

1. Przegląd literatury

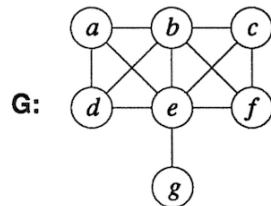
1.1. Istniejące metody partycjonowania grafów

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

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

1.1.1. 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 $G \setminus S$.



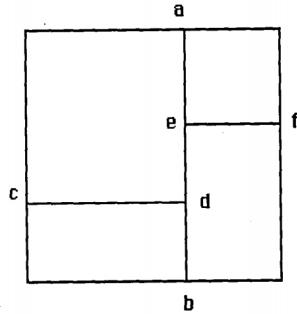
Rysunek 1: Separator wierzchołków d oraz c to $\{b, e\}$. Źródło: [22].

Metoda [26] będąca algorytmem spektralnym pokazuje algebraiczne podejście do obliczania separatorów wierzchołków. Artykuł [27] opisuje algorytm spectral nested dissection (SND). Algorytmy te 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 wszystko metody kosztowne z racji na obliczanie wektorowa własnego odpowiadającego drugiej najmniejszej wartości własne (Fielder wektor). Istnieją udane próby ulepszenia czasu wykonania tych metod, które polegają na liczeniu Fielder wektora poprzez algorytm wielopoziomowy - MSB [1]. Jednak nawet te metody wciąż charakteryzują się wysoką złożonością.

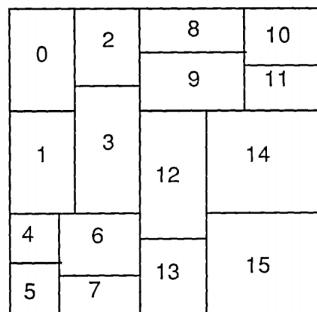
1.1.2. Metody rekursywne

Są to metody często prostsze w implementacji, jednak nie sprawdzają się tak dobrze w kontekście bardziej skomplikowanych problemów głównie ze względu na to, że mają ząchłanną naturę. Metoda [2] zakłada, że dzielimy siatkę na liczbę obszarów, która jest

równa potędze liczby dwa. Ta metoda 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 2 oraz 3. Specyfika tych metod polegająca na dzieleniu grafu na coraz mniejsze części sprawia, że nie jesteśmy w stanie uwzględnić części niepodzielnych, lub rozwiążanie tego problemu byłoby skomplikowane. Przykładem jest sytuacja kiedy dzielimy siatkę na 4 części. Można sobie wyobrazić sytuację, kiedy obszar niepodzielny zajmuje 50% całej siatki. Po pierwszej turze rekursywnego algorytmu mamy dwie partie, każda zajmująca 50% powierzchni. Chcielibyśmy podzielić każdą z nich na dwie części, natomiast nie jesteśmy w stanie tego zrobić ponieważ jedna z nich jest w całości obszarem niepodzielnym. Wykorzystanie takich algorytmów w kontekście niepodzielnych obszarów nie zdaje więc egzaminu.



Rysunek 2: Przedstawia 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 3: 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 na zmianę wykorzystując wertykalne i horyzontalne cięcie aż do otrzymania podziału na oczekiwany liczbę obszarów. Źródło: [2]

1.1.3. Metody geometryczne

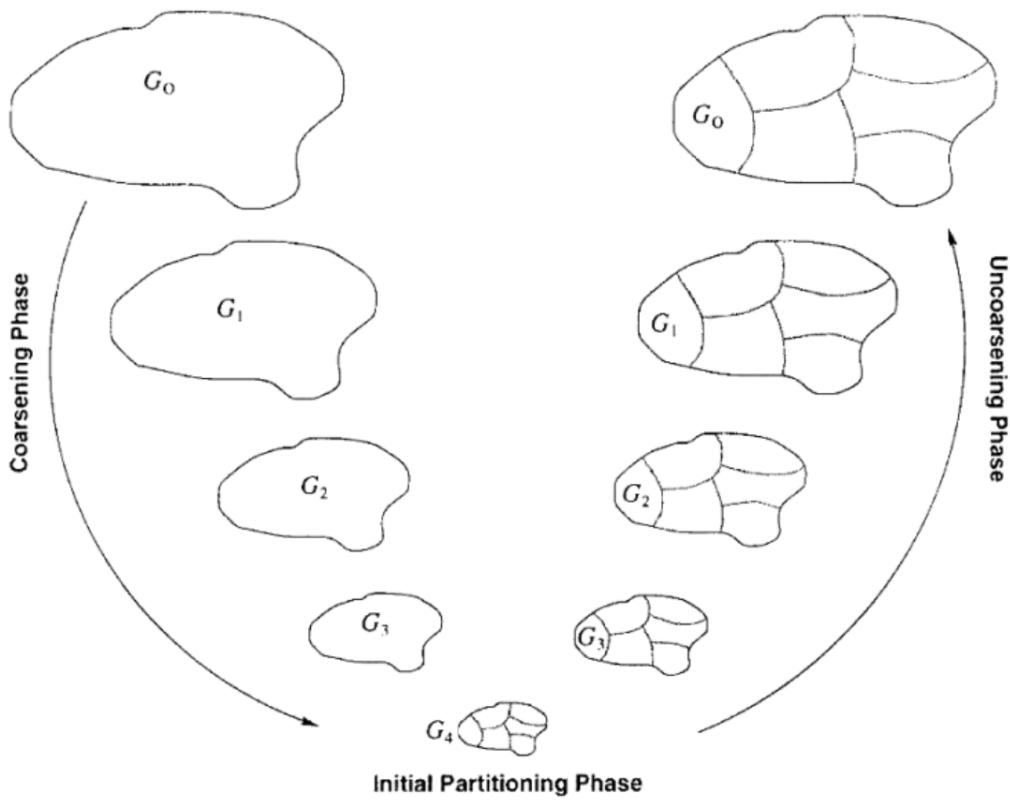
Inną klasą metod są metody geometryczne [20, 29, 21, 22, 25]. Używają danych geometrycznych o grafie w celu znalezienia dobrego partycjonowania. Ich cechą charakterystyczną jest szybki czas wykonania, natomiast gorsze rezultaty podziału niż metody spektralne. Najlepsze wyniki spośród wyżej wymienionych metod prezentują [21, 22]. W artykule [21] zaproponowane klasę grafów zwaną k -overlap graphs. Dla grafów k -overlap osadzonych w d wymiarach [32] 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 efektywną metodę partycjonowania siatek niestrukturalnych, która występują 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. Charakterystyką metod geometrycznych jest to, że z powodu losowej natury wymagane jest wielokrotne użycie algorytmu (od 5 do 50 razy) aby uzyskać wynik porównywalny z metodami spektralnymi. Wielokrotnie wywołanie zwiększa czas otrzymywania rezultatu, natomiast jest on wciąż niższy od metod spektralnych. Metody geometryczne są aplikowalne tylko w przypadku kiedy 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 ale są bardzo kosztowne i dominują czas potrzebny na samo 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 kontekście niniejszej pracy.



Rysunek 4: 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.

1.1.4. Metody wielopoziomowe

Istnieje wiele implementacji metod wielopoziomowych: [16, 31, 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 zachowując podział. Często graf zmniejszany jest aż liczba wierzchołków nie osiągnie liczby partycji, którą chcemy otrzymać [24], a fazie przywracania grafu do początkowej wielkości towarzyszy algorytm, którego celem jest ulepszanie podziału [6, 7]. Algorytm ten, bazując na zmniejszonym grafie, niesie za sobą niższy koszt obliczeniowy. Jego działanie polega na zmniejszaniu długości granic pomiędzy partycjami z jednoczesnym zachowaniem ich wielkości. Na tym etapie może także zostać dodana faza balansowania, która stopniowo zmniejsza różnice w wielkości pól pomiędzy obszarami. Metody te zostały stworzone z myślą o zmniejszeniu czasu patrycjonowania kosztem jego jakości. Obecnie dają jednak bardzo dobre rezultaty również w kwestii 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 [31], Chaco [11], dające state-of-the-art wyniki w kwestii jakości partycjonowania najczęściej bazują na schemacie wielopoziomowym [11].

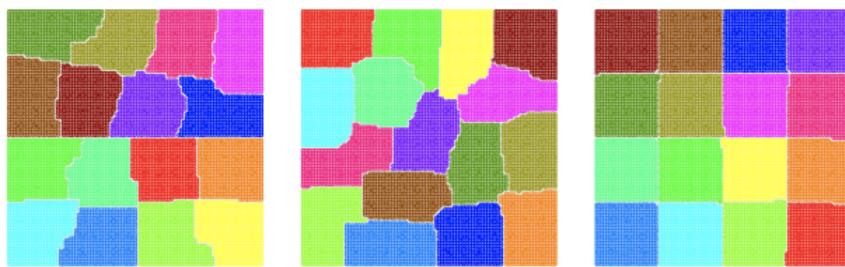


Rysunek 5: Wielopoziomowe partycjonowanie grafu przedstawiające fazę zmniejszania grafu, następnie przypisanie partycji na zmniejszonym grafie, na końcu przywrócenie grafu do początkowej wielkości. Źródło: [14].

1.2. 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 były najlepszym wyborem, z racji na to, że gwarantowały najlepsze wyniki partycjonowania. Przykładów metod wielopoziomowych było bardzo dużo [16, 31, 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 [31], Chaco [11].



Rysunek 6: Partycjonowanie siatki 100x100 na 16 obszarów. Od lewej - pmetis [16] używa edge-cut wynoszący 688, następnie Jostle [31] 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] używają random weighted matching, natomiast Party [24] stosuje LAM matching [28]. Ze względu na wymagania co do złożoności obliczeniowej 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 pozostałych metodach. Kiedy graf jest zmniejszony, wierzchołki są przyporządkowywane do partycji a informacja na temat partycji jest propagowana 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 przeciwieństwie do Metis i Jostle, faza ulepszenia 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 otrzymuje bardzo dobre wyniki stosując tę metodę [30], często uzyskując mniejszą długość granic 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 π 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 pozwalają na małe nierówności w kwestii wielkości obszarów co przekłada się na mniejsze długości granic, szczególnie na głębszych poziomach, kiedy przemieszczone 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 właściwością Party była znacznie niższa długość granic w porównaniu do pozostałych bibliotek (Rysunek 6), co było bardzo ważne dla mojej pracy. Dlatego to właśnie metodę z biblioteki Party zdecydowałem się wybrać jako podstawę rozwiązania prezentowanego w niniejszej pracy.

2. Szkielet rozdziału czwartego -

2.1. Ogólny opis algorytmu

W ramach algorytmu podziału siatki, 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.

2.1.1. Podział siatki na $m \cdot k$ obszarów

W ramach tego etapu 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 szumów.

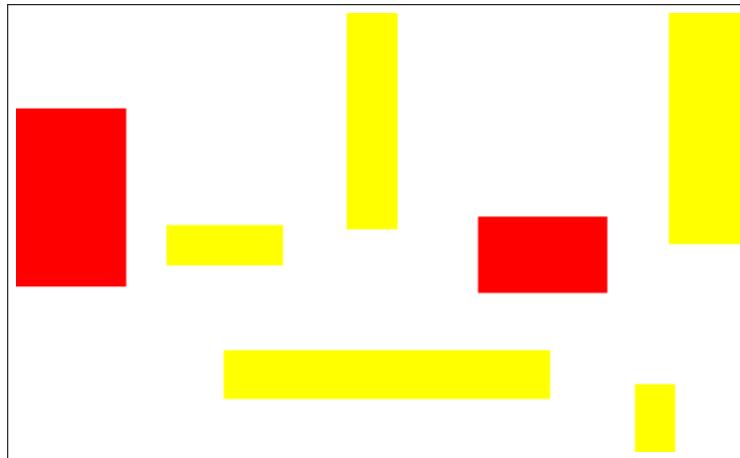
2.1.2. Podział siatki na m obszarów

W ramach tego etapu 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.

2.2. Podział siatki na $m \cdot k$ obszarów - dokładny opis

2.2.1. Mapowanie obrazka na graf



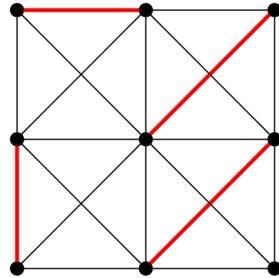
Rysunek 7: Obrazek, który reprezentuje strukturę siatki do podziału. Żółte obszary to obszary niepodzielne, czerwone to te wyłączone z obliczeń, białe to zwykłe przez które przechodzić będą granice podziału.

Stosunkowo krótki akapit o tym jak wygląda mapowanie. Obrazki powinny być w formacie png lub jpg. Ten etap polega na przeiterowaniu po siatce w celu stworzenia grafu, który ją odwzorowuje. Jest to część algorytmu, której nie było w żadnym z paperów, więc została ona w pełni stworzona przeze mnie. Celem tej części algorytmu było stworzenie szybkiej metody tworzenia kolejnych scenariuszy testowych. (PYTANIA):

- Czy tu powiniensem się wgłębiać w szczegóły implementacyjne? Że jest to for który iteruje sobie po pikselach, sprawdzając z jakim kolorem ma styczność, następnie buduje na tej podstawie graf którego wszystkie wierzchołki i krawędzie mają wagę 1?
- że prócz tego do osobnych struktur danych dodawane są obszary niepodzielne i wyłączone z obliczeń?
- że ”parsowanie” obszarów niepodzielnych i wyłączonych z obliczeń odbywa się rekurencyjnie?
- Czy powiniensem pisać w jaki sposób trzymane są dane na temat grafu, obszarów niepodzielnych (jakie struktury danych) itp?
- czy powiniensem dodawać pseudokod? Wydaje mi się to zbyt oczywiste.

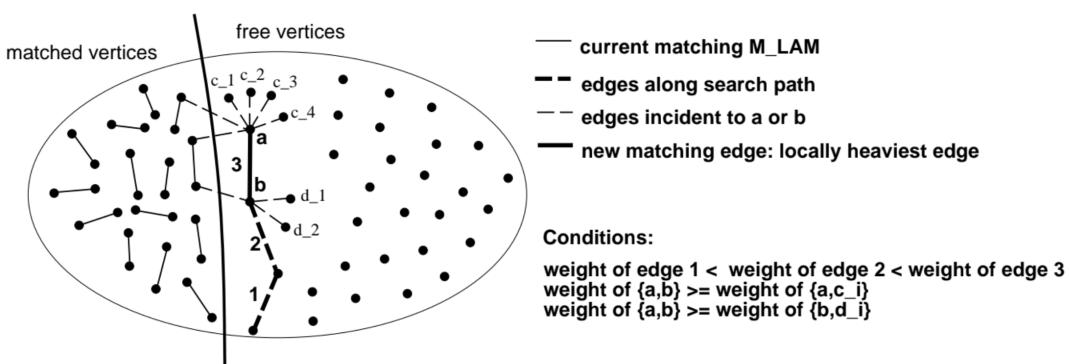
2.2.2. Zmniejszanie i partycjonowanie grafu za pomocą algorytmu LAM

[WSTĘP] Aby stworzyć mniejszy odpowiednik wejściowego grafu aplikowany jest algorytm tworzący skojarzenia (ang. matching). Skojarzenie to podzbiór krawędzi grafu (ozn. M) o tej własności, że każdy wierzchołek jest końcem co najwyżej jednej krawędzi z M . Pary wierzchołków połączone bezpośrednio krawędzią należącą do M są skojarzone przez M [33].



Rysunek 8: Skojarzenie największe, którego liczba krawędzi wynosi 4. Źródło: [33].

[INTUICJYJNE WYJAŚNIENIE ALGORYTMU LAM] W celu zmniejszenia grafu aplikowana jest heurystyka, która bazuje na metodzie [28]. Algorytm ten oblicza 2-approximation (jak to przetłumaczyć?) dla problemu maximum weighted matching w czasie liniowym. Algorytm ten startuje od arbitralnej krawędzi i sprawdza sąsiednie krawędzie. Tak, jak długo jest w stanie znaleźć sąsiednią krawędź z wyższą wagą, algorytm wywołuje się na niej i powtarza tę procedure aż znajdzie krawędź z lokalnie najwyższą wagą. Ta krawędź łączy dwa wierzchołki, które zostaną skojarzone. W ten sposób skonstruowany algorytm dąży do budowania obszarów, które są możliwe "zwarte", innymi słowy oznacza to, że granice między obszarami są możliwe krótkie oraz niepostrzępione. Krawędzie z wysokimi wagami to krawędzie między wierzchołkami, które reprezentują zbiory wierzchołków, które w początkowym grafie miały między sobą długą granicę.



Rysunek 9: Fragment przebiegu algorytmu LAM. Startując z 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 krawędzi sąsiadujące do $\{a, b\}$). Źródło: [28].

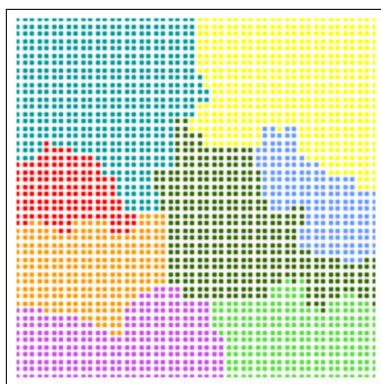
```

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  $U$ ;

```

Rysunek 10: Pseudokod przedstawiający zmodyfikowany przez mnie 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 pseudokod można znaleźć w artykule [28].

[SZCZEGÓŁOWY OPIS ALGORYTMU] Algorytm LAM został przedstawiony na rysunku 10. Jest to wersja przeze mnie zmodyfikowana. Startuje z pustym zbiorem skojarzeń M_{LAM} . Zbiór U przechowuje nieodwiedzone krawędzie ($U = E$ na starcie). 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 poszukiwaniu 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 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 rekursively. Rekursywne wywołania są powtarzane aż do znalezienia krawędzi z lokalnie najwyższą wagą. Następnie krawędź dodawana jest do M_{LAM} , a algorytm zakończa aktualne wywołanie 'try match' i kontynuuje pętle WHILE sprawdzając kolejne sąsiadujące krawędzie. Pętla WHILE zakończa 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 . 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.

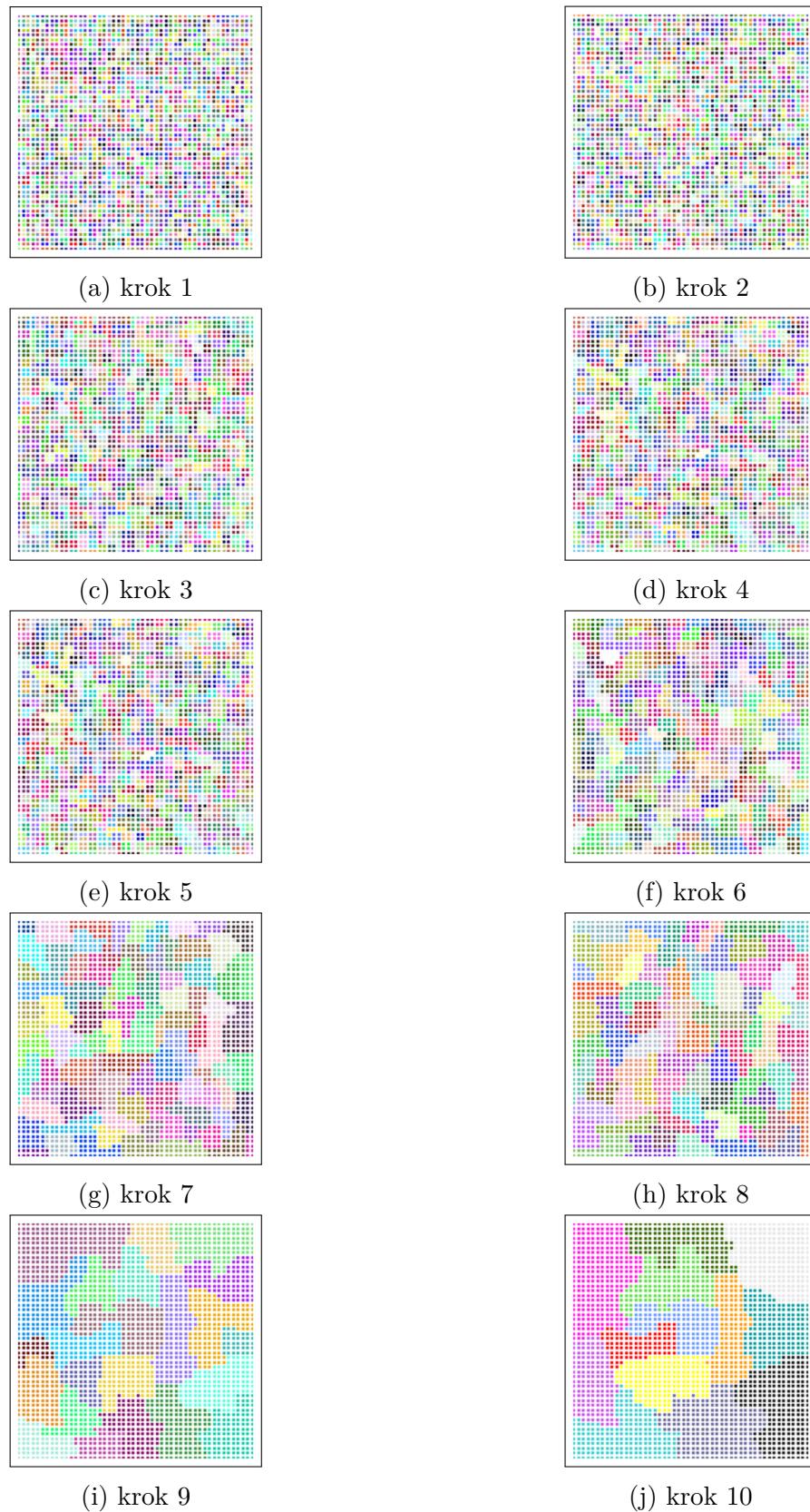


(a) partycjonowanie 1



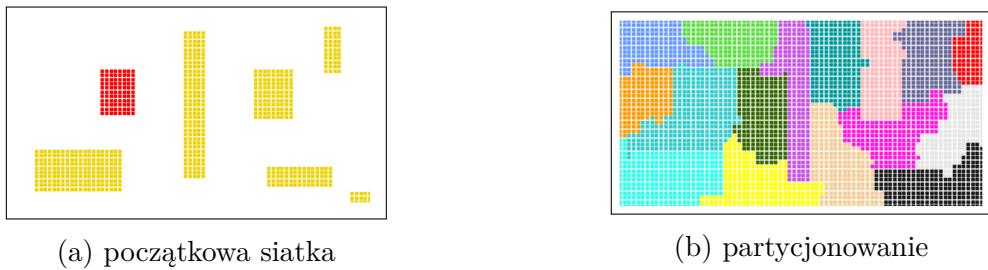
(b) partycjonowanie 2

Rysunek 11: Dwa losowo wybrane partycjonowania siatki 50x50 na 8 obszarów za pomocą samego algorytmu LAM - bez fazy ulepszania podziału. Granice podziałów nie są optymalne, pola obszarów nie są równe - podział jest jednak bliski bycia równym w kwestii pól co jest oczekiwana i dobrą bazą do dalszych operacji. Algorytm ma charakterystykę losową, to znaczy, że każde wywołanie będzie dawać inny rezultat. Zawsze jednak zachowane będzie dążenie do równych i zwartych obszarów.



Rysunek 12: Obrazki przedstawiają wybrane, kolejne kroki algorytmu LAM aż do otrzymania podziału siatki 50×50 na 13 obszarów. Nie są to wszystkie kroki. Rysowane było co siódme wywołanie algorytmu oraz efekt końcowy. Widoczne jest równomierne łączenie obszarów.

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 kiedy liczba wierzchołków w grafie nie osiągnie liczby 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 pierwsze n par wierzchołków, tak by otrzymać wymaganą liczbę wierzchołków. Waga wierzchołka powstałego 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.



Rysunek 13: Obrazek (b) przedstawia partycjonowanie siatki (a) samym algorytmem LAM. Widoczne jest uwzględnianie obszarów niepodzielnych, które na siatce (a) zaznaczone są jako obszary żółte.

2.2.3. Modyfikacje algorytmu LAM

[O MODYFIKACJACH TEGO ALGORYTMU PRZEPROWADZONYCH PRZEZ AUTORÓW PARTY] Algorytm ten bez odpowiednich modyfikacji zwykle tworzyłby grafy typu gwiazda, co oznacza, że powstawałaby część wierzchołków o bardzo wysokim stopniu. To powoduje, że zmniejszony graf nie przypomina początkowego grafu. W celu rozwiązania tego problemu autorzy [28] zaproponowali modyfikację polegającą na dodatkowym warunku dla skojarzenia wierzchołków. Skojarzenie wierzchołka a z wierzchołkiem b może zostać stworzone tylko wtedy, jeśli ich sumaryczna waga nie przekracza dwukrotności najmniejszej wagi wierzchołka w grafie zsumowanej z największą wagą wierzchołka w grafie.

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

Ten warunek powoduje, że nawet jeśli rozpatrujemy skojarzenie wierzchołka o relatywnie wysokiej wadze, to ma on szansę zostać skojarzony jedynie z wierzchołkiem o wadze niskiej, co prowadzi do równiejszych wag w grafie. Przy dalszych etapach zmniejszania grafu trudniej spełnić warunek na skojarzenie wierzchołków - pojawia się więcej różnorodności

w kwestii wag wierzchołków grafu, spada więc liczba udanych skojarzeń przypadających na wywołanie. W momencie, kiedy zbyt mało par jest kojarzonych podczas pojedynczego wywołania, warunek ten jest osłabiany poprzez zwiększenie dopuszczalnej sumarycznej wagi wierzchołka a oraz b .

-[O MODYFIKACJACH TEGO ALGORYTMU PRZEPROWADZONYCH PRZEZE MNIE] Warunek stworzony przez autorów artykułu [28] okazał się nie działać w kontekście obszarów niepodzielnych. W tym celu stworzony został nowy, zmodyfikowany warunek.

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

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

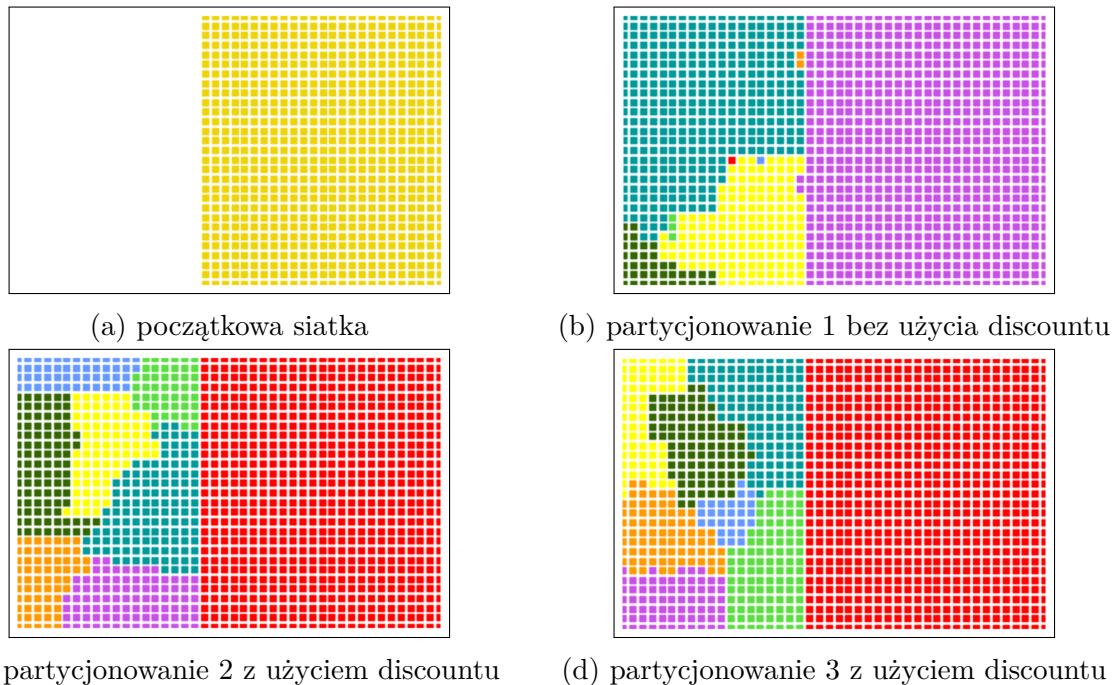
$$w_a + w_b \leq \text{discount} \cdot (w_{\text{highest}} + 2 \cdot w_{\text{lowest}}) \quad (3)$$

[DLACZEGO STARY WARUNEK NIE DZIAŁA] Niedziałanie poprzedniego warunku, jest spowodowane faktem, że w początkowej fazie algorytmu, tuż po zamianie wejściowej siatki na graf, wszystkie obszary niepodzielne oraz obszary wyłączone z obliczeń zamieniane są na pojedyncze wierzchołki. Pojedyncze wierzchołki reprezentujące obszary wyłączone z obliczeń mają zawsze wagę równą zero, natomiast waga dla wierzchołków reprezentujących obszary niepodzielne jest równa sumie wag wierzchołków, które do nich należą. Oryginalny algorytm LAM został zaprojektowany z założeniem o równomiernym budowaniu skojarzeń ze względu na wagi 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 wedle niego zmniejszany równomiernie od samego początku, kiedy wagi wszystkich wierzchołków w grafie mają taką samą wartość 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 14) niemal każde skojarzenie dwóch wierzchołków, z których żaden z nich nie jest wierzchołkiem reprezentującym obszar niepodzielny, będzie dozwolone. Każdy z takich wierzchołków ma bowiem wagę mniejszą niż suma podwojonej najmniejszej wagi wierzchołka w grafie oraz największej wagi wierzchołka w grafie, a więc warunek 1 jest niemal zawsze spełniony. W tego typu przypadkach realny podział odbywa się tak naprawdę poza obszarem niepodzielnym, ponieważ wiemy, że obszar niepodzienny będzie na końcu jedną partycją, a jako że jest największym wierzchołkiem w grafie - nie będzie mógł tworzyć wielu skojarzeń. Prowadzi to do tego, że obszary kojarzone są w sposób niezachowujący równomiernych pól jak na obrazku (b) na rysunku 14. W efekcie w skład wielu partycji wchodzi tylko jeden wierzchołek.

[DLACZEGO NOWY WARUNEK DZIAŁA] Zaproponowana przeze mnie modyfikacja (wzór 2), polegająca na dodaniu *discountu*, wzmacnia warunek i powoduje bardziej równomierne (ze względu na wagę) 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 UBOCZNY NOWEGO WARUNKU] Efektem ubocznym stosowania *discountu* jest większa liczba wywołań algorytmu LAM - czasami zmienna t musi urosnąć do konkretnej wartości, umożliwiającej dokonanie kolejnych, blokowanych obecnie skojarzeń, a żadne lepsze skojarzenia nie są w danej chwili możliwe. Rysunek 14 pokazuje pozytywne działanie modyfikacji na jakość podziału, wraz z porównaniem do poprzedniej wersji warunku.

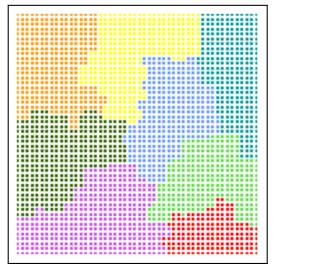
[JAK POWSTAŁ NOWY WARUNEK] Skonstruowanie *discountu* polegało na odpowiednim dobraniu jego wartości względem parametrów dzielonej siatki, tak by nie działał zbyt agresywnie, nadmiernie przez to przedłużając obliczenia, ale jednocześnie był na tyle agresywny, aby pozwalał tylko na kojarzenie wierzchołków prowadzące do możliwie równych podziałów. Został skonstruowany na bazie podstawowego współczynnika $\frac{t}{T}$, który na podstawie wielu prób dostosowałem dodatkowymi zmiennymi w poszukiwaniu wcześniej określonego balansu. W momencie, w którym 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.



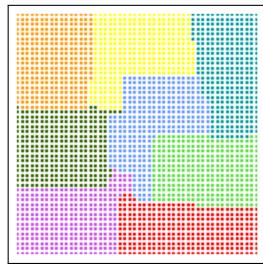
Rysunek 14: Obrazek (a) przedstawia siatkę wejściową, której więcej niż połowę zajmuje oznaczony kolorem żółtym obszar niepodzielny. Obrazek (a), (b) oraz (c) przedstawia partycjonowanie siatki na 8 części wykonane algorytmem LAM bez fazy ulepszania podziału. Obrazek (b) pokazuje partycjonowanie wykonane z użyciem warunku 1 na skojarzenie wierzchołków. Obraz (c) oraz (d) prezentują partycjonowanie wykonane z użyciem warunku 3.

2.3. Przywracanie grafu do początkowego rozmiaru wraz z ulepszeniem podziału

[WSTĘP] 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).



(a) po algorytmie LAM



(b) po fazie ulepszania

Rysunek 15: 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.

[INTUICJA NA TEMAT ALGORYTMU ULEPSZANIA PODZIAŁU] Etap pierwszy to heurystyka do poprawiania podziału. Jest oparta na wyszukiwaniu lokalnym - próbuje poprawić aktualny podział poprzez wielokrotne wprowadzanie lokalnych zmian. Wierzchołki klasyfikowane są pod kątem liczby krawędzi 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 faza szukania zbioru balansującego D . Jest budowany na zbiorze B . Jego celem jest zwrócenie zbioru wierzchołków z partycji B do partycji A o tej samej wielkości jak zbiór C z jednoczesnym możliwie utrzymanym ulepszeniem 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 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 w formie pojedynczych wierzchołków o dużych wagach efektywnie przenoszone są duże obszary. 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 skupić się na bardziej lokalnych poprawkach.

2.3.1. Używanie Helpful Sets w celu zmniejszenia długości granic

-[DOKŁADNY OPIS HELPFULSETS] Heurystyka HS startuje od początkowego podziału π i za pomocą lokalnie wprowadzanych zmian zmniejsza długość granic. 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, powiedzmy że 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 18. Zanim opiszę algorytm dokładniej, potrzebne są definicje zbiorów helpful oraz balancing:

Definition 1 (Helpful Set)

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

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

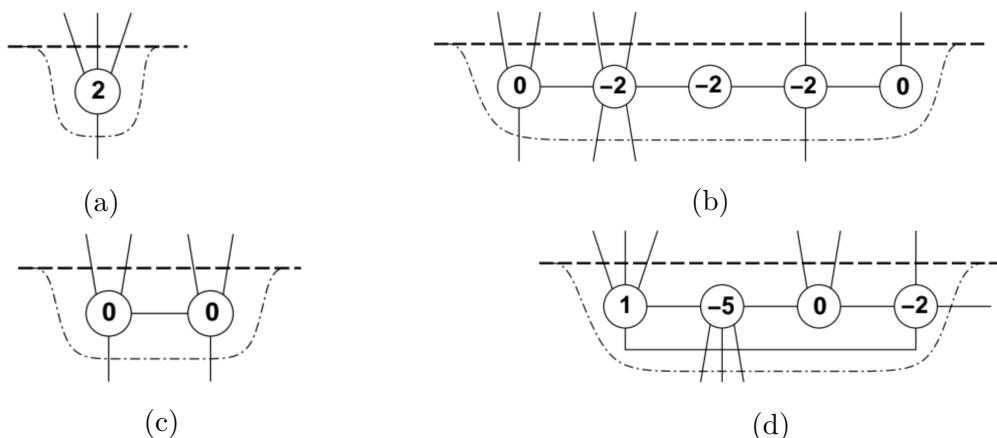
to **helpfulness** zbioru S . Zbiór S nazywany jest $H(S)$ -**helpful**.

Przeniesienie zbioru S do drugiej partycji zmniejsza długość granicy o $H(S)$. Przykłady zbiorów 2-helpful pokazane są na rysunku 16.

Definition 2 (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.

Ważną cechą algorytmu jest, że długość granicy zwiększa się o nie więcej niż $(k - 1)$ jeśli \bar{S} przenoszona jest z jednej części do drugiej.



Rysunek 16: 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): $6 - 8 + 4 = 2$. Źródło: [6].

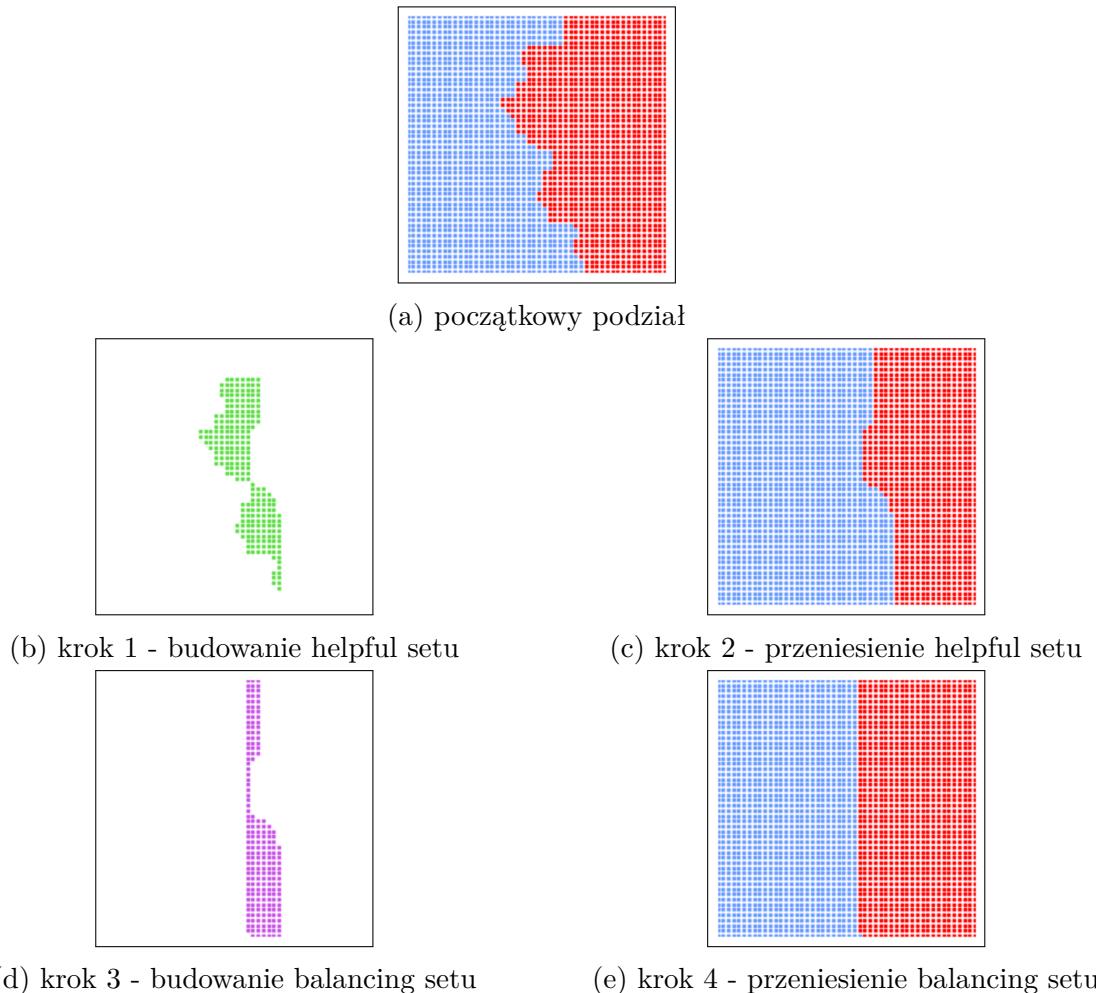
```

1 HelpfulSet( $A, B$ )
2    $l_A \leftarrow l_B \leftarrow cut/2;$ 
3    $s_{max} = (|A| + |B|)/2) \cdot 0.2;$ 
4
5   IF  $cut\_size_{A-B} < 0$ 
6     RETURN;
7   ENDIF
8   WHILE  $l_A + l_B \geq 1$ 
9     IF  $l_A = 0$  OR  $2 \cdot l_A \leq l_B$ 
10       Swap( $A, B$ );
11        $S_A = \text{BuildHS}(A, l_A, -d/2, s_{max});$ 
12     ENDIF
13     IF  $h(S_A) \leq l_A$ 
14        $l_A \leftarrow b(S_A);$ 
15     IF  $l_B > h(S_A)$ 
16        $S_B = \text{BuildHS}(B, l_B, -d/2, s_{max});$ 
17       IF  $h(S_B) \geq h(S_A)$ 
18         Swap( $A, B$ );
19       ELSE
20          $l_B \leftarrow b(S_B);$ 
21       ENDIF
22     ENDIF
23     UndoBuild( $S_B$ );
24   ENDIF
25   IF  $h(S_A) < 0$ 
26      $l_A \leftarrow b(S_A);$ 
27     CONTINUE;
28   ENDIF
29    $l_A \leftarrow \min(l_A, h(S_A));$ 
30   MoveSet( $S_A$ )
31    $min \leftarrow (w(B) - w_{max}(B) - grace)^+;$ 
32    $max \leftarrow (w(B) - w_{min}(B) + grace)^+;$ 
33    $S_B = \text{BuildBS}(B, 1 - h(S_A), min, max);$ 
34   IF  $w_l \leq w(S_B) \leq w_h$  and  $h(S_B) > -h(S_A)$ 
35     MoveSet( $S_B$ );
36      $l_A \leftarrow l_A + l_A;$ 
37      $l_B \leftarrow l_B + 1;$ 
38   ELSE
39     UndoBuild( $S_B$ );
40     UndoMove( $S_A$ );
41      $l_A \leftarrow l_A/4;$ 
42      $l_B \leftarrow l_B/2;$ 
43   ENDIF
44 ENDWHILE
45 IF  $cut\_size_{A-B} > 0.1 \cdot longer\_edge\_of\_a\_grid$ 
46   Balance( $A, B, grace$ );
47 ENDIF

```

Rysunek 17: Pseudokod przedstawiający zmodyfikowany przez mnie algorytm Helpful-Set 2-partitioning.

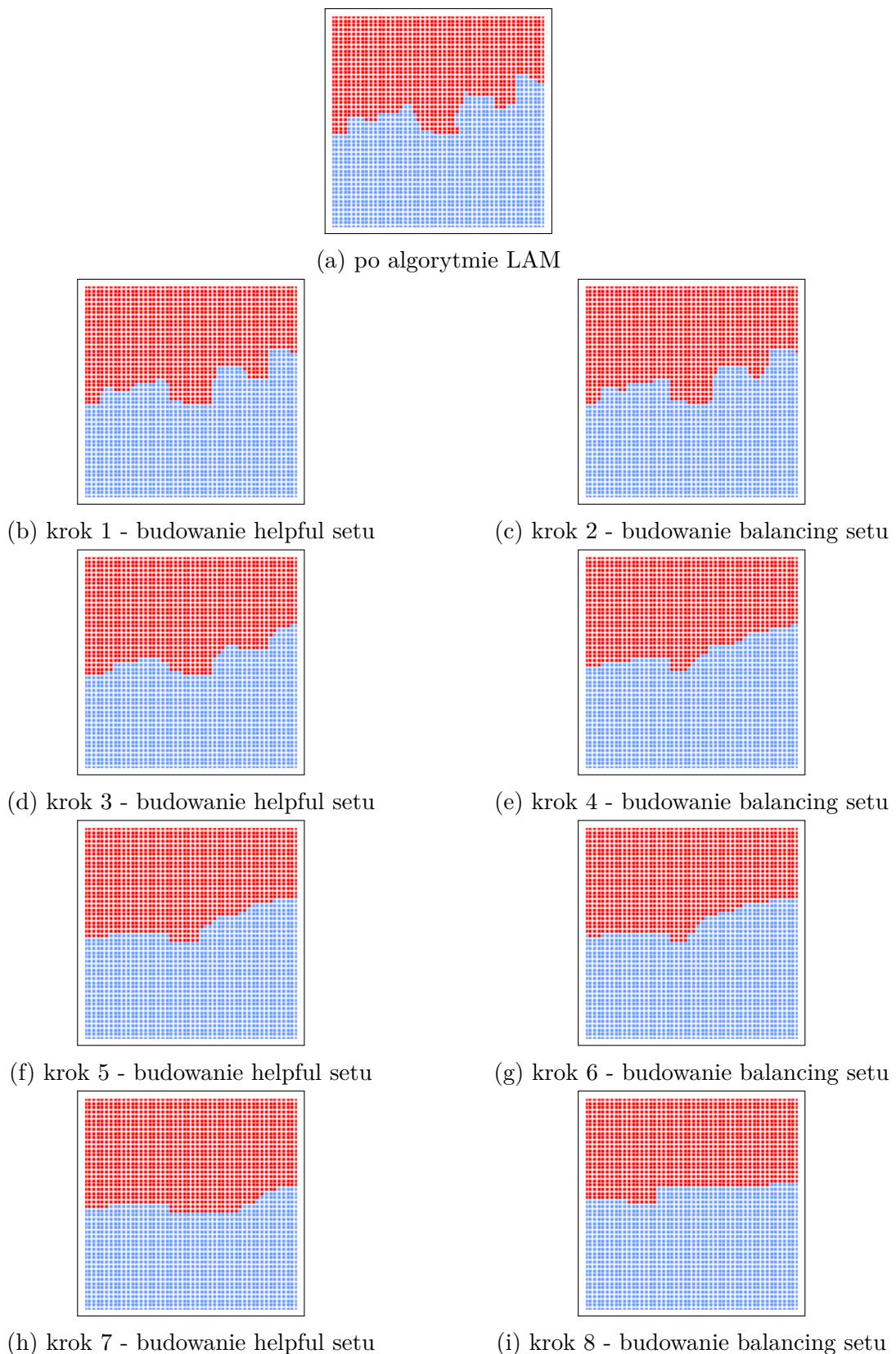
Okazało się, że kluczowym elementem algorytmu jest ustalenie wartości *helpfulness* l zbiorów, których szukamy. Jeśli l jest ustawione na zbyt małe to obiecujące zbiory są przeoczane, natomiast jeśli l jest za duże to czas wykonywania algorytmu jest dłuższy, natomiast znalezione przy tym zbiory wcale nie są lepsze. Party używa techniki zwanej **adaptive limitation** [6] w celu uzyskania l . l początkowe ustawiane jest na wartość $cut/2$. Wartość l jest zmniejszana o połowę albo ustawiana na najlepszą znalezioną *helpfulness* jeśli zbiór l -helpful nie może zostać znaleziony w żadnym z dwóch optymalizowanych zbiorów oraz podwajany 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, dopuszczono do pojawiania się małych nierówności w kwestii wielkości obszarów. Jest to szczególnie ważne, kiedy podział optymalizowany jest wielokrotnie, na całkowicie nieprzywróconym do początkowego rozmiaru grafie. Przenoszone wówczas wierzchołki mają często wagi ≥ 1 co przekłada się na to, że ciężko idealnie zbalansować optymalizowane obszary. To podejście zostało również zaimplementowane w bibliotekach Jostle and Metis. Udowodniono, że pozytywnie wpływa ona na optymalizacje długości granic, ponieważ małe nierówności w postaci pól pozwalają zwykle na lepsze zoptymalizowanie długości granic.



Rysunek 18: Obrazki pokazują początkowy podział, następnie obydwa etapu algorytmu Helpful Sets. Rozmiar partycji pozostaje identyczny, zmniejsza się długość granicy.

Po trzecie algorytm balansowania obszarów, który zachłannie wyrównuje pola obszarów został przeniesiony na koniec. Rysunek 17 przedstawia zmodyfikowany przeze mnie algorytm. Jak w poprzedniej implementacji limity l_a oraz l_b ustawiane są na połowę aktualnej długości granicy pomiędzy obszarami A i B . Jeśli limit l_a jest dużo mniejszy od limitu l_b zbiory A i B są ze sobą zamieniane. Ten warunek powoduje, że zbiór helpful szukany jest najpierw w lepiej obiecującej partycji. Podczas wyszukiwania zbiór helpful nie może stać się mniej niż $-d/2$ helpful, nie może również przekroczyć wartości s_{max} . Autorzy artykułu opisali jako średni stopień wierzchołka w grafie. Dla mojej implementacji założyłem, że d jest zawsze równe 4 - wynika to ze sposobu, w jaki buduje graf z wejściowej siatki. s_{max} z kolei, autorzy ustawiali na wartość 128. Ja uzależniłem s_{max} od rozmiaru optymalizowanych partycji. Użyty do tego współczynnik 0.2 może zostać dowolnie zmieniony. Jednak wedle mojego doświadczenia powinien być on raczej mniejszy niż 0.4, aby zbiór helpful nie był niepotrzebnie zbyt duży. Zwiększa to czas partycjonowania, ale nie poprawia wyników. Lepiej żeby był on nieco za mały niż za duży, ponieważ po udanym 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.

Jeśli zbiór l_A -helpful (S_A) (z wagą $w(S_A)$ oraz wartością helpfulness wynoszącą $h(S_A)$) nie może zostać znaleziony, limit jest redukowany do najlepszej znalezionej wartości helpfulness ($b(S_A)$). Dalej podejmowana jest decyzja czy nastąpi wyszukiwanie w partycji B . Jeśli tak, to ustalany jest zbiór S_b . Dalej zbiór z większą wartością helpfulness jest nazywany S_a , natomiast limit drugiej partycji jest redukowany a zbiór usuwany. Jeśli wartość helpfulness jest mniejsza niż 0, limity są ustawiane a procedura jest powtarzana. W przeciwnym wypadku do l_a przypisywana jest nowa wartość, S_a przenoszone jest do B i wywołanie wkracza w fazę szukania zbioru balancing.



Rysunek 19: Obrazki przedstawiają kolejne kroki działania algorytmu Helpful Sets na siatce 50x50 podzielonej na 2 obszary przez algorytm LAM. W 1 kroku tworzony jest stosunkowo mały helpful-set, a w kolejny krok odpowidnio mały balancing set. W momencie, kiedy to się udaje algorytm zwiększa limit wierzchołków, dlatego dwa kolejne kroki wnoszą więcej zmian. Po 4 cyklach budowania helpful-setu oraz balancing-setu granica między obszarami jest znaczaco mniejsza.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

 Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

 Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

 Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

 Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetur.

 Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit

erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

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] C. Walshaw and M. Cross. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22, 07 2004.
- [32] 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].
- [33] 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].