.  $m = 4$  oraz  $k = 4$ .

## 4. Wyniki

Rezultaty obliczeń były otrzymywane w podobny sposób jak w artykule [24]. Algorytm wywoływany był 100 razy, a na końcu wybierany był najlepszy podział. Jednak w artykule [24] wybranie najlepszego wyniku odbywało się poprzez liczenie długości granic między obszarami, a w moim przypadku byłybrane pod uwagę jeszcze inne kryteria. Autorzy [24] mogli w ten sposób zaprojektować wybieranie najlepszego podziału, ponieważ nie występowały u nich obszary niepodzielne, przez które znacznie trudniej zdefiniować najlepszy podział. W implementacji bez obszarów niepodzielnych, niemal pewne jest otrzymanie obszarów o równych polach. Jeśli założymy, że nasza implementacja zawsze będzie w stanie zwrócić podział równy pod względem wielkości pól - co da się zagwarantować na siatce bez obszarów niepodzielnych, ponieważ tam możliwości balansowania pól są praktycznie nieograniczone - to pozostaje nam minimalizować sumaryczną długość granic. Obszary niepodzielne sprawiają, że często pola partycji nie mogą być równe, a wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic może prowadzić do wyłonienia podziału o dużej nierówności pod względem wielkości pól partycji, co pokażę na dalszych przykładach. Kryterium wyłaniania najlepszego podziału było więc wybierane na podstawie wielkości i ułożenia obszarów niepodzielnych. Jeśli znacznie utrudniały lub uniemożliwiały one wyrównanie powierzchni pól, a także poprzez swoje ułożenie powodowały wydłużenie długości granic, wybierałem kryterium najmniejszej różnicy w wielkości pól, ponieważ był to ważniejszy parametr oraz ponieważ dla tego typu przykładów algorytm często dawał obszary o stosunkowo dużych różnicach w wielkości pól. Kiedy dowolność ułożenia partycji była duża z racji na małe obszary niepodzielne, kryterium była najmniejsza długość granic. W przypadku siatek, dla których wybór kryterium nie jest oczywisty można wybrać najlepszy podział osobno wedle każdego kryterium, a następnie podjąć decyzję na podstawie ich parametrów. Kryterium najmniejszej różnicy w wielkości pól może być liczne za pomocą odchylenia standardowego bądź w mniej skomplikowanych przypadkach za pomocą różnicy w wielkości pola między największą i najmniejszą partycją.

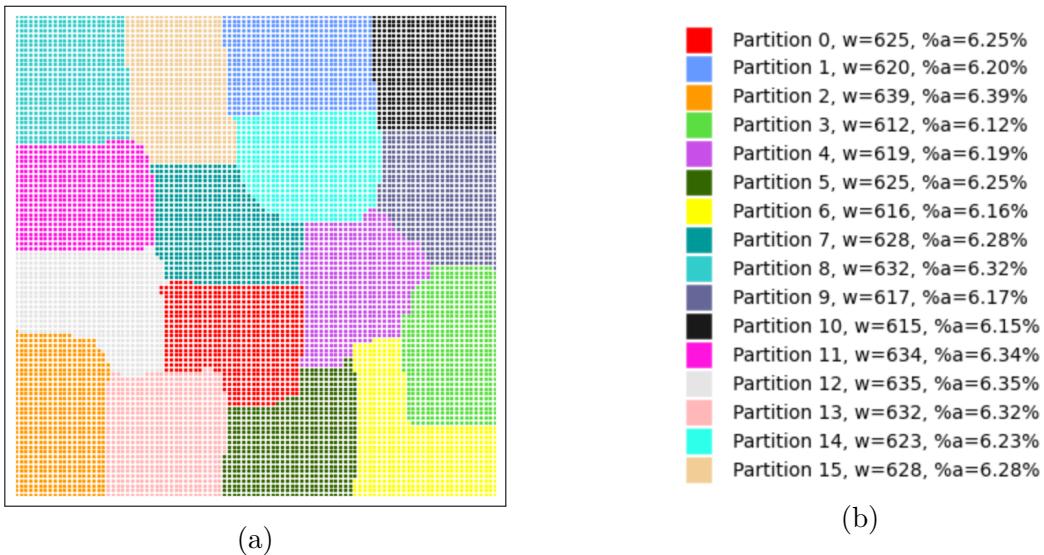
Dla wszystkich wyników prezentowanych w rozdziale 5.1 rysunek przedstawiający siatkę wejściową rysowany jest po lewej stronie. Kolorem żółtym zaznaczone są obszary niepodzielne, natomiast kolorem czerwonym obszary wyłączone z obliczeń. Odchylenie standardowe liczne jest dla procentowych udziałów wielkości partycji w całkowitej wielkości pola do podziału. Dla wyników prezentowanych w rozdziale 5.2 pierwszy rysunek od lewej prezentuje siatkę podzieloną na  $m \cdot k$  obszarów. Następny rysunek przedstawia jej partycjonowanie na  $m$  obszarów po  $k$  podobszarów każdy.

### 4.1. Wyniki dla podziału na $m \cdot k$ obszarów

Ten podrozdział opisuje wyniki podziału siatki dla  $m$  node'ów, każdy zawierający  $k$  rdzeni. Celem jest otrzymanie  $m \cdot k$  możliwie równych pod względem wielkości pola partycji oraz minimalizacja długości granic między nimi.

#### 4.1.1. Wyniki dla podziału siatek bez obszarów niepodzielnych i wyłączonych z obliczeń

Dla tej kategorii, kryterium była najmniejsza sumaryczna długość granic pomiędzy obszarami. Na rysunku 32 widzimy partycjonowanie siatki 100x100 bez obszarów wyłączonych z obliczeń oraz obszarów niepodzielnych. Rezultat pod względem długości granic jest bardzo dobry w porównaniu do innych bibliotek, których wyniki znajdują się na rysunku 7. Wszystkie te biblioteki traktowane są przez literaturę jako biblioteki dające wyniki state-



Rysunek 32: Siatka 100x100. Podział na 16 partycji. Sumaryczna długość granic dla tego wyniku wynosi 686. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standardowe wielkości pól wynosi 0.0779.

of-the-art. Z wynikiem 686 metoda przedstawiona w tej pracy plasuje się blisko biblioteki Jostle i Metis, które otrzymały odpowiednio 695 oraz 688. Metoda zaprezentowana w niżej położonej pracy bazuje na metodzie używanej przez bibliotekę Party, która dla tego problemu otrzymuje wynik 615 - jest to wynik bliski idealnemu. Wynikiem idealnym dla długości granic byłby wynik 600. Cechą charakterystyczną tego podziału jest to, że ze względu na brak obszarów niepodzielnych, które mogą utrudniać wymianę granicznych wierzchołków, udało się otrzymać niemal idealnie równe pola. Świadczy o tym niskie odchylenie standardowe od średniej wielkości pola partycji. Trudno powiedzieć, dlaczego wystąpiła aż tak duża różnica pod względem długości granic w stosunku do biblioteki Party. Może przesądziły o tym różnice w implementacji, które najprawdopodobniej wyniknęły z nie dość szczegółowo opisanego algorytmu w artykule dotyczącym tej biblioteki [24]. Przyglądając się mojemu rezultatowi, można zaproponować kilka pomniejszych usprawnień, które mogłyby jeszcze nieco polepszyć wynik. Można by przykładowo kosztem wielkości pól, zmusić algorytm do tworzenia prostych granic między obszarami. Modyfikacja, która sprawiałaby, że obszary ustawiałyby się tak jakby siatka dzielona była pionowymi i poziomymi separatorami jest trudna do wprowadzenia w algorytmie, który ma charakterystykę losową. Prawdopodobieństwo takiego podziału jest bardzo znikome. Jest to możliwe dla biblioteki [24]. W związku z powyższym przypuszczam, że w algorytmie biblioteki Party [24] mogły zostać wprowadzane dodatkowe rozwiązania nieopisane w artykule. Kolejną

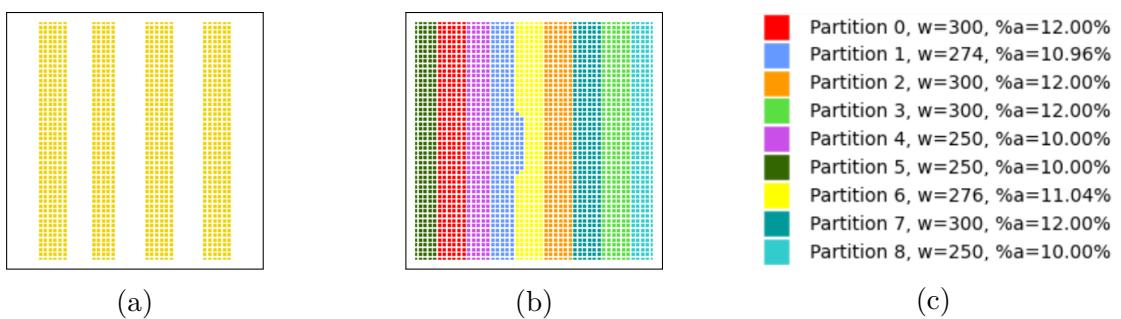
różnicą jest czas wykonania partycjonowania, który dla sprzętu wyposażonego w Pentium III 933 MHz dual processor wraz z 1 GB pamięci operacyjnej dla Party wyniósł 0.03s. Dla mojego sprzętu wyposażonego w 2.3 GHz Quad-Core Intel Core i5 z 2018 roku wraz z 8 GB pamięci operacyjnej czas ten wyniósł 18s. Tam gdzie było to możliwe wybierałem większe zużycie pamięci kosztem krótszych obliczeń. Starałem się również wybierać struktury danych ze stałym czasem dostępu do informacji. Operacje umieszczone w algorytmie ze względu na obszary niepodzielne oraz wyłączone z obliczeń nie wprowadzają modyfikacji, które mogłyby w aż tak znacznym stopniu wpływać na złożoność obliczeniową. Być może jest to kwestią kosztownych operacji na grafach dostarczonych przez bibliotekę NetworkX. Rozwiązanie zaproponowane w niniejszej pracy jest prototypem, którego celem nie było uzyskanie lepszej wydajności, ale wykazanie poprawności i jakości rozwiązania. Wydajność natomiast da się poprawić, czego dowodem jest istniejąca implementacja opisana w [24].

#### 4.1.2. Wyniki dla podziału siatek z obszarami niepodzielnymi i wyłączonymi z obliczeń

W tym podrozdziale przedstawione są wyniki dla siatek z obszarami niepodzielnymi i wyłączonymi z obliczeń. Przykłady są celowo dobrane tak, aby były problematyczne dla algorytmu.

##### Partycjonowanie 1

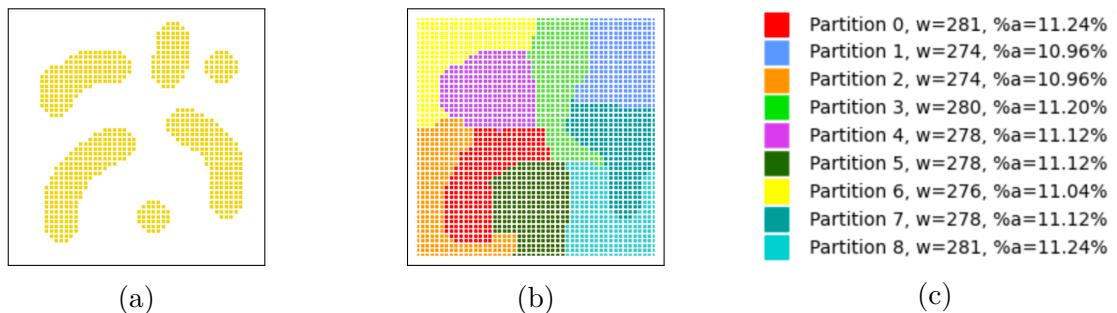
Siatka przedstawiona na rysunku 33 jest trudnym przypadkiem, ponieważ nie daje znacznych możliwości na wyrównanie pól w dalszych częściach algorytmu. Wynik jest więc zależny od algorytmu LAM. Aby otrzymać najlepszy rezultat musi on dobrze ustawić partycje od samego początku. Dla tego przypadku nie da się uzyskać idealnie równego podziału, a najlepszym rezultatem będzie, gdy każda partycja będzie w osobnym pionowym pasku. Drugi obszar niepodzielny jest o jeden piksel węższy od pozostałych, dlatego w rezultacie partycjonowania pomiędzy partycją 6 a 1 następuje wyrównanie pola. Otrzymany rezultat jest najlepszym lub niemal najlepszym jaki można otrzymać.



Rysunek 33: Siatka 50x50. Podział na 9 partycji. Sumaryczna długość granic dla tego wyniku wynosi 402. Kryterium wyboru najlepszego rezultatu to najmniejsza różnica pomiędzy polem największej i najmniejszej partycji. Odchylenie standardowe wielkości pól wynosi 0.875.

## Partycjonowanie 2

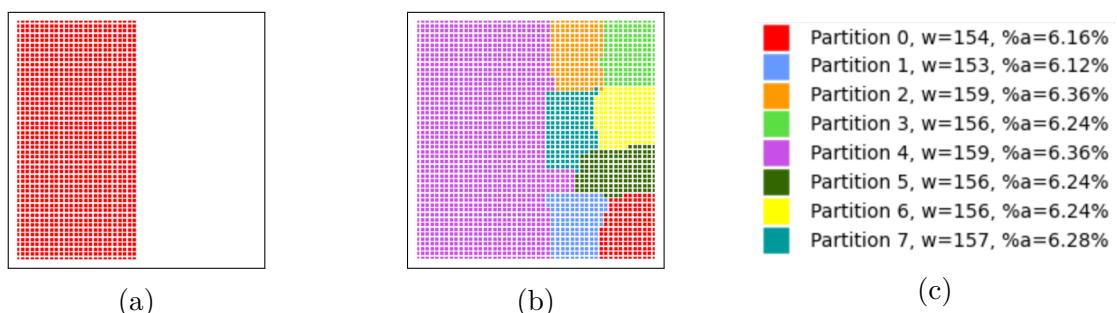
Siatka partycjonowania na rysunku 34 daje możliwość wymieniania się przez obszary wierzchołkami granicznymi. Jest to jednak znacznie utrudnione przez obszary niepodzielne. Wyniki wybierane według kryterium najkrótszej długości granic dawały rezultaty z dużymi różnicami pod względem wielkości pól partycji. Rezultat daje niemal idealnie równe pola partycji, o czym świadczy niska wartość odchylenia standardowego od średniej wielkości pola partycji. Obszary są raczej zwarte. Jest to dobry wynik.



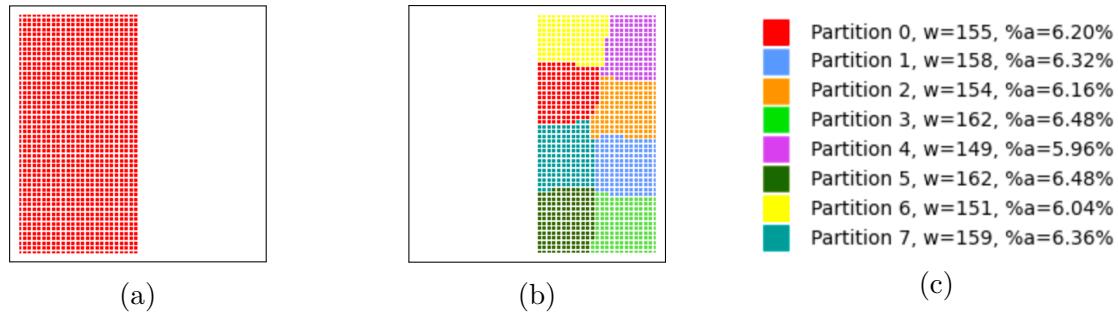
Rysunek 34: Siatka 50x50. Podział na 8 partycji. Sumaryczna długość granic dla tego wyniku wynosi 313. Kryterium wyboru najlepszego rezultatu to najmniejsze odchylenie standardowe dla pól partycji. Odchylenie standardowe wielkości pól wynosi 0.1011.

## Partycjonowanie 3

Dla siatki przedstawionej na rysunku 35 sprawdzane jest, czy algorytm poprawnie uwzględnia obszary wyłączone z obliczeń. Na rezultacie widać, że wszystkie pola są niemal identyczne pod względem wagi. Pole partycji 4 urosło do dużych rozmiarów, jednak mimo to ma taką samą wagę jak pozostałe pola. Algorytm poprawnie uwzględnia obszary wyłączone z obliczeń. Na rysunku 36 przedstawiono partycjonowanie tej samej siatki, ale dla tego przypadku obszary wyłączone z obliczeń nie są mapowane na wierzchołki w grafie. W efekcie partycjonowanie trwa krócej, a długość granic jest znacznie mniejsza. Oba rezultaty mają bardzo zbliżone wielkości pól partycji, jednak to przykład przedstawiony na rysunku 35 posiada niższe odchylenie standardowe dla wielkości partycji.



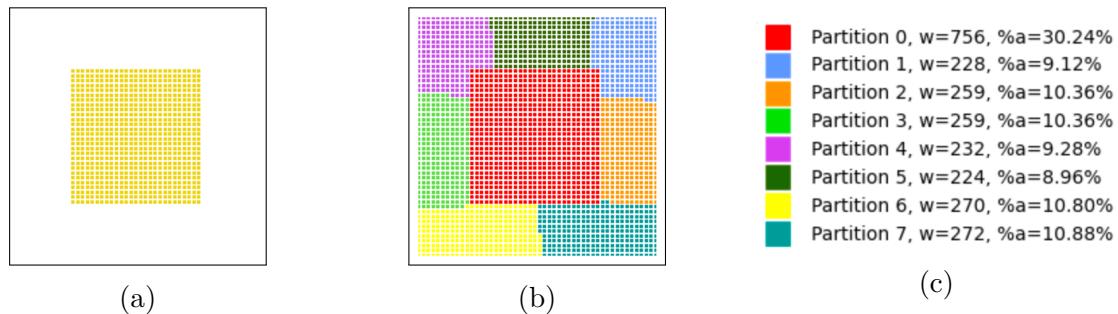
Rysunek 35: Siatka 50x50. Podział na 8 partycji. Obszary wyłączone z obliczeń mapowane na wierzchołki z wagą 0. Sumaryczna długość granic dla tego wyniku wynosi 177. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standardowe wielkości pól wynosi 0.0793.



Rysunek 36: Siatka 50x50. Podział na 8 partycji. Obszary wyłączone z obliczeń nie są mapowane na wierzchołki. Sumaryczna długość granic dla tego wyniku wynosi 137. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standar-dowe wielkości pól wynosi 0.1808.

#### Partycjonowanie 4

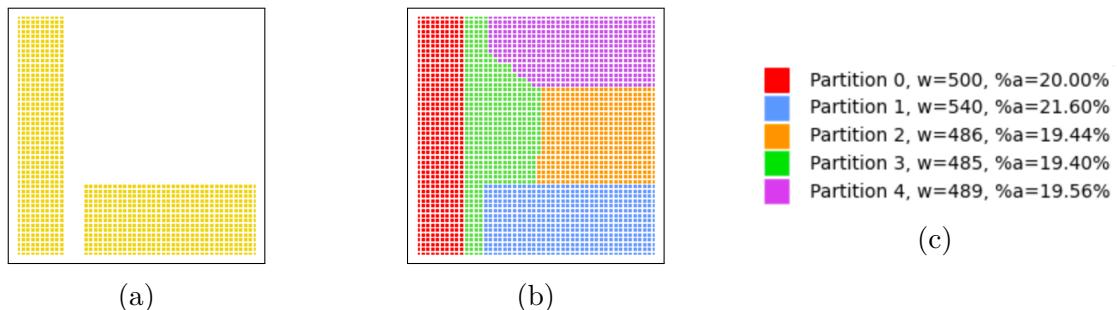
Dla siatki przedstawionej na rysunku 37 sprawdzane jest działanie *discount* dla algorytmu LAM. Tego typu podziały są trudne dla algorytmu. W rezultacie otrzymujemy podział, gdzie wszystkie obszary dookoła obszaru czerwonego mają niemal równe pola, co jest naj-lepszym możliwym rezultatem dla tej siatki. Nie są one idealnie równe, ponieważ podział był wybierany wedle kryterium długości granic. Uznałem, że jest to lepsze kryterium, ponieważ dla tego przypadku algorytm ma całkiem dużą swobodę w kwestii optymalizacji granic i pól partycji.



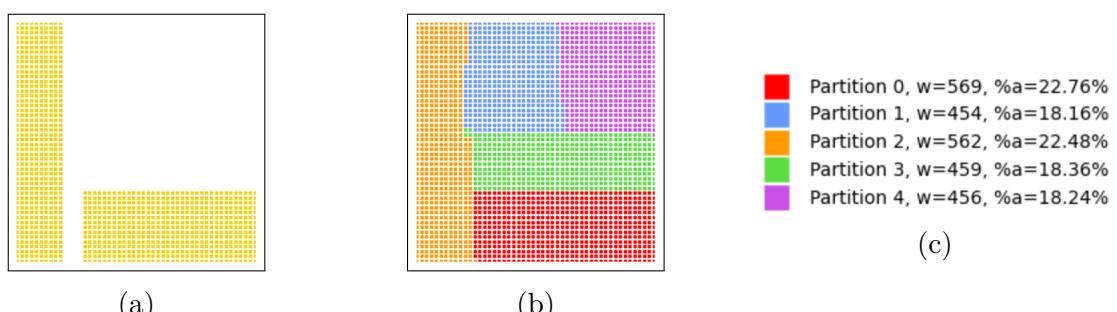
Rysunek 37: Siatka 50x50. Podział na 8 partycji. Sumaryczna długość granic dla tego wyniku wynosi 196. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standardowe wielkości pól bez partycji 0 wynosi 0.7596, natomiast z partycją 0 wynosi 6.7426.

### Partycjonowanie 5

Rysunek 38 oraz 39 przedstawia partycjonowanie tej samej siatki. Dla rysunku 38 najlepszy rezultat został wybrany poprzez zastosowanie kryterium najmniejszego odchylenia standardowego dla wielkości pól partycji, natomiast dla rysunku 39 poprzez kryterium najmniejszej długość granic. Widać jak różne podziały może tworzyć ten sam algorytm w zależności od początkowego ustawienia obszarów. Na rysunku 38 otrzymaliśmy partycje o bardziej zbliżonych wielkościach, kosztem większej długości granic pomiędzy obszarami. Na rysunku 39 otrzymaliśmy krótsze granice, kosztem równości pól.



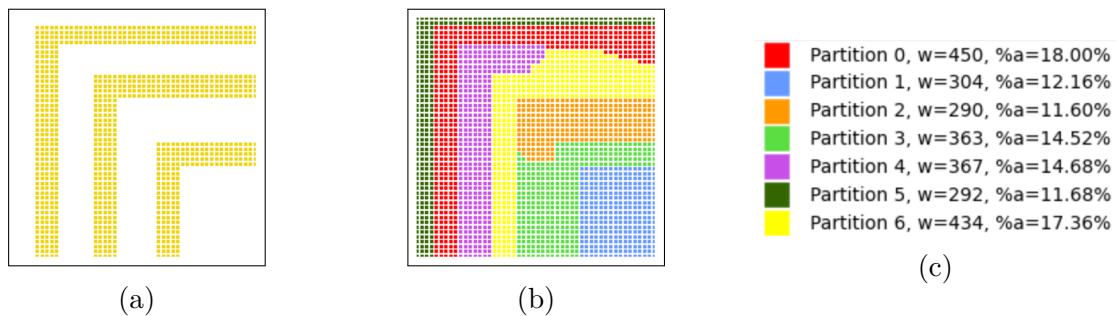
Rysunek 38: Siatka 50x50. Podział na 5 partycji. Sumaryczna długość granic dla tego wyniku wynosi 172. Kryterium wyboru najlepszego rezultatu to najmniejsze odchylenie standardowe dla pól partycji. Odchylenie standardowe wielkości pól wynosi 0.8279.



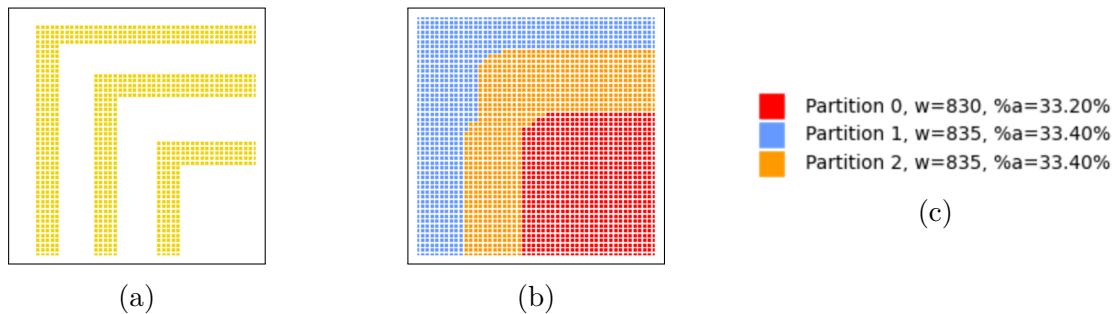
Rysunek 39: Siatka 50x50. Podział na 5 partycji. Sumaryczna długość granic dla tego wyniku wynosi 159. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standardowe wielkości pól wynosi 2.1419.

### Partycjonowanie 6

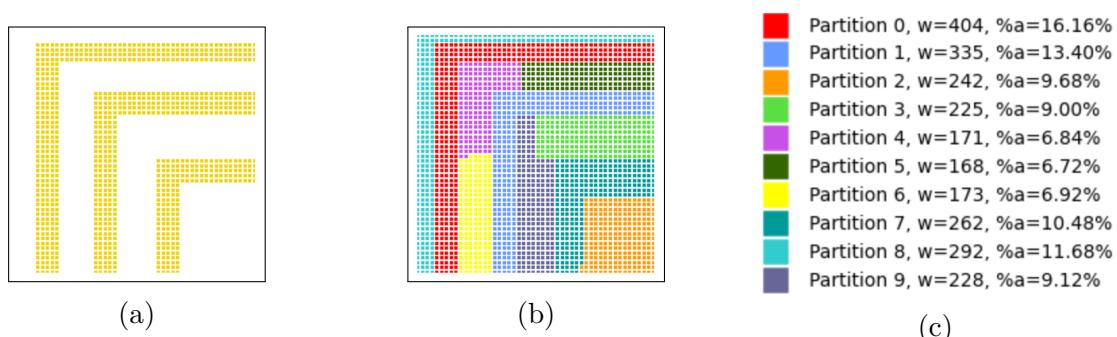
Rysunek 40, 41 oraz 42 przedstawia partycjonowanie tej samej siatki, odpowiednio na 7, 3 oraz 10 części. Jest to siatka bardzo trudna do partycjonowania, ponieważ z racji na ułożenie obszarów niepodzielnych nie pozwala na zbytnie optymalizowanie długości granic oraz wielkości pól partycji. Obszary niepodzielne będą wymuszać kształt granic oraz wielkość partycji. Pierwszy podział jest na 7 części, ponieważ tyle jest białych i żółtych stref na rysunku. Algorytm próbuje wyrównywać pola na tyle, na ile jest to możliwe. Widać, że partycje, które ze sobą sąsiadują i mają możliwość wyrównania wielkości pól są podobnej wielkości. Jest to na przykład partycja 6 oraz 0.



Rysunek 40: Siatka 50x50. Podział na 7 partycji. Sumaryczna długość granic dla tego wyniku wynosi 369. Kryterium wyboru najlepszego rezultatu to najmniejsze odchylenie standardowe dla pól partycji. Odchylenie standardowe wielkości pól wynosi 2.4488.



Rysunek 41: Siatka 50x50. Podział na 3 partycje. Sumaryczna długość granic dla tego wyniku wynosi 141. Kryterium wyboru najlepszego rezultatu to najmniejsze odchylenie standardowe dla pól partycji. Odchylenie standardowe wielkości pól wynosi 0.0942.

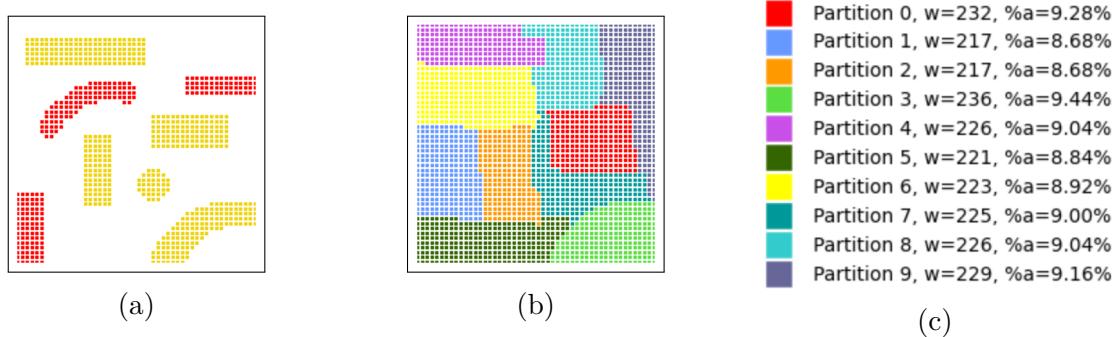


Rysunek 42: Siatka 50x50. Podział na 10 partycji. Sumaryczna długość granic dla tego wyniku wynosi 417. Kryterium wyboru najlepszego rezultatu to najmniejsze odchylenie standardowe dla pól partycji. Odchylenie standardowe wielkości pól wynosi 2.9097.

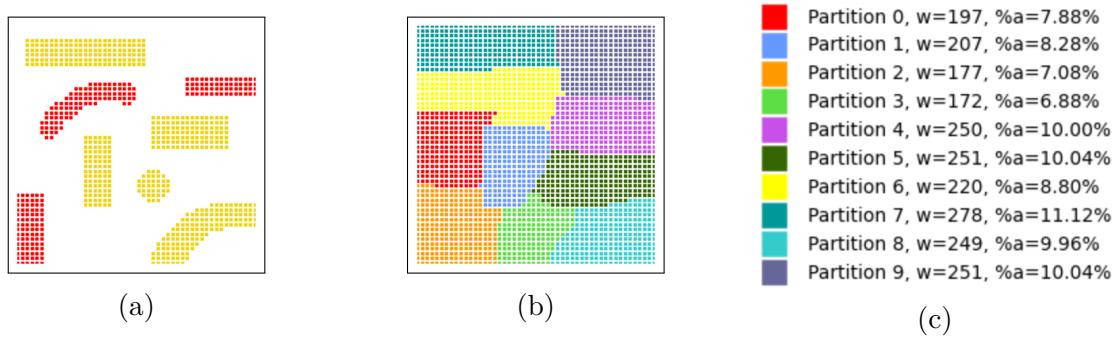
Można zaobserwować, że partycjonowanie na 3 części na rysunku 41 daje znacznie więcej swobody pod względem balansowania wielkości pól. W rezultacie partycje są niemal identycznej wielkości. Ustawienie obszarów niepodzielnych nadal warunkuje ułożenie partycji i zwiększa długość granic. Partycjonowanie przedstawione na rysunku 42 to bardziej ekstremalna wersja partycjonowania od tej na rysunku 40. Partycje, które miały możliwość wyrównania pola między sobą, jak partycje 6, 4 oraz 5, mają praktycznie identyczne pola. Algorytm bardzo dobrze radzi sobie z tym trudnym przypadkiem.

### Partycjonowanie 7

Rysunek 43 oraz 44 pokazuje bardzo podobną sytuację jak dla rysunku 38 oraz 39. Dla rysunku 43 otrzymaliśmy równiejsze pola, kosztem większej długości granic pomiędzy obszarami. Na rysunku 44 otrzymaliśmy krótsze granice, kosztem równości pól. Różnicą w tym przypadku jest to, że na siatce pojawiają się zarówno obszary wyłączone z obliczeń jak i obszary niepodzielne. Algorytm bardzo dobrze radzi sobie z uwzględnieniem obydwu, otrzymuje krótkie granice oraz niemal równe obszary.

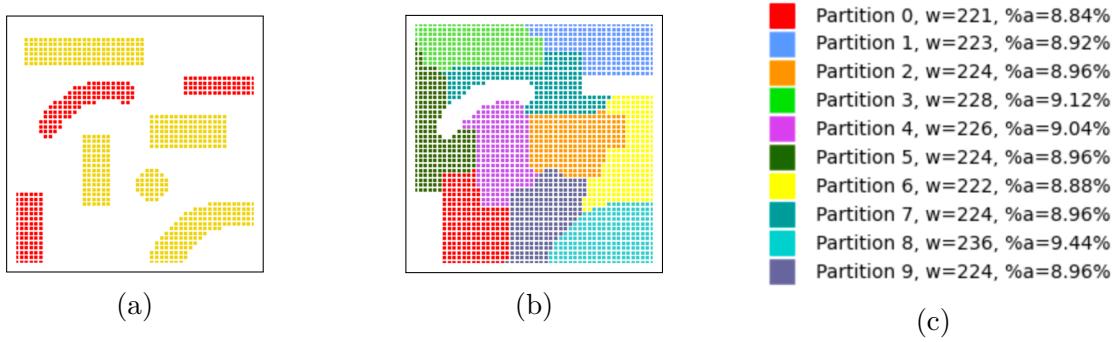


Rysunek 43: Siatka 50x50. Podział na 10 partycji. Obszary wyłączone z obliczeń mapowane na wierzchołki z wagą 0. Sumaryczna długość granic dla tego wyniku wynosi 295. Kryterium wyboru najlepszego rezultatu to najmniejsze odchylenie standardowe dla pól partycji. Odchylenie standardowe wielkości pól wynosi 0.2317.

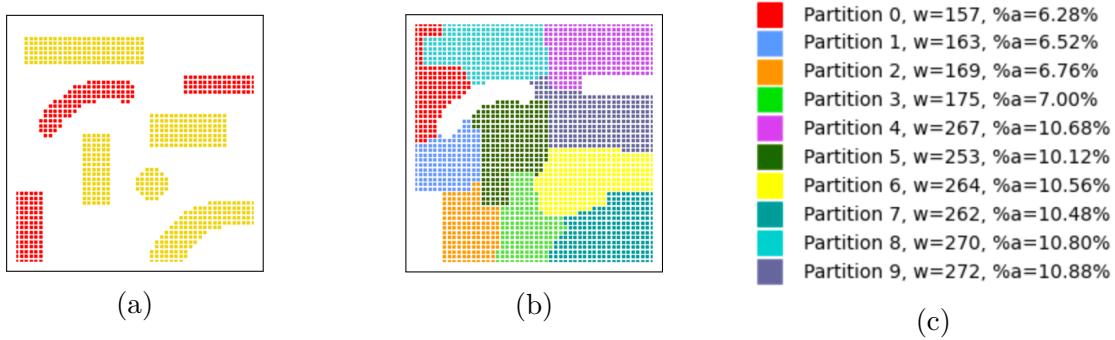


Rysunek 44: Siatka 50x50. Podział na 10 partycji. Obszary wyłączone z obliczeń mapowane na wierzchołki z wagą 0. Sumaryczna długość granic dla tego wyniku wynosi 261. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standardowe wielkości pól wynosi 1.3627.

Rysunek 45 oraz 46 prezentują partycjonowanie tej samej siatki, ale z użyciem wariantu algorytmu, gdzie obszary wyłączone z obliczeń nie są mapowane na wierzchołki. W efekcie



Rysunek 45: Siatka 50x50. Podział na 10 partycji. Obszary wyłączone z obliczeń nie są mapowane na wierzchołki. Sumaryczna długość granic dla tego wyniku wynosi 252. Kryterium wyboru najlepszego rezultatu to najmniejsze odchylenie standardowe dla pól partycji. Odchylenie standardowe wielkości pól wynosi 0.1617.



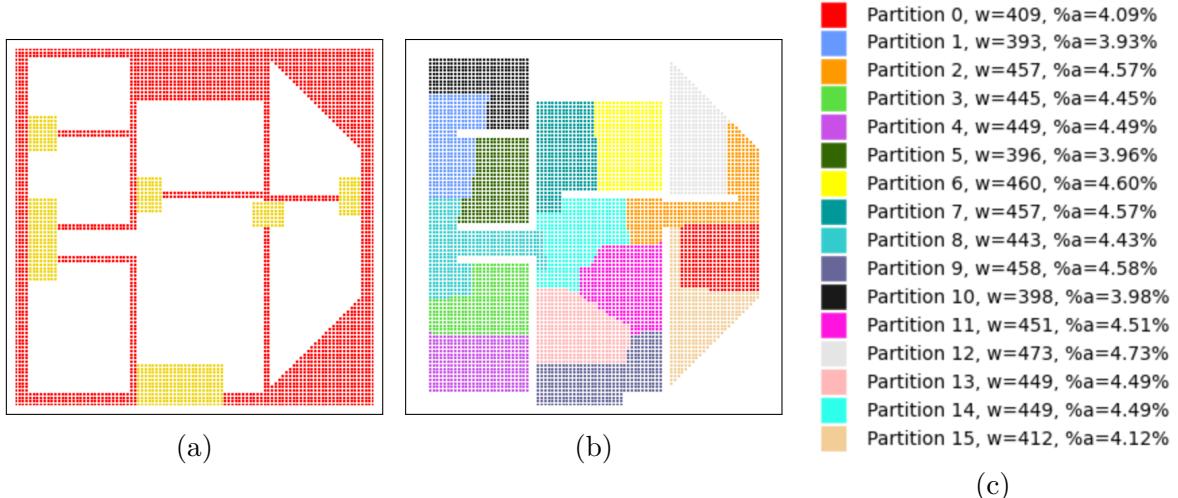
Rysunek 46: Siatka 50x50. Podział na 10 partycji. Obszary wyłączone z obliczeń nie są mapowane na wierzchołki. Sumaryczna długość granic dla tego wyniku wynosi 221. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standardowe wielkości pól wynosi 1.9504.

dla obydwu kryteriów wyboru najlepszego partycjonowania otrzymujemy mniejszą długość granic. Do tej pory dla wszystkich przykładów nietworzenie wierzchołków w miejsce obszarów wyłączeniowych z obliczeń dawało lepsze rezultaty i ten trend utrzyma się do samego końca. Moje testowanie wykazało, że jeśli obszary wyłączone z obliczeń są duże oraz mają dużą granicę w stosunku do wielkości pola to są bardzo problematyczne dla algorytmu LAM, a także znaczenie wydłużają czas partycjonowania. Zawsze szybszym, efektywniejszym wyjściem jest skorzystanie z opcji usunięcia ich z grafu.

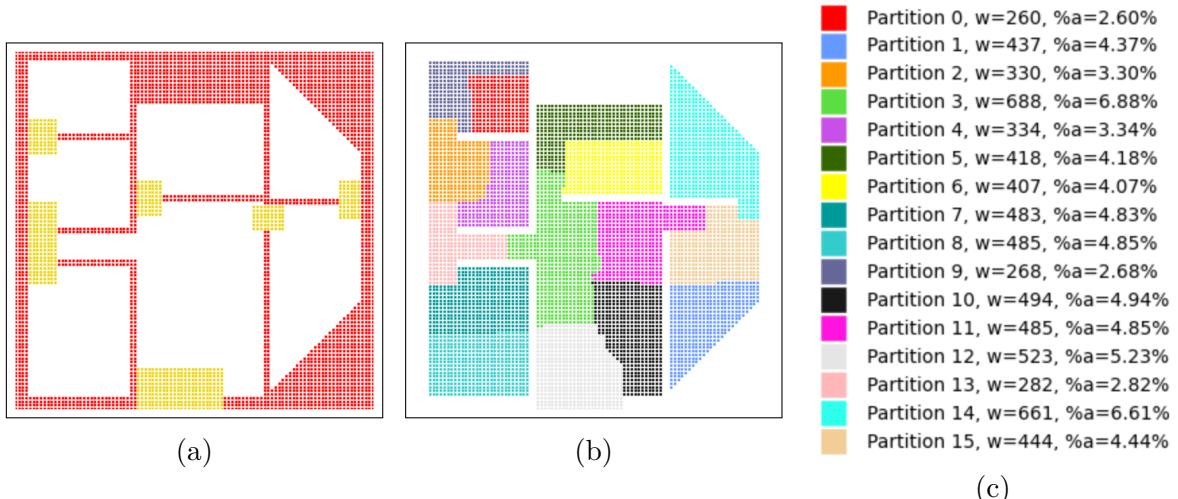
## Partycjonowanie 8

Na rysunkach 47, 48, 49 oraz 50 widać przykład siatki obrazującej mapę. Obszary wyłączone z obliczeń symbolizują ściany oraz pomieszczenia, na których nie będą przebiegały obliczenia. Żółte, niepodzielne obszary to drzwi lub ciasne przejścia. Wygenerowałem cztery różne podziały. Dwa pierwsze nie zamieniają obszarów wyłączeniowych z obliczeń na wierzchołki w grafie. Partycjonowanie przedstawione na rysunku 47 zostało wybrane według kryterium najmniejszego odchylenia standardowego dla wielkości pól obszarów, natomiast partycjonowanie z rysunku 48 według kryterium najmniejszej długości granic. Dla partycjonowań 49 oraz 50 wygląda to analogicznie, tutaj natomiast obszary wyłączone

z obliczeń zostały zamienione na wierzchołki o wagach 0.



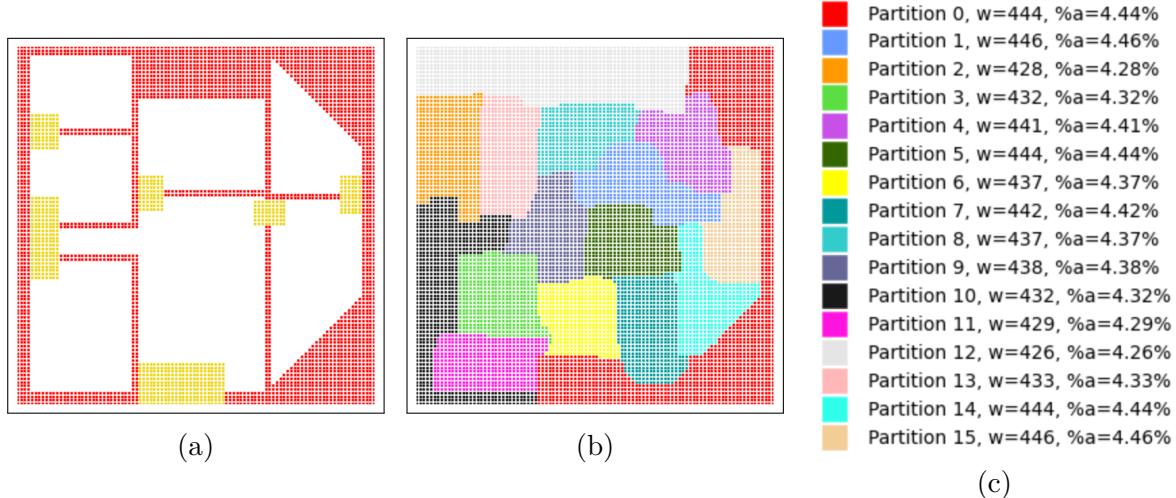
Rysunek 47: Siatka 100x100. Podział na 16 partycji. Obszary wyłączone z obliczeń nie są mapowane na wierzchołki. Sumaryczna długość granic dla tego wyniku wynosi 392. Kryterium wyboru najlepszego rezultatu to najmniejsze odchylenie standardowe dla pól partycji. Odchylenie standardowe wielkości pól wynosi 0.2541.



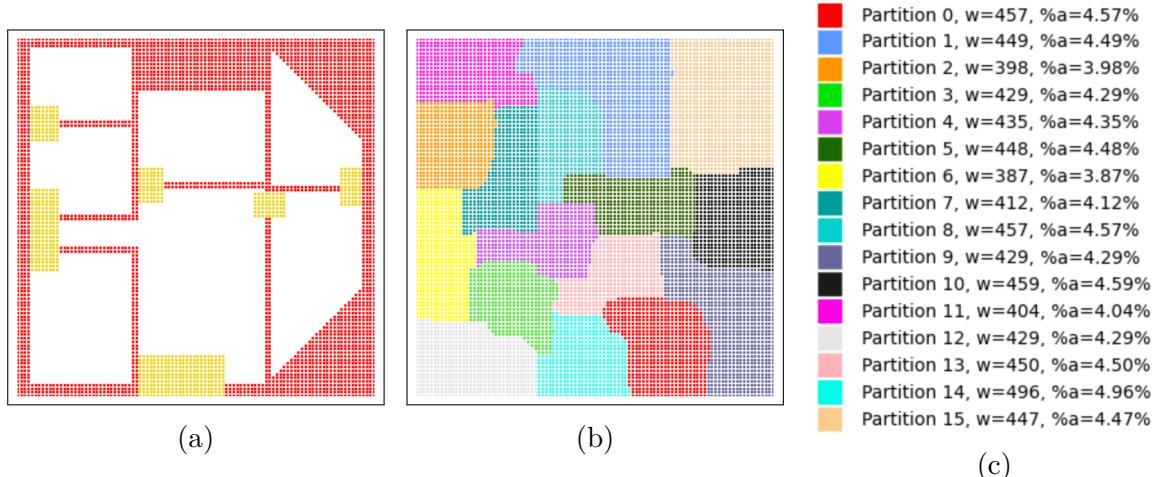
Rysunek 48: Siatka 100x100. Podział na 16 partycji. Obszary wyłączone z obliczeń nie są mapowane na wierzchołki. Sumaryczna długość granic dla tego wyniku wynosi 347. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standardowe wielkości pól wynosi 1.2042.

Rezultaty 47, 48 są lepsze od 49 oraz 50 pod względem długości granic, ze względu na ściany, które ograniczają liczbę krawędzi łączących różne partycje. Natomiast rezultaty 49 oraz 50 są lepsze pod względem równości pól partycji. Wynika z tego, że jeśli wyżej cenimy optymalizację długości granic między partycjami (na przykład jak w niniejszej pracy) oraz na wejściowej siatce nie ma żadnych obszarów, które są całkowicie odłączone od innych (innymi słowy można znaleźć ścieżkę między każdą parą pokoi na mapie), lepsze rezultaty daje metoda, która nie tworzy wierzchołków dla obszarów wyłączenych z obliczeń. Tak jak wcześniej, tam gdzie wybierany jest podział według kryterium najmniejszej długości

granic, otrzymujemy krótsze granice kosztem nieco mniej równego podziału, dla drugiego kryterium jest na odwrót. Na rysunku 49 widać, że pojedyncze partycje mają tendencję do zabierania dużych pól obszarów wyłączonych z obliczeń wydłużając niepotrzebnie granice (partycja 0, 10).



Rysunek 49: Siatka 100x100. Podział na 16 partycji. Obszary wyłączone z obliczeń nie są mapowane na wierzchołki. Sumaryczna długość granic dla tego wyniku wynosi 908. Kryterium wyboru najlepszego rezultatu to najmniejsze odchylenie standardowe dla pól partycji. Odchylenie standardowe wielkości pól wynosi 0.0651.



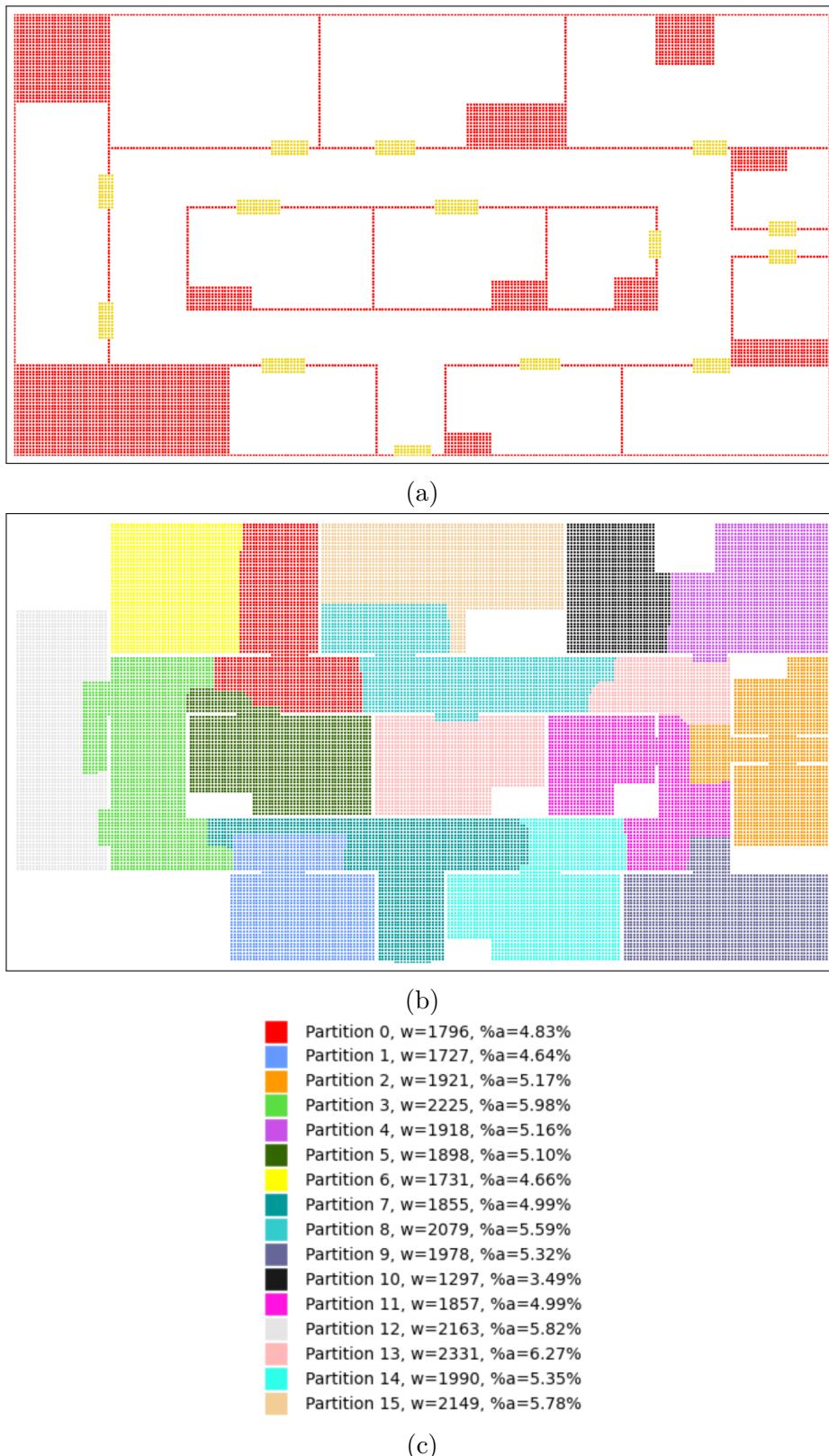
Rysunek 50: Siatka 100x100. Podział na 16 partycji. Obszary wyłączone z obliczeń nie są mapowane na wierzchołki. Sumaryczna długość granic dla tego wyniku wynosi 739. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standardowe wielkości pól wynosi 0.2649.

## Partycjonowanie 9

Na rysunku 51 widoczne jest partycjonowanie siatki o większym rozmiarze, mającej przypominać plan budynku. Tym razem ze względu na czas liczenia, najlepszy rezultat został wybrany spośród 10 wywołań algorytmu. Algorytm bardzo dobrze poradził sobie z zadaniem, otrzymując podobne pod względem wielkości pola obszary oraz uwzględniając obszary niepodzielne i wyłączone z obliczeń. Powstał jeden obszar rozproszony dla partycji numer 13. Dla tego przykładu usuwanie obszarów rozproszonych było wyłączone, ponieważ ściany bardzo ograniczają możliwości balansowania pól i nie jest oczywistym, że obszary nierożproszone będą zawsze dawać najlepsze rezultaty. Ta część algorytmu daje bardzo dobre rezultaty bez względu na wielkość siatki wejściowej. Wielokrotne, kosztowne obliczeniowo wywołania algorytmu miały na celu wykazanie najlepszych rezultatów, jednak algorytm ten działa bardzo stabilnie i również bez wielokrotnych wywołań dostarcza dobre rezultaty. Wyniki odchylenia standardowego wielkości pól partycji dla dziesięciu kolejnych wywołań, dla siatki przedstawionej na rysunku 51 (jest to inne wywołanie algorytmu, niż to z którego został wybrany wynik przedstawiony na rysunku 51):

1.  $std = 0.7902, cut\_size = 531$
2.  $std = 0.8277, cut\_size = 552$
3.  $std = 0.5269, cut\_size = 607$
4.  $std = 0.4602, cut\_size = 473$
5.  $std = 0.3403, cut\_size = 601$
6.  $std = 1.1961, cut\_size = 583$
7.  $std = 0.4940, cut\_size = 545$
8.  $std = 0.8982, cut\_size = 497$
9.  $std = 0.7613, cut\_size = 615$
10.  $std = 0.6832, cut\_size = 578$

Wyniki są zależne od wielu czynników, na przykład od wyboru stopnia agresywności balansowania pól lub od tego, czy pozwolimy balansować pola przez krótkie granice. Dla przykładu jak na rysunku 51, ma to bardzo duży wpływ na rezultat.

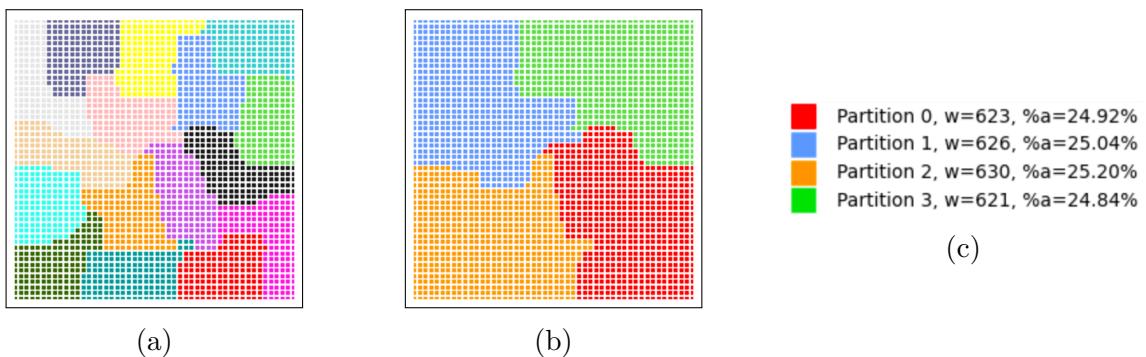


Rysunek 51: Siatka 260x143. Podział na 16 partycji. Obszary wyłączone z obliczeń nie są mapowane na wierzchołki. Sumaryczna długość granic dla tego wyniku wynosi 531. Kryterium wyboru najlepszego rezultatu to najmniejsze odchylenie standardowe dla pól partycji. Odchylenie standardowe wielkości pól wynosi 0.6339.

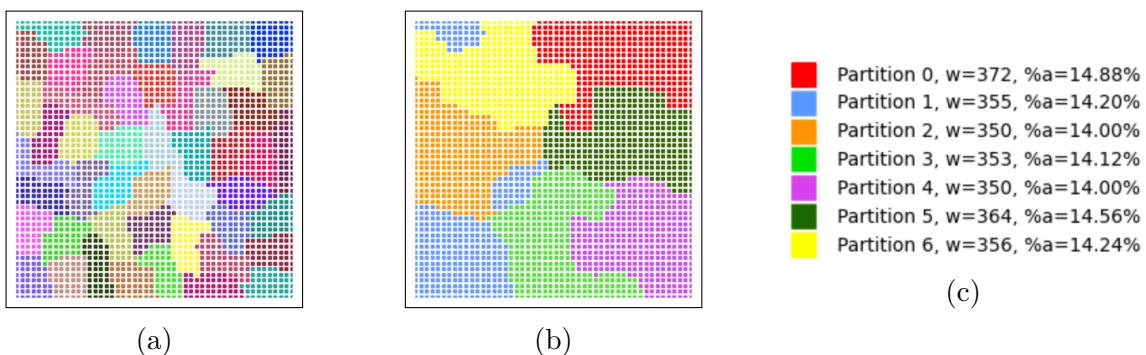
## 4.2. Wyniki dla podziału na $m$ obszarów

Ten podrozdział opisuje wyniki podziału siatki podzielonej na  $m \cdot k$  partycji na  $m$  partycji, z których każda zawiera  $k$  podobszarów. Dla tej części zakładam, że podział na  $m \cdot k$  partycji jest na równe, bądź niemal równe części i podaję wyniki tylko dla podziału na  $m$  partycji po  $k$  podobszarów każda. Różnica w wywołaniu algorytmu dla tej części polega na tym, że wykonywanych jest 100 iteracji partycjonowania na  $m \cdot k$  partycji, a dla każdego z tych podziałów wykonywane jest 100 iteracji poszukiwania najlepszego podzielenia tych  $m \cdot k$  partycji na  $m$  partycji po  $k$  podobszarów każda. Jest to możliwe, ponieważ szukanie najlepszego podziału  $m \cdot k$  partycji na  $m$  partycji po  $k$  podobszarów ma niski koszt obliczeniowy. Ze wszystkich wywołań wybierany jest ten podział na  $m$  partycji, który ma najkrótszą długość granic. Długość granic pod rysunkiem podawana jest dla podziału na  $m$  obszarów.

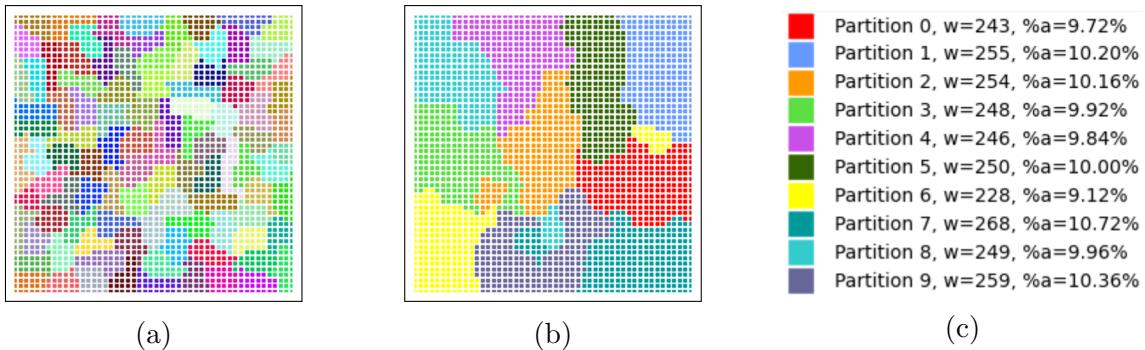
Rysunek 52 pokazuje partycjonowanie dla  $k$  i  $m$  wynoszącego 4. 16 obszarów dzielone jest na 4 partycje po 4 podobszary każda. Podział na  $m$  obszarów nie wykazał żadnych obszarów rozproszonych - każda partycja jest całością, a granica między obszarami jest krótka. Wszystkie z czterech partycji mają niemal równe rozmiary, co znaczy, że wielkości partycji dla podziału (a) są niemal idealnie równe.



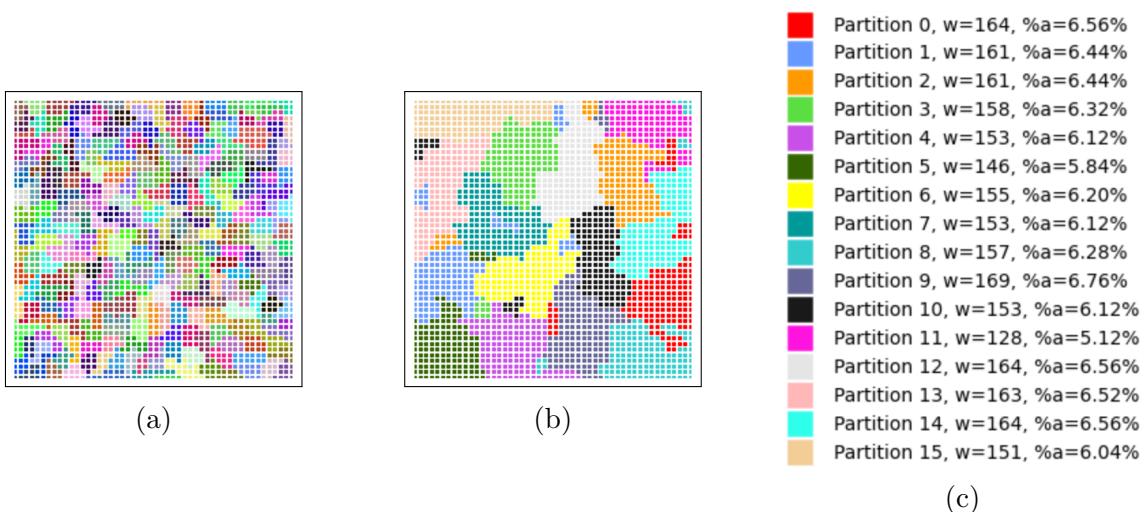
Rysunek 52: Siatka 50x50.  $k$  i  $m$  wynosi 4. Sumaryczna długość granic dla tego wyniku wynosi 146. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standardowe wielkości pól wynosi 0.1356.



Rysunek 53: Siatka 50x50.  $k$  i  $m$  wynosi 7. Sumaryczna długość granic dla tego wyniku wynosi 365. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standardowe wielkości pól wynosi 0.2996.



Rysunek 54: Siatka  $50 \times 50$ .  $k$  i  $m$  wynosi 10. Sumaryczna długość granic dla tego wyniku wynosi 509. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standardowe wielkości pól wynosi 0.4.

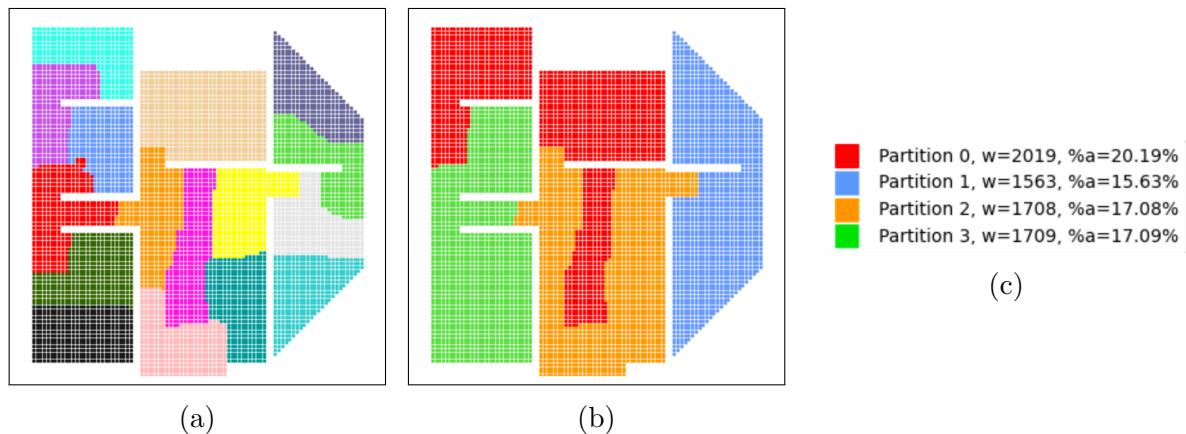


Rysunek 55: Siatka  $50 \times 50$ .  $k$  i  $m$  wynosi 16. Sumaryczna długość granic dla tego wyniku wynosi 793. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standardowe wielkości pól wynosi 0.3742.

Rysunek 53 pokazuje partycjonowanie dla  $k$  i  $m$  wynoszącego 7. 49 obszarów dzielone jest na 7 partycji po 7 podobszarów każda. Po raz pierwszy widoczne są obszary rozproszone, są one jednak pojedyncze i tylko dla partycji 1. Wszystkie z siedmiu partycji mają niemal równe rozmiary, to znaczy, że wielkości partycji dla podziału (a) są niemal idealnie równe. Rysunek 54 pokazuje partycjonowanie dla  $k$  i  $m$  wynoszącego 10. 100 obszarów dzielone jest na 10 partycji po 10 podobszarów każdej. Wraz z liczbą obszarów rośnie liczba partycji rozproszonych, nie jest ona jednak duża. Jest to tylko kilka pojedynczych obszarów. Podział można uznać za udany. Rysunek 55 pokazuje partycjonowanie dla  $k$  i  $m$  wynoszącego 16. 256 obszarów dzielone jest na 16 partycji po 16 podobszarów każdej. Widoczne jest analogiczne zachowanie jak wcześniej, rośnie liczba rozproszonych obszarów, jest ich jednak mniej niż 10%.

Można zaobserwować, że wraz z liczbą partycji rośnie liczba partycji rozproszonych, nie jest to jednak bardzo intensywne zjawisko. Eksperymenty wykazały, że w celu znalezienia najlepszego partycjonowania lepiej wykonać więcej podziałów na  $m \cdot k$  partycji (100 prób), a następnie dla każdego z nich podobną liczbę partycjowań na  $m$  partycji po  $k$  obszarów

(100 prób), niż wykonać mniej podziałów na  $m \cdot k$  partycji (1-10 prób) i dla każdego z nich znacznie więcej partycjowań na  $m$  partycji po  $k$  obszarów (1000-3000 prób). Druga opcja jest możliwa, ponieważ wyliczanie partycjowań na  $m$  partycji po  $k$  obszarów jest bardzo mało kosztowne obliczeniowo, ale dla tej opcji występuje znacznie więcej obszarów rozproszonych i długość granic między partycjami jest większa.



Rysunek 56: Siatka 100x100.  $k$  i  $m$  wynosi 16. Sumaryczna długość granic dla tego wyniku wynosi 200. Obszary wyłączone z obliczeń nie są mapowane na wierzchołki. Wybór najlepszego rezultatu wedle kryterium najmniejszej długości granic. Odchylenie standardowe wielkości pól wynosi 1.6641.

Na rysunku 56 przedstawiono partycjonowanie siatki, która wystąpiła również na rysunku 50. W tym wypadku obszary nie muszą być "zwarte" w celu uzyskania niskiej wartości długości granic, ponieważ ściany ograniczają wartość tego parametru.

## 5. Podsumowanie i kierunek rozwoju

W ramach niniejszej pracy została odtworzona oraz zmodyfikowana metoda partycjonowania grafów używana przez bibliotekę Party [24]. Celem modyfikacji było partycjonowanie siatek dla efektywnego zrównoleglenia symulacji 2D. Symulacje przeprowadzane są na  $m$  węzłach, każdy zawierający  $k$  rdzeni. Siatka musi być podzielona na  $m$  równych partycji po  $k$  równych podobszarów każda. Ważnym wymaganiem było dzielenie z zachowaniem możliwie krótkich granic między obszarami, ponieważ im dłuższe granice, tym większy koszt komunikacji.

Algorytm został zrealizowany w dwóch etapach. Pierwszy dzielił siatkę na  $m \cdot k$  partycji. Drugi dzielił siatkę podzieloną na  $m \cdot k$  partycji na  $m$  partycji, każda po  $k$  równych podobszarów. Nowym elementem było uwzględnienie przy partycjonowaniu obszarów niepodzielnych oraz wyłączonych z obliczeń.

Pierwszy etap podziału na  $m \cdot k$  partycji był najbardziej czasochłonnym elementem pracy. Odtworzenie algorytmu oraz dostosowaniu go pod rozszerzone wymagania zakończyło się sukcesem. Algorytm otrzymał wyniki bliskie bibliotekom uznawanym przez literaturę jako dające wyniki state-of-the-art. Ponadto obsługuje on nieobsługiwane przez te biblioteki obszary niepodzielne oraz wyłączone z obliczeń.

Drugi etap został zrealizowany przy pomocy narzędzi stworzonych na potrzeby pierwszego, dlatego wyniki nie są aż tak dobre. Bazuje głównie na wielokrotnym powtarzaniu obliczeń, a następnie na wybieraniu najlepszych rezultatów. Algorytm wybrany do tego etapu nie do końca nadaje się do dzielenia na partycje o idealnie równej liczbie podobszarów, dlatego musiał zostać dopełniony algorytmem zachłannym.

Cel pracy został osiągnięty. Jako dalsze kierunki rozwoju wskazałbym:

1. Poprawienie implementacji etapu pierwszego, w celu otrzymania wyników bliższych oryginalnej implementacji. Oznacza to poprawienie wyników dla długości granic, ale w szczególności poprawienie wydajności, ponieważ czasy wykonywania stanowiły największą różnicę w stosunku do oryginalnej implementacji.
2. Rozwiniecie drugiego etapu poprzez zaproponowanie innego algorytmu, który w swojej charakterystyce od samego początku został zaprojektowany pod kątem otrzymywania obszarów o równej liczbie wierzchołków lub też udoskonalenie aktualnego rozwiązania.

## Materiały źródłowe

- [1] S. Barnard and H. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. pages 711–718, 01 1993.
- [2] M. Berger and S. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, C-36, 06 1987.
- [3] T. N. Bui and C. Jones. A heuristic for reducing fill-in in sparse matrix factorization. In *PPSC*, 1993.
- [4] T. F. Chan, J. R. Gilbert, and S.-H. Teng. Geometric spectral partitioning. Technical report, 1995.
- [5] C.-K. Cheng and Y.-C. Wei. An improved two-way partitioning algorithm with stable performance (vlsi). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(12):1502–1511, 1991.
- [6] R. Diekmann and B. Monien. Using helpful sets to improve graph bisections. 08 1994.
- [7] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, DAC '82, page 175–181. IEEE Press, 1982.
- [8] J. Garbers, H. Promel, and A. Steger. Finding clusters in vlsi circuits. In *1990 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*, pages 520–523, 1990.
- [9] Hagen and Kahng. A new approach to effective circuit clustering. In *1992 IEEE/ACM International Conference on Computer-Aided Design*, pages 422–427, 1992.
- [10] L. Hagen and A. Kahng. Fast spectral methods for ratio cut partitioning and clustering. In *1991 IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pages 10–13, 1991.
- [11] B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. pages 28– 28, 02 1995.
- [12] J. Hromkovič and B. Monien. The bisection problem for graphs of degree 4 (configuring transputer systems). In A. Tarlecki, editor, *Mathematical Foundations of Computer Science 1991*, pages 211–220, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.
- [13] G. Karypis and V. Kumar. Analysis of multilevel graph partitioning. pages 29– 29, 02 1995.
- [14] G. Karypis and V. Kumar. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

- 
- [15] G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48(1):71–95, 1998.
  - [16] G. Karypis and V. Kumar. Kumar, v.: A fast and high quality multilevel scheme for partitioning irregular graphs. *siam journal on scientific computing* 20(1), 359-392. *Siam Journal on Scientific Computing*, 20, 01 1999.
  - [17] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.
  - [18] R. Leland and B. Hendrickson. An improved spectral graph partitioning algorithm for mapping parallel computations. 16, 09 1992.
  - [19] N. Mansour, R. Ponnusamy, A. Choudhary, and G. C. Fox. Graph contraction for physical optimization methods: A quality-cost tradeoff for mapping data on parallel computers. In *Proceedings of the 7th International Conference on Supercomputing*, ICS ’93, page 1–10, New York, NY, USA, 1993. Association for Computing Machinery.
  - [20] G. Miller, S. Teng, and W. Thurston. A cartesian parallel nested dissection algorithm. 1994.
  - [21] G. Miller, S.-H. Teng, and S. Vavasis. A unified geometric approach to graph separators. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 538–547, 1991.
  - [22] G. L. Miller, S. Teng, W. Thurston, and S. A. Vavasis. Automatic mesh partitioning. In A. George, J. Gilbert, and J. Liu, editors, *Graphs Theory and Sparse Matrix Computation*, The IMA Volumes in Mathematics and its Application, pages 57–84. Springer-Verlag, 1993. Vol 56.
  - [23] B. Monien and R. Preis. Upper bounds on the bisection width of 3- and 4-regular graphs. *Journal of Discrete Algorithms*, 4(3):475–498, 2006. Special issue in honour of Giorgio Ausiello.
  - [24] B. Monien and S. Schamberger. Graph partitioning with the party library: helpful-sets in practice. In *16th Symposium on Computer Architecture and High Performance Computing*, pages 198–205, 2004.
  - [25] B. Nour-Omid, A. Raefsky, and G. Lyzenga. Solving finite element equations on concurrent computers. 1987.
  - [26] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, May 1990.
  - [27] A. Pothen, H. D. Simon, L. Wang, and S. T. Barnard. Towards a fast implementation of spectral nested dissection. In *Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, Supercomputing ’92, page 42–51, Washington, DC, USA, 1992. IEEE Computer Society Press.

- 
- [28] R. Preis. Linear time  $1/2$ -approximation algorithm for maximum weighted matching in general graphs. In *Proceedings of the 16th Annual Conference on Theoretical Aspects of Computer Science*, STACS'99, page 259–269, Berlin, Heidelberg, 1999. Springer-Verlag.
  - [29] P. Raghavan. Line and plane separators. Technical report, LAPACK WORKING NOTE 63 (UT CS-93-202), 1993.
  - [30] S. Schamberger. Improvements to the helpful-set algorithm and a new evaluation scheme for graph-partitioners. In V. Kumar, M. L. Gavrilova, C. J. K. Tan, and P. L'Ecuyer, editors, *Computational Science and Its Applications — ICCSA 2003*, pages 49–53, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
  - [31] R. Sedgewick and K. Wayne. *Algorithms, 4th Edition*. Addison-Wesley, 2011.
  - [32] C. Walshaw and M. Cross. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22, 07 2004.
  - [33] Wikipedia. Graph embedding — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Graph%20embedding&oldid=1019397582>, 2021. [Online; accessed 28-June-2021].
  - [34] Wikipedia. Skojarzenie (teoria grafów) — Wikipedia, the free encyclopedia. [http://pl.wikipedia.org/w/index.php?title=Skojarzenie%20\(teoria%20graf%C3%B3w\)&oldid=56877732](http://pl.wikipedia.org/w/index.php?title=Skojarzenie%20(teoria%20graf%C3%B3w)&oldid=56877732), 2021. [Online; accessed 30-June-2021].