PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
Ministry of Higher Education and Scientific Research
MOHAMED KHIDER UNIVERSITY - BISKRA
Faculty of Exacts Sciences, Nature and Life Sciences
Department of Computer Science

THESIS

To obtain the academic degree of

Doctor in Computer Sciences LMD 3rd Cycle
Option : Techniques of Image and Artificial Intelligence

Parallelization of Morphological Operators Based on Graphs

Presented by

Imane YOUKANA

Members of the jury :

- Pr. Okba KAZAR	University of Biskra	President
- Dr. Rachida SAOULI	University of Biskra	Director of thesis
- Pr. Mohamed AKIL	ESIEE-Paris, France	Co-Director of thesis
- Dr. Laid KAHLOUL	University of Biskra	Examinator
- Pr. Noureddine ZERHOUNI	ENSMM-Besançon, France	Examinator
- Dr. Jean COUSTY	ESIEE-Paris, France	Invited

REPUBLIQUE ALGERIÉNNE DÉMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
University Mohamed Khider - BISKRA
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'informatique

THÈSE

en vue d'obtenir le titre de

Docteur en Informatique 3éme cycle LMD
Techniques de l'image et de l'intelligence artificielle

La parallélisation des opérateurs morphologiques basés sur les graphes

---

Présentée par

Imane YOUKANA

Devant le jury composé de:

- Pr. Okba KAZAR	Université de Biskra	Président
- Dr. Rachida SAOULI	Université de Biskra	Directeur de thèse
- Pr. Mohamed AKIL	ESIEE-Paris, France	Co-Directeur de thèse
- Dr. Laid KAHLOUL	Université of Biskra	Examinateur
- Pr. Noureddine ZERHOUNI	ENSMM-Besançon, France	Examinateur
- Dr. Jean COUSTY	ESIEE-Paris, France	Invité

This thesis is dedicated to:

My mother and my father,
without them I would never have seen the light of this day,

My husband Boubakeur,

My sister Ihsane,

My brothers Hamza and Abderhman,

My sun Adem,

All my family, my step family and all my friends.

Acknowledgments

I am glad to express my very sincere gratitude to my thesis directors Dr. Rachida Saouli, Pr. Mohamed Akil and Dr. Jean Cousty for their time, patience, consideration, and support by thoughts and motivation. I could not have imagined having a better advisors and mentors for my Ph.D study.

I am very grateful to my supervisor Dr. Rachida SAOULI, who offered her continuous advice, encouragement and love given to me as a mother throughout the course of this thesis. I would never have been able to finish my dissertation without the guidance of her.

I am very thankful to Pr. Mohamed AKIL because he opened the doors for me to ESIEE Paris and for accepting me into his group. I would like to thank him for his patience, advises and guidance.

I would like to express my deepest sense of gratitude to Dr. Jean COUSTY for their tremendously kindness, insight, endless support. Without their everyday effort in enhancing my work the quality of my thesis could not have been like this. I thank him for the systematic guidance and great effort he put into training me in the scientific field.

I thank my colleagues and the other members of the A3SI and LIGM teams at ESIEE Paris for their friendly welcome and nice working environment. My internship in ESIEE was very inspiring and fruitful.

My hearty thanks go to the members of my jury: Pr. Okba KAZAR, Pr. Noureddine ZERHOUNI, and Dr. Laid KAHLOUL for their time necessary to read and understand the manuscript.

We also warmly thank the consortium of the RECOVER3D project funded by the French 1st *Programme d'Investissements d'Avenir (PIA)* including, in particular, Ludovic Blache and Laurent Lucas, for providing us with the 3D textured meshes processed in our work presented in this manuscript.

Finally, the most eminent thanks goes to my parents who were unconditionally encouraging me during the entire university study to achieve the goals I can be proud of. I would also thank my husband, my sister and my brothers for their encouragement and patience with me, also to all my family and everyone who made this thesis possible.

Abstract

Mathematical morphology is one of the most powerful frameworks which provides a set of filtering and segmenting tools that are very useful in applications to image analysis. The first field of applications of mathematical morphology was binary image by Matheron and Serra in 1964 [54]. The theory of mathematical morphology is based on that the underlying image space is a complete lattice [28] allowing to consider the processing of a very broad class of data with mathematical morphology operators. On the other hand, considering digital objects carrying structural information, mathematical morphology has been developed on graphs, simplicial complexes, and on hypergraphs. This thesis report is focused on the framework of mathematical morphology on graphs spaces presented in [14]. This framework considers operators whose input and output are both graphs (sets of vertices as well as sets of edges). The basic operators go from one kind of sets to another one. They can be combined in order to obtain operators acting on the subset of edges, on the subset of vertices, and on the subgraphs of a given graph.

The main objective is to provide efficient computation and implementation of these morphological operators on graphs. To this end, we study the (unweighted) graph-based mathematical morphology operators. These operators depend on a size parameter that specifies the number of iterations of elementary dilations/erosions. Thus, the associated running times of these iterated operators increase with the size parameter, the algorithm running in $O(\lambda.n)$ time, where n is the size of the underlying graph and λ is the size parameter. In our work, we are focused in distance maps that allow us to recover (by thresholding) all considered dilations and erosions, hence all the operators proposed in [14].

In the first part, we propose three new distance maps on graphs called edge-edge, edge-vertex, and vertex-edge distance maps. Furthermore, based on new notion of path which considers both the numbers of edges and of vertices along the path, we present our vertex-vertex distance map. We show that these distance maps lead to original characterization of all operators presented in [14]. Then, a linear-time sequential algorithms for distance maps in unweighted graphs, hence the operators of dilations/erosions on graphs is proposed. In fact, any dilation, erosion, opening and closing on graphs can be obtained with a single iteration by thresholding these distance maps. Therefore, these operators can be computed in linear time with respect to the size of the graph, with a single iteration and without any

dependence to this size parameter.

In the second part, we investigate a parallelization strategy on multi-core architecture leading to efficiently compute our proposed distance maps, hence the morphological operators on graphs. The proposed strategy consists of building iteratively the successive level-sets of the distance maps, each level set being traversed in parallel. Indeed, to state the time complexity of our parallel strategy we make assumptions about the graph and the sets under consideration. Under these assumptions, our parallel algorithms run in $O(n/p + K \log_2 p)$ where n, p, and K are the size of the graph, the number of available processors, and the number of distinct level-sets of the distance map, respectively.

In the third part, a description of the 2D image and 3-dimensional meshes datasets used for experimental evaluations is provided. Then, we assess the regularity of the proposed assumptions on these experimental datasets. And, we perform an analysis of the results obtained by applying the implementations of the proposed sequential and parallel algorithms on the target architectures. This evaluation shows a significant improvement of the processing time over the previously available implementations.

Keywords:

(unweighted) Graph-based mathematical morphology, distance maps, parallelization strategy, multicore architecture .

Résumé

La morphologie mathématique est un cadre puissant qui fournit un ensemble d'outils de filtrage et de segmentation trés utiles dans les applications d'analyse d'image. Initialement le premier champ d'applications de la morphologie mathématique était sur les images binaires et proposé par Matheron et Serra en 1964 [54]. En outre la théorie de la morphologie mathématique repose sur le fait que l'espace de l'image est un treillis complet [28] ce qui permet d'envisager le traitement d'une trés large classe de données avec les opérateurs de la morphologie mathématique. En prenant en considération les objets numériques portant des informations structurelles, la morphologie mathématique a été développée sur des graphes, des complexes simpliciaux, et sur les hypergraphes. Ainsi le travail proposé dans cette thèse se focalise sur le cadre de la morphologie mathématique basée sur les graphes présenté dans [14]. Dans ce cadre les ensembles d'entrées et sorties des opérateurs considérés sont des graphes (ensembles de sommets ainsi que des ensembles d'arêtes) et les opérateurs de base peuvent aller d'un type d'ensembles vers l'autre. Ils peuvent être conbinés afin d'obtenir des opérateurs agissant sur le sous ensemble d'arêtes, sur le sous-ensemble des sommets, et sur les sous-graphes d'un graphe donné.

L'objectif principal dans cette thèse est de fournir un calcul et une implémentation efficaces pour ces opérateurs morphologiques en se basant sur les graphes non pondérés. En effet, ces opérateurs dépendent d'un paramètre de taille qui spécifie le nombre d'itérations de dilatations et d'érosions élémentaires. Ainsi, le temps d'exécution associé à ces opérateurs itératifs augmente avec le paramètre de taille et la complexité est de l'ordre de $O(\lambda.n)$, où n est la taille du graphe et λ le paramètre de taille. Dans notre travail, nous sommes particulièrement intéressés par les cartes de distances qui nous ont permis (par seuillage) de caractériser et d'effectuer toutes les dilatations/érosions considérées et par conséquent tous les opérateurs proposés dans [14].

En premier lieu nous avons proposés trois nouvelles cartes de distance basées sur les graphes. Il s'agit des cartes de distance arête-sommet, sommet-arête et arête-arête. En plus, basé sur une nouvelle notion de chemin qui considère à la fois le nombre d'arêtes et de sommets sur le chemin, nous présentons une carte de distance sommet-sommet. Nous montrons que ces nouvelles cartes de distance mènent à des caractérisations originales de toutes les dilatations et érosions présentées dans [14]. Ensuite, des algorithmes sèquentiels en temps linèaire ont été proposés afin de calculer ces nouvelles cartes de distance dans

les graphes et par conséquent les opérateurs de dilatations/érosions basés sur les graphes. Toute dilatation, érosion, ouverture et fermeture sur les graphes peuvent être obtenues avec une seule itération par le seuillage de ces cartes de distance. Ainsi, ces opérateurs peuvent être calculés en temps linéaire par rapport à la taille du graphe avec une seule itération et sans aucune dépendance avec le paramètre de taille.

Afin de calculer efficacement les cartes de distance proposées et les opérateurs morphologiques basés sur les graphes, nous avons développé une stratégie parallèle sur une architecture multi-coeurs à mémoire partagée qui consiste à construire d'une manière itérative les niveaux successifs des cartes de distance avec un parcours en parallèle de chaque niveau. Afin d'estimer la complexité de notre stratégie parallèle, nous proposons et démontrons des hypothèses sur le graphe et les ensembles considérés. Selon ces hypothèses, la complexité de nos algorithmes parallèles est $O(n/p + Klog_2p)$ où n, p, et K sont respectivement la taille du graphe, le nombre de processeurs disponibles et le nombre de niveaux distincts de la carte de distance, respectivement.

Dans l'évaluation expérimentale, nous avons fourni une description des bases de données utilisées ; Il s'agit de deux bases d'images 2D et une base des 3D maillages triangulaires texturées. Nous avons évalué la régularité des hypothèses proposées sur les bases de données considérées pour effectuer ensuite une analyse des résultats obtenus par les implémentations séquentiels et parallèle des algorithmes proposés sur les architectures cibles. Cette évaluation montre une amélioration significative en termes de temps d'exécution par rapport aux implémentations disponibles auparavant.

Mots clés:

la morphologie mathématique basée sur les (non pondérée) graphes, cartes de distance, stratégie parallèle, architecture multicoeur.

ملخص

المورفولوجيا الرياضية هي واحدة من أقوى الإطارات التي توفر مجموعة من أدوات التصفية والتجزئة و التي هي مفيدة جدا في تطبيقات تحليل الصور. المجال الأول للتطبيقات المورفولوجيا الرياضية كان الصورة الثنائية بواسطة ماثرون وسرا في عام 1964 [54]. الجانب النظري للمورفولوجيا الرياضية يقوم على أن فضاء الصورة هو شعرية كاملة [28] الذي يسمح بمعالجة فئة واسعة من البيانات مع المورفولوجيا الرياضية. من جهة أخرى، الأخذ بالنظر في الكائنات الرقمية التي تحمل المعلومات الهيكلية, المورفولوجيا الرياضية طبقت على الرسوم البيانية والمجمعات البسيطة وعلى هيبيرغرافس. بتركز تقرير هذه الأطروحة على سياق العوامل المورفولوجية التي نرتكز على أساس الرسوم البيانية [14] التي تسمح للانتقال من نوع واحد من مجموعات إلى أخرى حيث المدخلات والمخرجات على حد سواء هي الرسوم البيانية من أجل الحصول على تطبيقات تعمل على مجموعة فرعية من الحواف، على المجموعة الفرعية من القمم، وعلى جزء من رسم بياني معين.

الهدف الرئيسي الرئيسي من هذه الرسالة هو توفير التنفيذ الفعال لهذه العوامل المورفولوجية التي تعتمد على الرسوم البيانية. من أجل هذه الغاية، سوف ندرس العوامل المورفولوجية على أساس (غير الموزونة) الرسوم البيانية. في الواقع، تعتمد هذه العوامل المورفولوجية على معامل الحجم الذي يحدد عدد مرات التكرار من التوسعات / التقلصات الاولية. إذا وقت التنفيذ المرتبط بهذه العوامل المكررة متعلق بمعامل الحجم وتعقيد الخوارزمية هو (ن. λ) حيث ن هو حجم الرسم البياني و λ هي معامل الحجم. نحن مهتمون بشكل خاص ببطاقات المسافة التي سمحت لنا (بالعتبة) بوصف واستعادة كل عمليات التوسعات / والتقلصات, وبالتالي جميع العوامل المورفولوجية المقترحة في [14].

في الجزء الأول، سوف نقترح ثلاث خرائط مسافية جديدة على الرسوم البيانية تسمى : حافة- حافة، حافة-قمة، قمة-حافة. وبينا أن هذه الخرائط المسافية تؤدي إلى الوصف الأصلي لجميع العوامل المورفولوجية المقترحة في [14]. ثم، اقترحنا خوارزميات بناء على هذه الخرائط المسافية الجديدة التي تمكن العوامل من أن تحسب بتكرار واحد مع تعقيد (ن) دون أي الاعتماد على معامل الحجم.

في الجزء الثاني، اقترحنا استراتيجية موازية لحساب فعال لهذه الخرائط المسافية الجديدة. الفكرة الرئيسية لهذه الإستراتيجية هي بناء تكراري لمستويات الخرائط المسافية، حيث كل مجموعة من المستويات تتم معالجتها بشكل متواز. مع الأخذ بعين الاعتبار بعض الافتراضات المعقولة على الرسم البياني والمجموعات التي يراد توسيعها, خوارزميتنا الموازية تعمل في: (ن / ص + ص log2 ك) حيث ن هي حجم الرسم البياني , و ك عدد مستويات مختلفة من خريطة المسافة , و ص عدد المعالجات المتوفرة ، على التوالي.

بعد ذلك، الجزء الثالث يمثل تقييم نظريا و تنفيذ استراتيجيتنا الموازية على بنية متعددة المعالجات و ذاكرة مشتركة بينهم و على 45 مجموعة صور ثنائية الأبعاد و 6 مجسمات ثلاثية الأبعاد، وأجرينا تحليل للنتائج التي تم الحصول عليها من خلال تطبيق وتنفيذ خوارزميات المتتابعة والمتوازية المقترحة على البنى المستهدفة والتي أظهرت ربحا في المعالجة.

الكلمات الافتتاحية:

العوامل المورفولوجية على أساس (غير الموزونة) الرسوم البيانية, الخرائط المسافية, استراتيجية موازية، بنية متعددة المعالجات.

Contents

List of Figures

List of Tables

Chapter 1

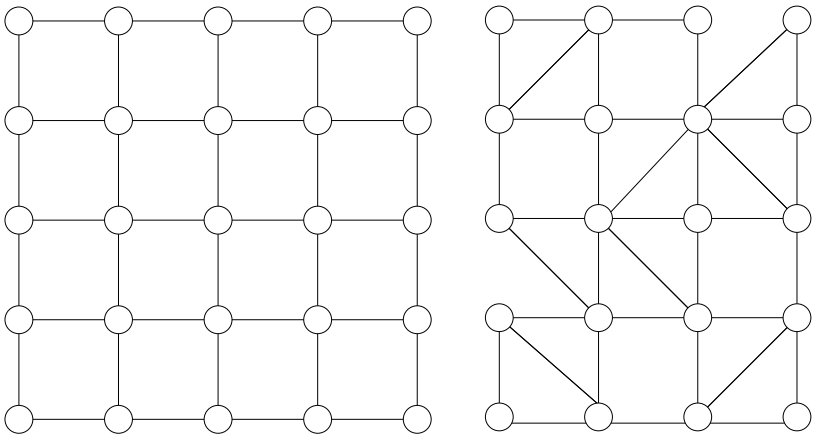
General introduction

Contents

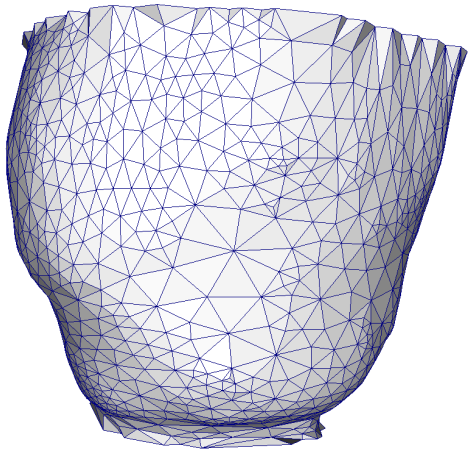
1.1 Context

Mathematical morphology was introduced in 1964 by Matheron and Serra [54]. It holds a fundamental role since it offers powerful set of filtering and segmenting tools that are very useful in applications to image analysis such as noise removing [26], biometrics [48], image segmentation [43, 30], medical imaging [53, 2],etc. From a historical point of view, the first field of applications of mathematical morphology was digital images, *i.e.*, sets of pixels aligned on a 2D or 3D grid and equipped with binary, grayscale, or vectorial values [54]. However, the theoretical basis of mathematical morphology relies on the abstract algebraic structure of a complete lattice [28] allowing us to consider the processing of a very broad class of data with mathematical morphology operators. On the other hand, there is a growing interest for considering digital objects not only composed of points but also composed of elements lying between them and carrying structural information about how the points are glued together (see *e.g.*, [45], [12, 13, 18, 17], and [7] for recent mathematical morphology operators on graphs, on cubical/simplicial complexes, and on hypergraphs respectively). The simplest of these representations are the graphs. The domain of an image is considered as a graph (which can be planar or not) whose vertex set is made of the pixels and whose edge set is given by an adjacency relation on these pixels (see Figure 1.1(a)). Note that this adjacency relation can be either spatially invariant or spatially variant (see Figure 1.1(b)) leading to operators that are either spatially invariant or spatially variant (see applications of spatially variant mathematical morphology [35, 57, 56]). Graphs are

also useful to process other kinds of discrete structures defined for instance on 3-dimensional meshes (see Figure 1.1(c)). In this context, it becomes relevant to consider morphological transformations acting on the subsets of vertices, the subsets of edges and the subgraphs of graphs and not only those acting on the set of all subsets of pixels.



(a) Pixel adjacency graph (b) Spatially variant graph



(c) 3-dimensional meshes

Figure 1.1: Different kinds of discrete structures.

The first work about mathematical morphology on graphs dates back to Vincent [58] who introduced operators relying on a dilation (and its adjunct erosion) that deal with the vertices $X^\bullet$ of a graph $\mathbb{G}$. These operators map to a set of vertices the set of all vertices composed of neighborhoods of the elements in the initial set. More recently, [14, 42] introduce a framework of mathematical operators where the input and output are both graphs. They investigate four basic dilations and erosions that map a set of vertices to a set of edges and a set of edges to a set of vertices. The four elementary operators can be combined in order to obtain operators acting on the subsets of edges, on the subsets of vertices and on the subgraphs of a given graph. These last operators acting on subgraphs enriches the one acting on subsets of vertices [58] by allowing the distinction between the connectivity and adjacency information. This distinction is very interesting to enrich and

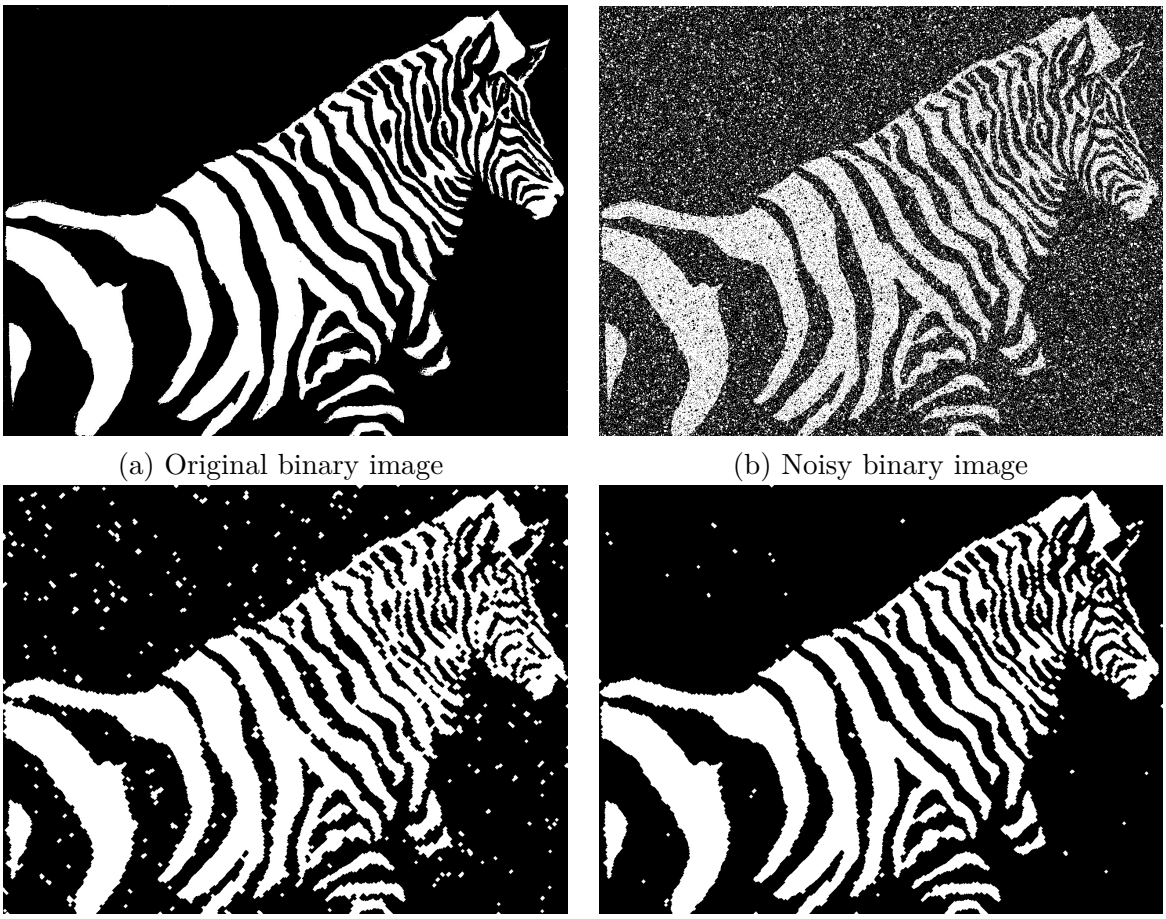
introduce efficient filter in Mathematical morphology [14].

1.2 Motivation and Objectives

In morphology interesting morphological filters called opening and closing are obtained by composition of corresponding (adjunct) dilations and erosions [28, 27, 50]. Such filters as openings, closings are useful in applications in order to filter out noise and to regularize the contours of images. Furthermore, composing openings and closings with increasing size parameter, alternate sequential filters (ASFs) are obtained. They generally outperform the results of elementary openings and closings in applications to image regularization. The openings, closings, and then the associated alternate sequential filters resulting from the compositions of the dilations and erosions of [14] act either on sets of vertices or on sets of edges of graph $\mathbb{G}$. For this, using this alternate sequential filter based on graph, more noise are removed and more details are preserved compared to the usual ASF (see in particular the head of the zebra in Figure 1.2).

Our work in this thesis focuses on the graph-based mathematical morphology operators presented in [J. Cousty et al, " Morphological filtering on graphs ", CVIU 2013]. The basic operators considered in this framework can be combined (iterated) to obtain efficient filters acting on the subsets of edges, on the subsets of vertices and on the subgraphs of a given graph. The behavior of the iterated operators depends on the parity of a size parameter λ. Indeed, we distinguish two cases even and odd value of λ. When λ is even value, the input and output sets of operators are the same. In this case, the operators acting on vertices amount to classical morphology. Whereas, when λ is odd, the input and output sets of operators are distinct. The operators defined in this case cannot amount to classical operators. Based on the straightforward definition, the naive computation of the iterated operators consists of performing λ iterations of basic operators. This parameter λ related to the number of iterations that constitutes a filtering parameter corresponding to the size of the features to be preserved or removed. Higher numbers of iterations are considered when larger structures are removed/preserved. Consequently, the time-complexity is very significantly influenced by the size parameter λ. Thus, if the complexity of basic operator is $O(n)$ where n is the size of the underlying graph, then the time-complexity of the associated algorithm increases with the size parameter. More precisely, for a parameter value of λ the algorithm runs in $O(\lambda.n)$ time.

Therefore, the main objective of this work is to provide efficient computation and implementation of these iterated morphological operators on graphs in order to optimize the time complexity of these operators. To this end, we need first to avoid the dependence with the parameter λ in the time-complexity. Then, we need to exploit the parallel machines in order to benefit from the multi-core/processor architecture. This leads to a fast implementation of these morphological filtering on graphs, hence all applications using these



(a) Original binary image (b) Noisy binary image

(c) Usual ASF (d) ASF on graph

Figure 1.2: ASF Illustration.

operations also get automatically faster.

1.3 Thesis Contributions

Our work in thesis consists of developing threefold: First, we introduce three original notions of distance maps on graphs called edge-edge, edge-vertex, and vertex-edge distance maps to compute the results of the operators of [14]. In addition, we present our vertex-vertex distance map based on new notion of length of path which considers both the numbers of vertices and of edges along the path. Given a set of edges, the edge-edge (resp. edge-vertex) distance map provides for each edge (resp. each vertex) of the graph a geodesic distance to the closest edge in the input set. Given a set of vertices, the vertex-edge (resp. vertex-vertex) distance map provides for each edge (resp. each vertex) a geodesic distance to the closest vertex in the input set. Then, we prove that the introduced distance maps lead to original characterizations of all dilations and erosions on graphs presented in [14]. In particular, we prove that the edge-vertex (resp. vertex-edge) distance maps allow us to characterize (by thresholding) the two operators acting on subset of edges (resp. on the

subset of vertices) defined when λ is odd. The edge-edge (resp. vertex-vertex) distance maps allow us to characterize (by thresholding) the two operators acting on subsets of edges (resp. on vertices), when λ is even.

Second, we propose sequential algorithms to compute vertex-vertex, edge-edge, vertex-edge and edge-vertex distance maps in linear time with respect to the size of the graph $\mathbb{G}$. These algorithms are variations on breadth-first search [11]. The basic idea is to perform a breadth-first exploration of the edges (resp. the vertices) of $\mathbb{G}$ in order to compute edge-edge and edge-vertex (resp. vertex-vertex and vertex-edge) distance maps. The distance map value of each edge (resp. vertex) is computed when the first edge (resp. vertex) is encountered during exploration. The breadth-first exploration, which ensures correct distance values, is made possible thanks to the auxiliary queue $\mathcal{Q}$ that is managed with a FIFO (First-In-First-Out) property. .

These algorithms allow us for efficiently computing the results of all operators in linear time with respect to the size of the graph, with a single iteration and without any dependence to the size parameter of the operators. Then, we establish the correctness and we analyze the complexity of these algorithms.

Third, we propose a first parallelization strategy leading to a fast computation of the proposed distance maps, hence the morphological operators of [14]. In order to handle the regular and non-regular structure of a graph, our new parallelization strategy is based on dynamic partitioning which depends on the input set and which is iteratively computed during the execution. In order to state the complexity of our parallel algorithms, we propose and prove some assumptions about the graph and set under consideration. The assumptions assert that the unbalancing factor of the graph $\mathbb{G}$ is lower than a small constant and the considered set of vertices is p-regular. under these proved assumptions, our algorithm runs in $O(n/p + K \log_2 p)$ time, where n, p, and K are the size of the underlying graph, the number of available processors and the number of distinct level sets of the distance map, respectively.

1.4 Thesis Outline

The remainder of this dissertation is organized as follows. In chapter 2, we present briefly introduction about classical mathematical morphology. Then, a survey of the state of the art covered three part is presented. The first part aims at defining the framework of mathematical morphology based on graphs defined in [14]. We discuss in detail the problem of the computation of iterated morphological operators on graphs in order to obtain interesting morphological filters. The second one is about fundamental notions about distance map on graphs and the problem of finding the length of the shortest paths are presented. In the last part, we present a short overview about parallel computing followed by presentation of related work about parallel distance maps.

Chapter 3 introduces new notion of distance maps that lead to characterizations of all dilations and erosions presented in [14]. Then, characterization theorems lead to obtain by thresholding the proposed distance maps all dilations and erosions on graphs are proved. After, Linear-time algorithms for distance maps in unweighted graphs derive from breadth first search are presented in order to efficiently compute these proposed distance maps, hence all dilations and erosions of [14].

Chapter 4 is devoted to the description of the proposed parallelization strategy that lead to fast compute of the proposed distance maps, hence the morphological operators of [14]. We also present necessary auxiliary functions of the proposed parallel strategy in this chapter. The complexity of the parallel algorithms are analyzed under some proved assumptions about the graph and the set under consideration.

In chapter 5 , we describe the target platform used for multithreaded implementation of our proposed parallel strategy and the datasets used for experimental evaluations. Then, a set of experiments is reported on the used datasets that containing 2D images commonly used in the literature as well as 3D textured meshes. The assessment confirms in practice the regularity assumptions on the considered datasets. Furthermore, in term of execution times, the assessment shows, in particular, that the proposed parallel implementation runs up to 55 times faster achieved for 8 cores machine than the implementations of the operators of [14] that were available before the present work.

Finally, chapter 6 concludes the manuscript. Future works and directions are pointed as well.

Chapter 2

State of the Art

Contents

2.1 Introduction

Mathematical morphology, which covers both a theory and a practice, appeared in the middle of the 1960s by Matheron and Serra for describing structures seen under the microscope (porous media, crystals, alloys, etc.), in order to link them to their physical

properties [46]. It is based on set theory, for this the fundamental objects are considered as sets. Matheron and Serra [40, 54] work to develop this theory of Mathematical Morphology for binary images (sets of points with the use of set theoretical operations) which was extended in 1970s to numerical functions (for a full study see [54]). During the 1980s, when morphological operations were acting on increasingly varied objects, the need for unification led to the concept of complete lattice as a common framework [46]. This fundamental structure of a complete lattice [28] allows to consider the processing of a very broad class of data with mathematical morphology operators such as on graphs. This last represents the background in which our work in this thesis has been developed.

After presenting briefly introduction about classical mathematical morphology, in this chapter, we provide a survey of the state of the art divided into threefold. At first, the framework of mathematical morphology based on graphs defined in [14] are reviewed in Section 2.5.2. Efficient filters for processing images or more generally graphs can be obtained by composition and iteration their elementary operators. The problem statement of our work which concerning the computation of iterated morphological operators on graphs is discussed in Section 2.5.3. There exist efficient algorithms based on distance maps in [58] to compute some of these iterated operators. For this end, we present secondly fundamental notions about distance map on graphs in Section 2.6. The computation of distance map is reduced to the computation of shortest paths. So, the problem of Finding the length of the shortest paths is studied in Section 2.6.1. Finally, to get fast computation, we present a short overview about parallel computing in Section 2.7, followed by presentation of related work about parallel distance maps in Section 2.8.

2.2 Digital Images

A discrete (digital) image is produced by a process of transformation of continuous image called digitisatdion (*i.e.* discretization of spatial coordinates). Discrete images I is consider as a function $f : X \to Y$ where X is the supported space (rectangular domain of image), $X \subset \mathbb{Z}^2$ or $X \subset \mathbb{Z}^3$ and Y is a countable set of defined values of f. Each element $x \in X$ is called an image point. If $X \subset \mathbb{Z}^2$ the image point is called a pixel, and if $X \subset \mathbb{Z}^3$ is a voxel. The value y of an image point x is defined as $y = f(x)$. Depending to the set of image values Y we can specify the type of image. If Y contains exactly two elements (zero or one), the function f is considered as a binary image. The gray-scale image consider Y as an ordered set of gray levels, or more generally $Y \subset \mathbb{Z}$ [46].

2.3 3-Dimensional Triangular Meshs

The 3D triangular meshes have become the most common representation for the surface of a 3D object in computer vision (see examples of a triangular mesh model in Figure 2.1).

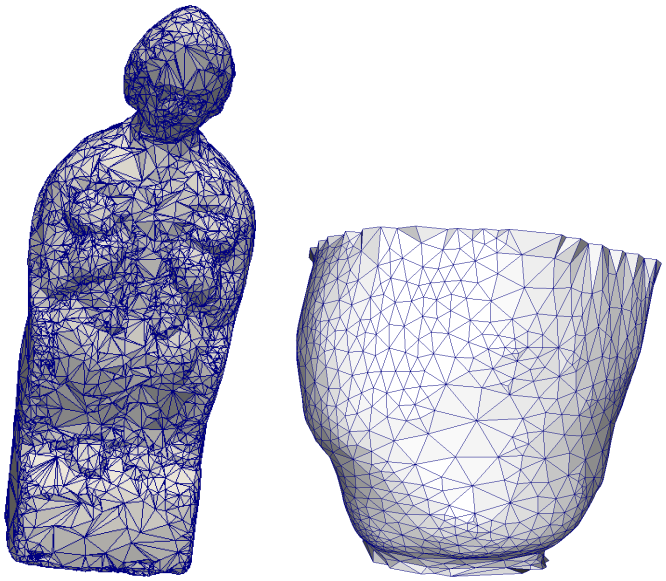


Figure 2.1: Examples of a triangular mesh models.

In practice, a 3D Mesh can be defined as a (list) set of vertices $V = (v_1, \ldots, v_{nV})$ (points in 3D space), and set of faces $F = (f_1, \ldots, f_{nF})$ that define the shape of an object in 3D computer graphics. The face list consists the list of triangles, each one defined as an ordered list of three vertices identifying the corners. A list of edges $E = (e_1, \ldots, e_{nE})$ (sides or line segments), can be obtained implicitly from the first two sets. A mesh structure does not contain any isolated vertices or edges. So, a triangular mesh is composed of triangles, sides (line segments) and corners (points) glued together according to certain rules (*e.g.* two triangles can have a common side or a common corner) [45].

It can be associated to a mesh a texture by giving a color to each vertex, or by mapping each face to an 2-D image.

2.4 Mathematical Morphology

The objective of the Mathematical Morphology operators is extracting some relevant spatial information from an image (for an extensive study see [54]). Where the image is considered as a set, all morphological operators trait this image (set) with another set of a known shape called the Structuring Element. This Structuring Element used to probe the image under study for properties of interest. It allows us to define arbitrary neighborhood structures of points. This element is simply a small binary image (*i.e.* matrix of pixels with a value of one or zero) and has an origin which is usually one of its pixels. The structuring element size is specified by the matrix dimensions, and its shape is specified by the position of ones and zeros. The shape of this element influences significantly on the result of morphological operators. Thus, the choice of the shape and size of this element is

made with respect to some a priori knowledge of the image geometry. Figure 2.2 illustrates some examples of structuring elements.

There exist two types of structuring element, the first one is the flat Structuring Element which is used for the case of 2D image. The second one is non-flat Structuring Element which is used for the case of 3D object. [49]

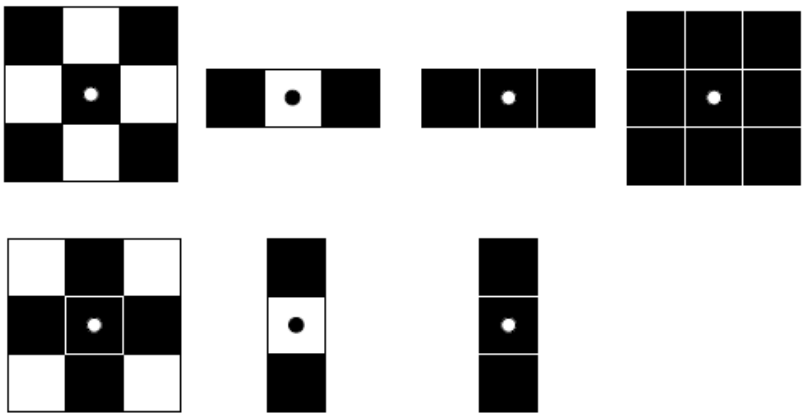


Figure 2.2: Example of simple structuring elements. Black pixels are part of the structuring element, white ones are not. The pixel with the circle marks the origin.

In order to present the classical mathematical morphology operators, we need first to define the complete lattice notion which is the fundamental structure of this theory.

2.4.1 Complete Lattices

A relation of order is a binary relation $\leq$ on E that have the following properties, for all $x, y, z \in E$:

1. $x \leq x$, ($\leq$ is **reflexivity**)
2. if $x \leq y$ and $y \leq x \Rightarrow x = y$, ($\leq$ is **antisymmetry**)
3. if $x \leq y$ and $y \leq z \Rightarrow x \leq z$. ($\leq$ is **transitivity**)

A lattice $(E, \leq)$ is a set E equipped with a relation of order $\leq$ [6]. Such as, for two elements x and y, one can define a larger element $x \bigvee y$ and a smaller element $x \bigwedge y$. A lattice is complete if every subset P of E has both a least upper bound (the supremum) and a greatest lower bound (the infimum). The supremum is denoted by $\bigvee P$, and the infimum by $\bigwedge P$. These two bounds of P (*i.e.* the supremum and the infimum) are defined dually and play symmetrical roles in a trellis.

An illustration of a lattice known as Hasse diagram, is presented in Figure 2.3. In this example, the set $\mathcal{P}(S) = \{\{x, y, z\}, \{x, y\}, \{x, z\}, \{y, z\}, \{x\}, \{y\}, \{z\}\}$ is considered as the power set of the set $S = \{x, y, z\}$. This set is ordered by the inclusion relation in order to

obtain a lattice bounded by S itself and the null set. The supremum of two elements of this lattice is given by the union operator and the infimum by the intersection operator.

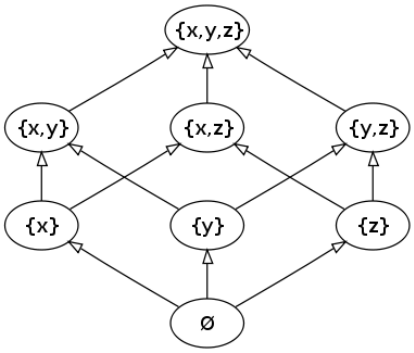


Figure 2.3: An example of a lattice.

2.4.2 Fundamental Operators of Mathematical Morphology

Let us now describe the fundamental operations of mathematical morphology. First, we start by the case of binary mathematical morphology which is based on set theory. Then, we move to the case of gray scale morphology (numerical functions).

We begin our discussion with two operators, dilation and erosion, which are fundamental operators in the mathematical morphology, from which many others operators are derived, such as closing (dilation followed by erosion), opening (erosion followed by dilation) etc.

2.4.2.1 Dilation and erosion

Dilation and erosion are basic morphological morphology operations. They are defined in terms of usual set theoretic operations of union and intersection (the Minkowski addition $\oplus$ and subtraction $\ominus$ [8]), and are depended on the translation operation. We denote the translation of a set X over a vector $y \in E$ by X_y, i.e. $X_y = \{x + y \mid x \in X\}$.

In practice of the mathematical morphology, we study the images (binary and gray-scale image) by a structuring element B. Let X be a binary image (*i.e.* subset of E) and B a structuring element. The dilation of X by B is defined by:

$$\delta_B(X) = X \oplus B = \cup_{b \in B} X_b \tag{2.1}$$

$$= \cup_{x \in X} B_x \tag{2.2}$$

$$= \{x + b \mid x \in X \text{ and } b \in B\}. \tag{2.3}$$

The erosion of X by B is defined by:

$$\epsilon_B(X) = X \ominus B = \cap_{b \in B} X_{-b} \tag{2.4}$$

$$= \{p \in E \mid B_p \subseteq XB\}. \tag{2.5}$$

In general, dilation has the effect of expanding objects in an image, and hence, the small hole can be eliminated. The structuring element control the manner of this growth image. $\delta_B(X) = p, B_p \cap X \neq \phi$ [46]. the sets X and B play symmetric roles in the definition of dilation. The erosion "shrinks" objects in the image and the small peak is eliminated.

It should be noted that the dilation and the erosion are dual by complementation, *i.e.* dilating a set X is equivalent to eroding its complement X^c with the symmetrical structuring element $\check{B}$.

$$(X \oplus B)^c = X^c \ominus \check{B}, (X \ominus B)^c = X^c \oplus \check{B}. \tag{2.6}$$

where X^c is the complement of X in E, that is $X^c = E \setminus X$, $\check{B}$ is the transpose or symmetrical of B that is $\check{B} = \{-b \mid b \in B\}$.

Therefore, the properties of erosion are derived from those of dilation by duality: dilation inflates the object, deflates the background and deforms convex corners of the object; thus erosion deflates the object, inflates the background and deforms concave corners of the object [8]. We present an illustration [46] in Figures 2.4 and 2.5 of the dilation and the erosion of a cross by a triangular structuring element, respectively.

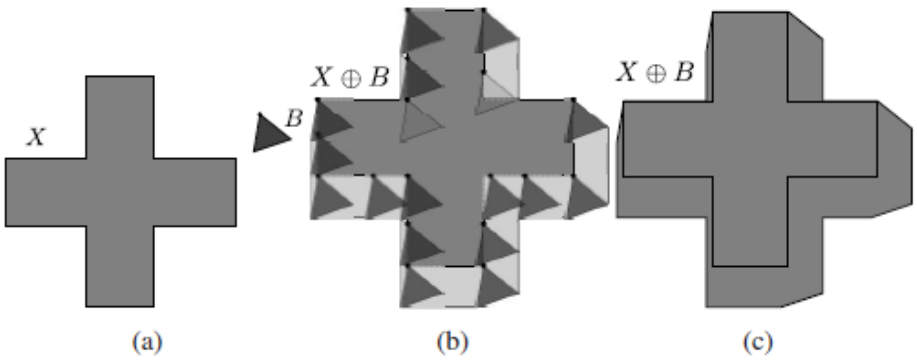


Figure 2.4: The dilation of a cross by a triangle. The origin of the structuring element is one of the vertices of triangle B and is shown as a small black disk: (a) the original X (the light-gray cross) and B (the dark triangle); (b) the dilation taking place; and (c) the final result with the original set X overlaid [46].

The binary mathematical morphological operators can be extended to gray scale mathematical morphological operators. The gray scale dilation and erosion by a flat structuring

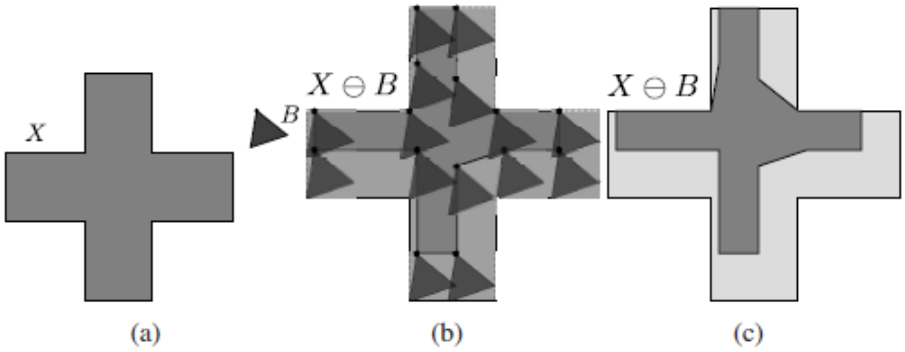


Figure 2.5: The erosion of a cross by the same triangle structuring element as in Figure 2.4: (a) the original X (the light-gray cross) and B (the dark triangle); (b) the erosion taking place; and (c) the final result, overlaid within the original set X [46].

element can be defined by two versions [5]. The first one is an extension to functions of the Minkowski addition as follows:

$$[\delta_B(f)](x) = [\bigvee_{b \in B} f_b](x). \tag{2.7}$$

$$[\epsilon_B(f)](x) = [\bigwedge_{b \in \check{B}} f_b](x). \tag{2.8}$$

where f_b is the translations of f by a vector $b \in B$ which is computed by $f(x + b)$.

The second one is an extension to function of the intersection/inclusion operations of set as follows:

$$[\delta_B(f)](x) = \wedge\{v \in Y \mid \check{B} + v \geq f\}. \tag{2.9}$$

$$[\epsilon_B(f)](x) = \vee\{v \in Y \mid \check{B} + v \leq f\}. \tag{2.10}$$

The implementation of the gray scale dilation consists of finding the maximum of f within the scope of B whereas the erosion consists of finding the minimum of f such as [4]:

$$[\delta_B(f)](x) = \max_{b \in B}[f(x + b)]. \tag{2.11}$$

$$[\epsilon_B(f)](x) = \min_{b \in \check{B}}[f(x + b)]. \tag{2.12}$$

Dilation and erosion are the basic building block for a large class of morphological operators. They are considered as an adjunction pair of operators (δ, ϵ) *i.e.* $\delta(X) \leq Y \Leftrightarrow X \leq \epsilon(Y)$ [46] that allows the composition of dilation and erosion to form morphological filters

2.4.3 Morphological filtering

In mathematical morphology, a morphological filter on a lattice $\mathcal{L}$ is an operator α which is both increasing $\forall X, Y \in \mathcal{L}, \quad X \leq Y \Rightarrow \alpha(X) \leq \alpha(Y)$ and idempotent $\forall X \in \mathcal{L}, \quad \alpha(\alpha(X)) = \alpha(X)$. The two main operators derived from dilation and erosion are opening and closing.

2.4.3.1 Opening and Closing

The opening and closing operators can be defined as a composition of operators of dilation and erosion by a structuring element. Thus, the opening of X by B is a morphological filter (increasing and idempotent) that is also anti-extensive (*i.e.* $\alpha(x) \leq x$). It is defined as follows:

$$\gamma_B(X) = X \circ B = (X \ominus B) \oplus B \tag{2.13}$$

$$= \cup\{B_p, p \in E \text{ and } B_p \subseteq X\}. \tag{2.14}$$

and the closing of X by B is the complement of the opening and is extensive (*i.e.* $x \leq \alpha(x)$). It is defined as follows:

$$\varphi_B(X) = X \bullet B = (X \oplus B) \ominus B. \tag{2.15}$$

In the other word, the opening X by B is the erosion of X by B, followed by a dilation of the result by B. The closing X by B is the dilation of X by B, followed by an erosion of the result by B. This two operators of opening and closing are dual by complementation.

The opening smoothes the contour of an object and removes narrow parts of the object. The closing also allows to smooth sections of contours but filling narrow breaks and long, thin gulfs and eliminating small holes.

The gray scale opening and closing are defined as follows:

$$\gamma_B(f) = \vee\{B + v \leq f\}. \tag{2.16}$$

$$\varphi_B(f) = \wedge\{\check{B} + v \geq f\}. \tag{2.17}$$

and they can be implemented by:

$$\gamma_B(f) = \delta_{\check{B}}[\epsilon_B(f)]. \tag{2.18}$$

$$\varphi_B(f) = \epsilon_{\check{B}}[\delta_B(f)]. \tag{2.19}$$

Note that the opening and closing are dual operators.

2.4.3.2 Granulometries

The granulometric families were introduced by [40] in a study of porous materials. These families are a sequence of openings and closings that are parametrized by an integer parameter λ. A granulometry is a decreasing family of openings $\Psi = (\psi_\lambda)_\lambda$ that satisfied:

$$\forall \lambda \geq 0, \quad \psi_\lambda \text{ is an opening} \tag{2.20}$$

$$\forall \lambda \geq 0, \mu \geq 0, \ \lambda \geq \mu \Rightarrow \psi_\lambda \leq \psi_u. \tag{2.21}$$

Similarly, anti-granulometry is a family of increasing closings.

2.4.3.3 Alternative Sequential Filters

The alternative sequential filters (ASF) are, as the name indicates, a composition of openings and closings which form granulometric families of increasing sizes such as:

$$ASF_n = \gamma_n\varphi_n\gamma_{n-1}\varphi_{n-1}\ldots\gamma_1\varphi_1 \text{ or} \tag{2.22}$$

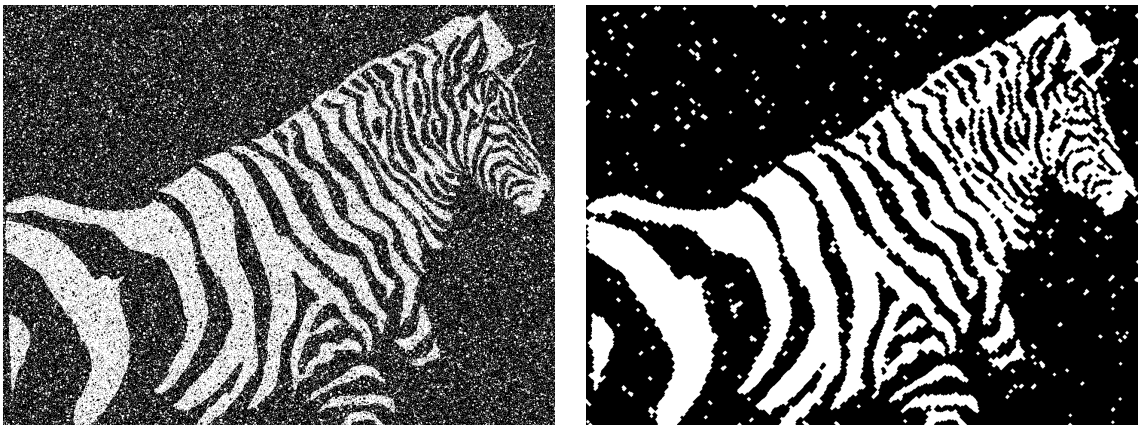
$$ASF_n = \varphi_n\gamma_n\varphi_{n-1}\gamma_{n-1}\ldots\varphi_1\gamma_1 \tag{2.23}$$

For instance, the ASF beginning with an opening is called the white. Whereas, the black ASF which begins with a closing [46].

The alternative sequential filters can be used to denoise both binary and grayscale images (see Figure 2.6).

2.5 Graph-based mathematical morphology framework

In this section, we recall background notions for mathematical morphology on graphs. After providing basic concepts for graphs, we present four operators studied in [15, 14]. They constitute a set of basic building blocks for morphological filters on graphs.



(b) Noisy binary image (b) ASF

Figure 2.6: ASF Illustration.

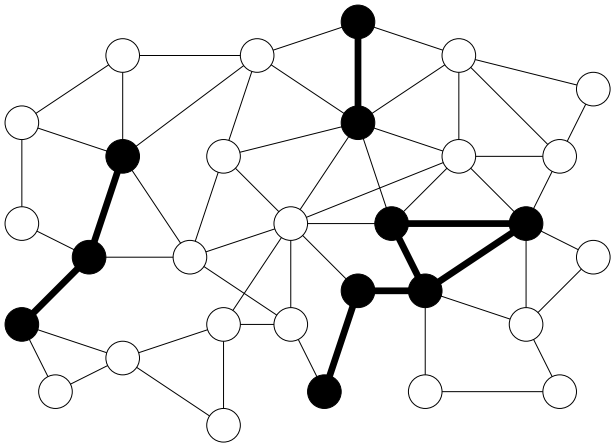


Figure 2.7: Undirected graph.

2.5.1 Basic theoretical concepts of graphs

Definition 1 (Graph)**.** *A* ***(undirected) graph*** *is a pair* $X = (X^{\bullet}, X^{\times})$ *where* $X^{\bullet}$ *is a finite nonempty set and* $X^{\times}$ *is composed of unordered pairs of distinct elements in* $X^{\bullet}$*,* i.e.*,* $X^{\times}$ *is a subset of* $\{\{x, y\} \subseteq X^{\bullet} \mid x \neq y\}$*. Each element of* $X^{\bullet}$ *is called a* vertex or a point (of X)*, and each element of* $X^{\times}$ *is called an* edge (of X)*.*

A ***(directed) graph*** *or digraph graph is a pair* $X = (X^{\bullet}, X^{\times})$ *where* $X^{\bullet}$ *is a set and* $X^{\times}$ *is a set of ordered pairs which are named directed edges.*

A ***weighted graph*** *is a graph with weight function which assigns a real value to each edge. Whereas, when the weight function returns one for all edges one, the graph is called* unweighted graph*.*

Figure 2.7 illustrates an example of undirected graph where the vertices are represented by black dots and the edges by black line segments.

Definition 2 (Subgraph)**.** *Given* X *and* Y *be two graphs. If* $Y^{\bullet} \subseteq X^{\bullet}$ *and* $Y^{\times} \subseteq X^{\times}$*,*

then X and Y are ordered and we write $Y \sqsubseteq X$. If $Y \sqsubseteq X$, we say that Y is a subgraph of X, or that Y is smaller than X and that X is greater than Y [14].

Definition 3 (Degree of vertex)**.** *Let x be a vertex of graph $\mathbb{G}$, the* degree of x*, denoted by $d(x)$, is the number of edges that contain x,* i.e.*, $d(x) = |\{\{x, y\} \in \mathbb{G}^{\times}\}|$.*

Definition 4 (Regular graph)**.** *A graph $\mathbb{G}$ is regular if every vertex x has the same degree. We say that $\mathbb{G}$ is regular graph of degree k (or k-regular) if: $\forall x \in \mathbb{G} \mid d(x) = k$.*

Definition 5 (Adjacency relation)**.** *We say that an edge $\{x, y\}$ connects x and y. Thus, the vertices x and y are adjacent and x is a neighbor of y (and vice versa).*

In graph $\mathbb{G}$, the vertex set $X^{\bullet}$ represents the image domain and the edge set $X^{\times}$ is given by an adjacency relation. In fact, there are several kinds of adjacency relation for examples see Figures 2.8 and 2.9 for 2D and 3D images respectively.

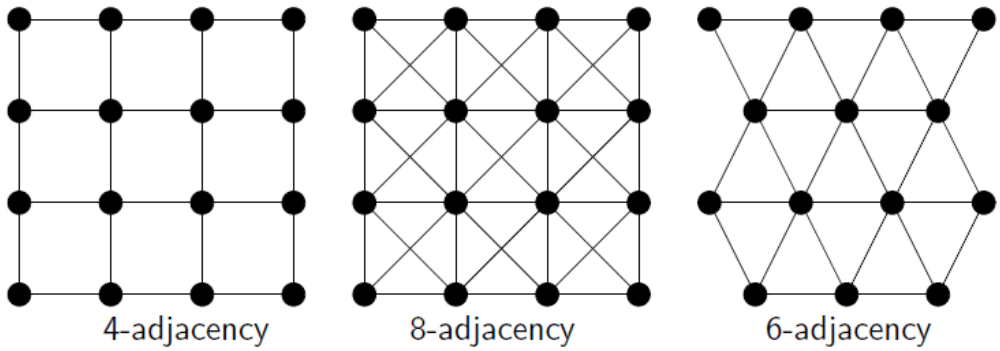


Figure 2.8: Example of adjacency relation.

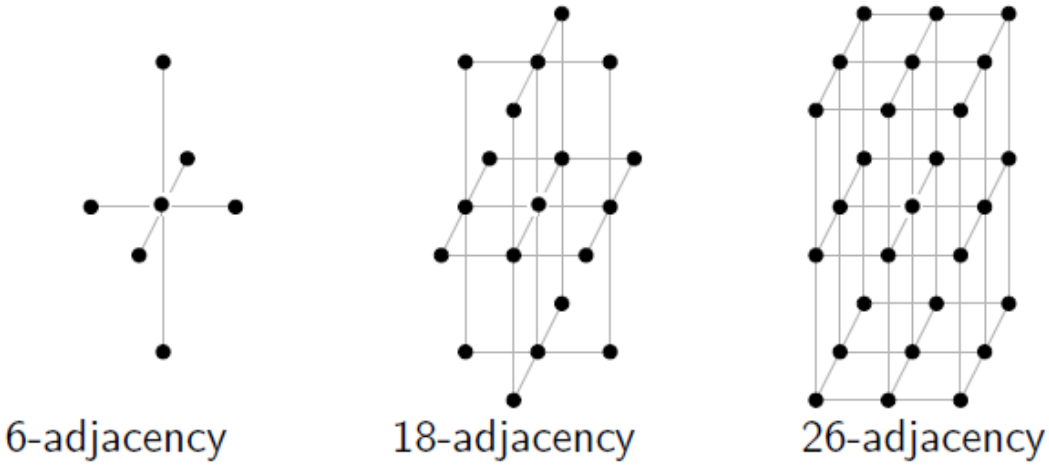


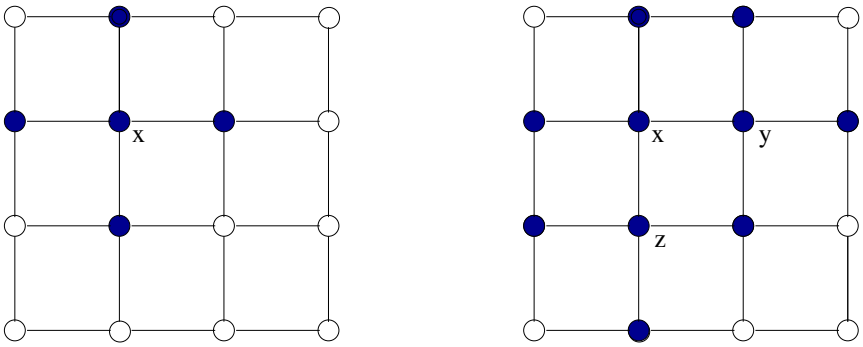
Figure 2.9: Example of adjacency relation for 3D images.

Definition 6 (Neighborhood). *The neighborhood of a vertex x in graph $\mathbb{G}$ is the set of all vertices linked by an edge in $\mathbb{G}$ to this vertex x (see Figure 2.10(a)):*

$$\forall x \in \mathbb{G}^{\bullet}, \Gamma(x) = \{y \in \mathbb{G}^{\bullet} | \{x, y\} \in \mathbb{G}^{\times}\} \tag{2.24}$$

The neighborhood in graph $\mathbb{G}$ of a subset of vertices, is the union of the neighborhood of the vertices exist in this set X (see Figure 2.10(b)):

$$\forall X \subseteq \mathbb{G}^{\bullet}, \Gamma(X) = \{\cup_{x \in X} \Gamma(x)\} \tag{2.25}$$



(a) Neighborhood of vertex x (b) Neighborhood of subset $X = \{x, y, z\}$.

Figure 2.10: Example of neighborhood relation.

Definition 7 (Path). *A path π in graph $\mathbb{G}$ is an ordered finite sequence written: $\pi = (x_0, \ldots, x_\ell)$. We say π is a path from x_0 to x_ℓ such that: any two consecutive vertices x_i, x_{i+1} of π are linked by an edge: $\forall i \in [1; \ell], \{x_i, x_{i+1}\} \in \mathbb{G}^{\times}$.*

The length of the path π is denoted by $L(\pi)$, in unweighted graph is often considered as the sum of the number of edges along the path.

$$L(\pi) = \sum_{0}^{n-1} \{\ell(x_i, x_{i+1}) | 0 \leq i \leq n-1\} \tag{2.26}$$

In a weighted graph the length of a path is the sum of weights of edges which belongs to the path.

An example of path is illustrated in Figure 2.11.

Definition 8 (Shortest path). *Let x and y be two vertices of $\mathbb{G}$. A shortest path between two vertices x and y is a path π of minimal length from x to y:*

$$\forall \pi' path\ from\ x\ to\ y, L(\pi) \leq L(\pi') \tag{2.27}$$

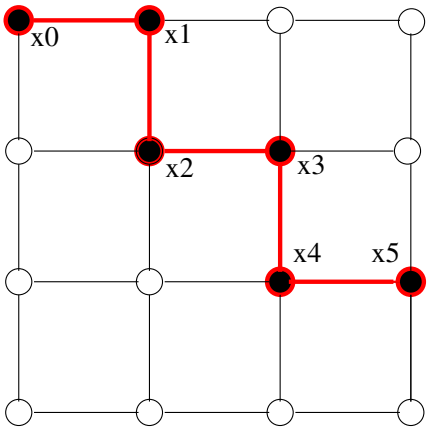


Figure 2.11: A path of length 5

.

2.5.2 Mathematical Morphology operators on graphs

The domain of an image is considered as a graph whose vertex set is made of the pixels and whose edge set is given by an adjacency relation on these pixels. Considering digital objects not only composed of points but also composed of elements lying between them and carrying structural information about how the points are glued together, mathematical morphology has been developed on graphs. In this context, it becomes relevant to consider morphological transformations acting on the subsets of vertices, the subsets of edges and the subgraphs of graphs and not only those acting on the set of all subsets of pixels.

Morphological operators are well studied on graphs. Originally, binary morphology on graphs were proposed by Vincent in 1989 [58]. A dilation (and its adjunct erosion) that deal with the vertices $X^\bullet$ of a graph $\mathbb{G}$ (see Figure 2.5.2(a)). These operators map any set of vertices the set of all vertices composed of neighborhoods of the elements in the initial set. More recently, [14, 42, 44] consider sets of vertices as well as sets of edges. [44, 42] defined operators capable of dealing with weighted graphs to obtain a new approach to image segmentation and levellings, respectively. [14] introduce a framework of mathematical operators where the input and output are both graphs. they investigate four basic dilations and erosions that map a set of vertices to a set of edges and a set of edges to a set of vertices. These operators can be combined in order to obtain operators acting on the subsets of edges, on the subsets of vertices and on the subgraphs of a given graph.

In this framework of mathematical operators on graphs, the structuring element is replaced by the choice of the edge set that indicates which data are connected [45].

Remarking in Figure 2.5.2 that using operators proposed in [58], the dilation of the subset $X^\bullet$ of red and blue vertices of Figure 2.5.2(a) leads to the set of red and blue vertices of Figure 2.5.2(b) which is connected. On the other hand, using operators proposed in [14], the dilation of the subgraph X induced by $X^\bullet$ (depicted in red and blue in Figure 2.5.2(c))

leads to the red and blue subgraphs shown in Figure 2.5.2(d) which are not connected (the two connected components are adjacent to each other)[14]. Thus, these last operators acting on subgraphs enriches the one acting on subsets of vertices by allowing the distinction between the connectivity and adjacency information. This distinction is very interesting to introduce efficient filter in Mathematical morphology [14].

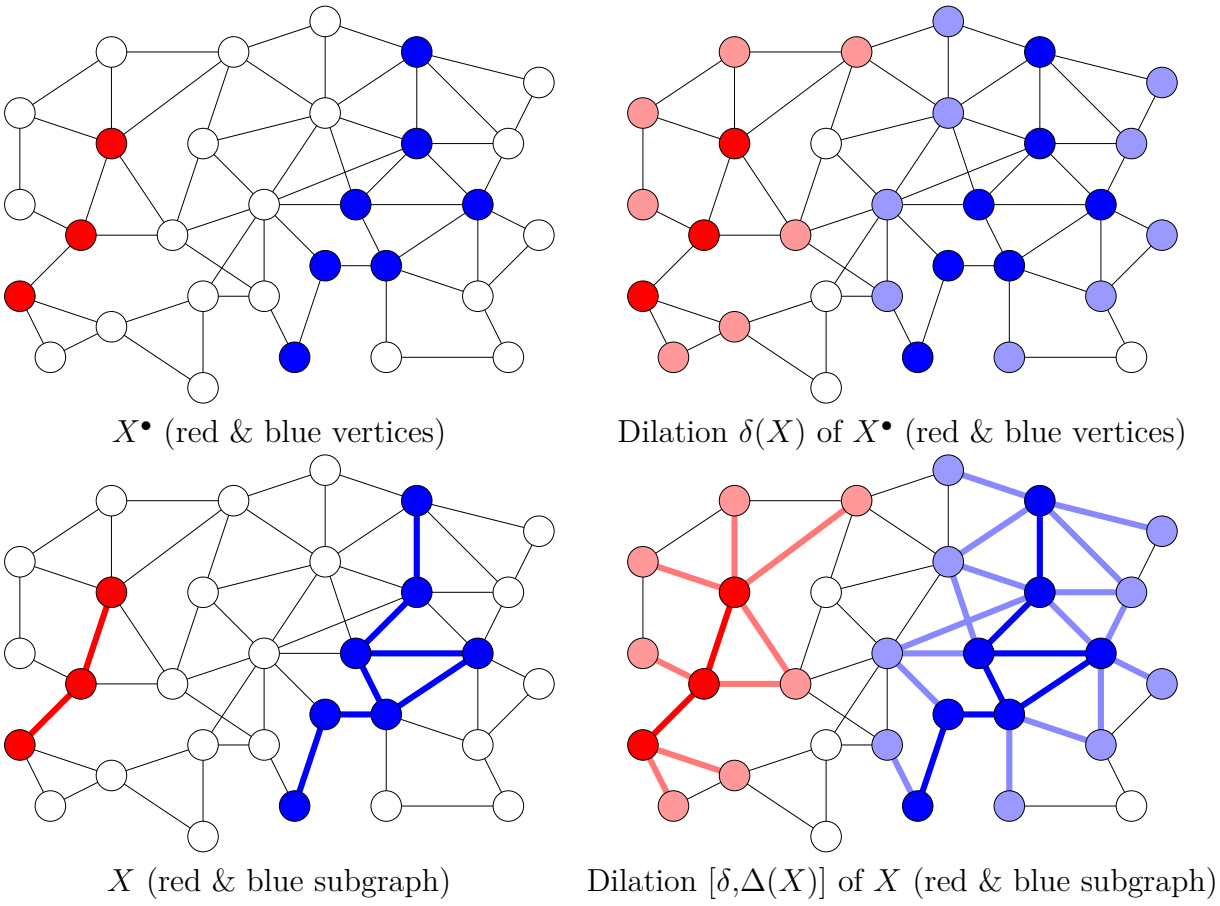


Figure 2.12: Illustration of a vertex-dilation and of a graph-dilation [14].

In the following sections, we will study in detail the framework of mathematical morphology on graphs of [14] in which our work in this thesis has been developed.

2.5.2.1 Lattice of graphs

Recall that a graph G consists of a set of vertices $X^\bullet$, and a binary relation called adjacency relation between two vertices. This adjacency relation is represented by a set of edges $X^\times$.

Important notation. Hereafter, the workspace is a graph $\mathbb{G} = (\mathbb{G}^\bullet, \mathbb{G}^\times)$ and we consider the sets $\mathcal{G}^\bullet$, $\mathcal{G}^\times$ and $\mathcal{G}$ of respectively all subsets of $\mathbb{G}^\bullet$, all subsets of $\mathbb{G}^\times$ and all subgraphs of $\mathbb{G}$.

As seen above in Section 2.4, the fundamental theoretical basis of mathematical morphology is a complete lattice [28]. Thus, according to the Property 9 provided in [14], the

structure of graphs can be characterized by a structure that has the same foundations of complete lattices (see [14] for proof of this property), and so morphological operators can be built on it.

Property 9 (from [14]). *The set $\mathcal{G}$ of the subgraphs of $\mathbb{G}$ form a complete lattice. The infimum and the supremum of any family $F = \{X_1, \ldots, X_\ell\}$ of elements in $\mathcal{G}$ are given by respectively:*

- $\sqcup F = \{\cup_{i\in[1,\ell]} X_i^\bullet, \cup_{i\in[1,\ell]} X_i^\times\}$
- $\sqcap F = \{\cap_{i\in[1,\ell]} X_i^\bullet, \cap_{i\in[1,\ell]} X_i^\times\}$.

2.5.2.2 Elementary morphological operators on graphs

The mathematical morphology filtering on graphs introduced in [15, 14], relies on four basic operators which are used to derive a set of edges from a set of vertices and a set of vertices from a set of edges. Then, based on these basic operators (building blocks), several dilations and erosions acting on the lattice of all subgraphs of $\mathbb{G}$ are derived.

Definition 10. *The operators $\delta^\bullet$ and $\epsilon^\bullet$ from $\mathcal{G}^\times$ to $\mathcal{G}^\bullet$ and the operators $\epsilon^\times$, and $\delta^\times$ from $\mathcal{G}^\bullet$ to $\mathcal{G}^\times$ are defined as follows:*

$$\delta^\bullet(X^\times) = \{x \in \mathbb{G}^\bullet \mid \exists\{x, y\} \in X^\times\}, \text{ for any } X^\times \subseteq \mathbb{G}^\times; \tag{2.28}$$

$$\epsilon^\bullet(X^\times) = \{x \in \mathbb{G}^\bullet \mid \forall\{x, y\} \in \mathbb{G}^\times, \{x, y\} \in X^\times\}, \text{ for any } X^\times \subseteq \mathbb{G}^\times; \tag{2.29}$$

$$\epsilon^\times(X^\bullet) = \{\{x, y\} \in \mathbb{G}^\times \mid x \in X^\bullet \text{ and } y \in X^\bullet\}, \text{ for any } X^\bullet \subseteq \mathbb{G}^\bullet; \text{ and} \tag{2.30}$$

$$\delta^\times(X^\bullet) = \{\{x, y\} \in \mathbb{G}^\times \mid x \in X^\bullet \text{ or } y \in X^\bullet\}, \text{ for any } X^\bullet \subseteq \mathbb{G}^\bullet. \tag{2.31}$$

In other words, the operator $\delta^\bullet$ maps to any edge set $X^\times$ the set of all vertices that belong to an edge in $X^\times$. The operator $\epsilon^\bullet$ maps to any edge set $X^\times$ the set of vertices that are "completely covered" by edges in $X^\times$, *i.e.* the vertices that do not belong to any edge of the complement $\mathbb{G}^\times \setminus X^\times$ of $X^\times$. The operator $\epsilon^\times$ maps to any vertex set $X^\bullet$ the set of all edges whose two extremities are in $X^\bullet$. The operator $\delta^\times$ maps to any vertex set $X^\bullet$ the set of all edges that have at least one extremity in $X^\bullet$.

The operators $\epsilon^\times$, $\delta^\times$, $\epsilon^\bullet$ and $\delta^\bullet$ are illustrated in Figure 2.13 where the workspace $\mathbb{G} = (\mathbb{G}^\bullet, \mathbb{G}^\times)$ is illustrated in Figure 2.14 (first column).

Many mathematical morphology operators rely on the algebraic structure of a lattice [28, 27, 50]. In particular, any operator that commutes with the union (resp. intersection) is called a *dilation* (resp. an *erosion*). It is established in [14] that the operators $\delta^\bullet$ and $\delta^\times$ are indeed morphological dilations and that the operators $\epsilon^\bullet$ and $\epsilon^\times$ are erosions since

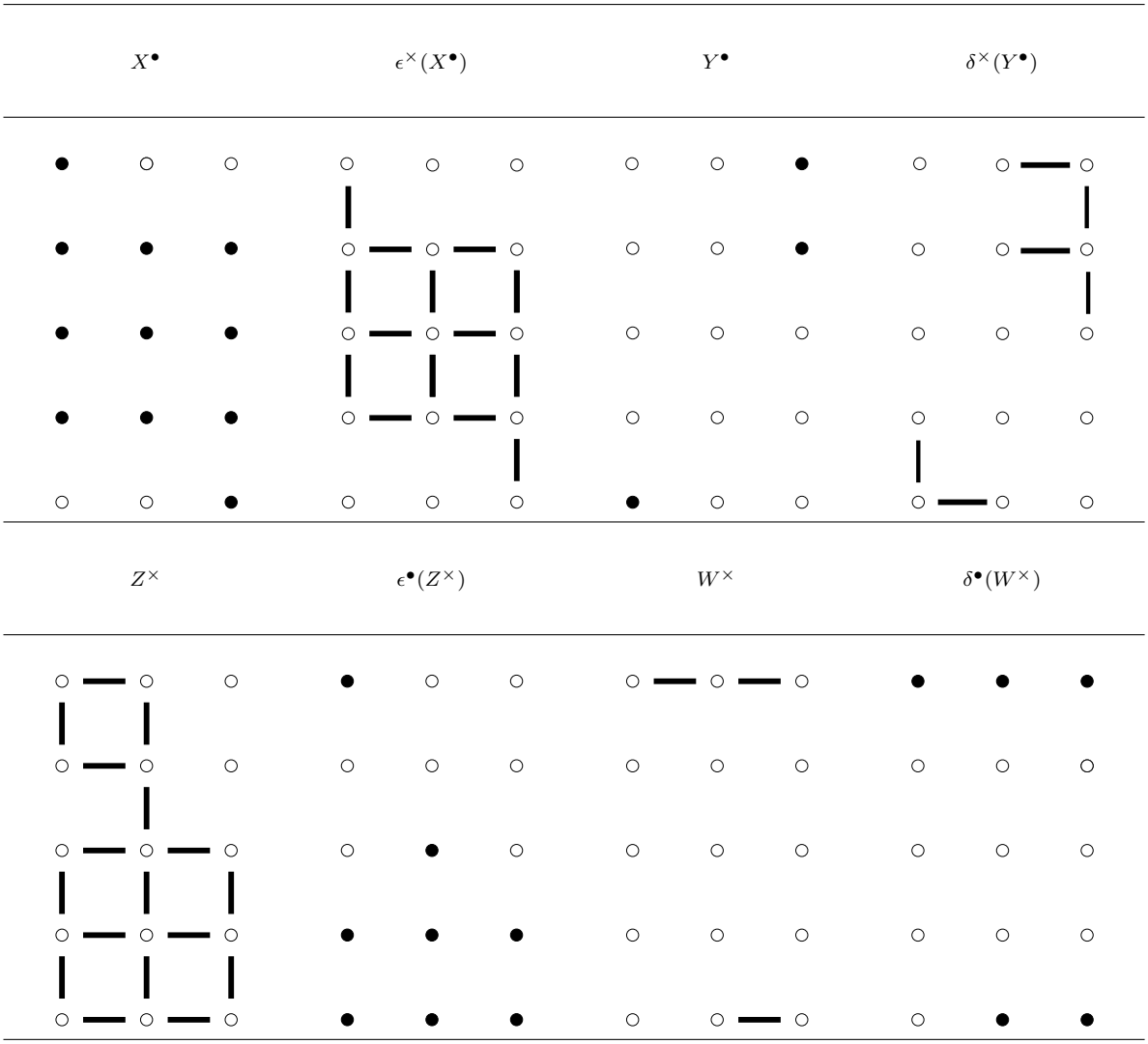


Figure 2.13: Illustration of the operators $\epsilon^\times$, $\delta^\times$, $\epsilon^\bullet$ and $\delta^\bullet$. The considered sets are depicted by black dots or line segments.

they commute with union and intersection respectively, this means that we have:

$$\delta^\bullet(X^\times \cup Y^\times) = \delta^\bullet(X^\times) \cup \delta^\bullet(Y^\times), \text{ for any } X^\times, Y^\times \subseteq \mathbb{G}^\times; \tag{2.32}$$

$$\epsilon^\bullet(X^\times \cap Y^\times) = \epsilon^\bullet(X^\times) \cap \epsilon^\bullet(Y^\times), \text{ for any } X^\times, Y^\times \subseteq \mathbb{G}^\times; \tag{2.33}$$

$$\delta^\times(X^\bullet \cup Y^\bullet) = \delta^\times(X^\bullet) \cup \delta^\times(Y^\bullet), \text{ for any } X^\bullet, Y^\bullet \subseteq \mathbb{G}^\bullet; \text{ and} \tag{2.34}$$

$$\epsilon^\times(X^\bullet \cap Y^\bullet) = \epsilon^\times(X^\bullet) \cap \epsilon^\times(Y^\bullet), \text{ for any } X^\bullet, Y^\bullet \subseteq \mathbb{G}^\bullet. \tag{2.35}$$

Therefore, in the following, the operators $\delta^\times$ and $\delta^\bullet$ (resp. $\epsilon^\times$ and $\epsilon^\bullet$) are referred to as the *vertex-edge dilation* and the *edge-vertex dilation* (resp. the *vertex-edge erosion* and the *edge-vertex erosion*).

In particular, by composition of the four basic operators presented above. The elementary vertex dilation $\delta(X^\bullet)$(resp. edge dilation $\Delta(X^\times)$) is obtained by the composition of

the vertex-edge (resp. edge-vertex) dilation and the edge-vertex (resp. vertex-edge) dilation, whereas the associated vertex erosion $\epsilon(X^\bullet)$ (resp. edge erosion $\varepsilon(X^\times)$) is obtained by the composition of the vertex-edge (resp. edge-vertex) erosion and the edge-vertex (resp. vertex-edge) erosion.

Definition 11. *For any $X^\bullet \subseteq \mathbb{G}^\times$, and $X^\times \subseteq \mathbb{G}^\times$, the following statements hold true.*

$$\delta(X^\bullet) = \{x \in \mathbb{G}^\bullet \mid \exists \{x,y\} \in X^\times, \{x,y\} \cap X^\bullet \neq \phi\}, \tag{2.36}$$

$$\epsilon(X^\bullet) = \{x \in \mathbb{G}^\bullet \mid \forall \{x,y\} \in \mathbb{G}^\times, \{x,y\} \subseteq X^\bullet\} \tag{2.37}$$

$$\Delta(X^\times) = \{\{x,y\} \in \mathbb{G}^\times \mid \textit{either } \exists \{x,z\} \in X^\times \textit{ or} \exists \{y,w\} \in X^\times\} \tag{2.38}$$

$$\varepsilon(X^\times) = \{\{x,y\} \in \mathbb{G}^\times \mid \forall \{x,z\},\{y,w\} \in \mathbb{G}^\times, \{x,z\} \in X^\times, \{y,w\} \in X^\times\}. \tag{2.39}$$

Informally, the set $\delta(X^\bullet)$ is the set of vertices which contains in at least on edge of $\mathbb{G}$ and all vertices adjacent to a vertex in $X^\bullet$ (*i.e.* which share a common vertex with an edge), while the set $\epsilon(X^\bullet)$ contains all vertices of $X^\bullet$ whose neighborhood is in $X^\bullet$. The set $\Delta(X^\times)$ is the set of edges in $\mathbb{G}$ that have at least on vertex with an edge in $X^\times$, while the set $\varepsilon(X^\times)$ contains each edge in $X^\times$ whose neighborhood is in $X^\times$.

Observe that the operators introduced in Definition 10 are given for arbitrary (unweighted, undirected) graphs and are all increasing. Therefore, they are considered as elementary building blocks that induce stack operators acting on functions weighting the vertices and/or edges of a graph (see Definition 12). This leads to define morphological operators for weighted graphs, hence for grayscale images. from the ones on non-weighted graphs proposed in [14].

Definition 12. *Let $F^\bullet \in I(\mathbb{G}^\bullet)$, and $F^\times \in I(\mathbb{G}^\times)$, the following statements hold true.*

$$\delta^\bullet(F^\times)(x) = max\{F^\times\{x,y\} \mid \{x,y\} \in G^\times\}, \quad \forall x \in G^\bullet \tag{2.40}$$

$$\epsilon^\times(F^\bullet)\{x,y\} = min\{F^\bullet(x), F^\bullet(y)\} \quad \forall \{x,y\} \in G^\times \tag{2.41}$$

$$\epsilon^\bullet(F^\times)(x) = \{F^\times\{x,y\} \mid \{x,y\} \in G^\times\} \quad \forall x \in G^\bullet \tag{2.42}$$

$$\delta^\times(F^\bullet)\{x,y\} = max\{F^\bullet(x), F^\bullet(y)\}, \quad \forall \{x,y\} \in G^\times. \tag{2.43}$$

Furthermore, the operators presented in this section allow for defining filters such as opening, closing and alternate sequential filters that are useful in applications [14, 41].

2.5.2.3 Morphological filters on graphs

Interesting morphological filters on graphs can be obtained by composition and iteration of the basic operators of [14] (Iterated versions of basic dilations and erosions see next section). These morphological filters are useful in applications in order to filter out noise and to regularize the contours of images. The size of the structures which are

preserved/suppressed with openings and closings depends on the number of iterations of the basic dilations and erosions. Larger structures are removed/preserved when higher numbers of iterations are considered. Therefore, the number of iterations involved in a morphological operator is often referred to as its *size parameter* λ. The openings and closings resulting from the composing of two operators of an adjunction (iterated dilations and erosions on graphs of next section Definition 13) act either on sets of vertices or on sets of edges according to the parity of λ as recalled in Table 2.1. (see [14] for an extensive study). Furthermore, the simultaneous application of these filters (for a same size parameter) on the vertices and on the edges of a subgraph of $\mathbb{G}$ leads to a subgraph of $\mathbb{G}$, hence morphological filtering on subgraphs.

Domain	**Parity of λ**	**Openings** $[\gamma, \Gamma]_{\lambda/2}(X)$	**Closings** $[\phi, \Phi]_{\lambda/2}(X)$
Vertex sets $X^\bullet$	even	$\gamma_{\lambda/2} = \delta_{\lambda/2} \circ \epsilon_{\lambda/2}$	$\phi_{\lambda/2} = \epsilon_{\lambda/2} \circ \delta_{\lambda/2}$
-	odd	$\gamma_{\lambda/2} = \Delta_{\lambda/2} \circ \epsilon_{\lambda/2}$	$\phi_{\lambda/2} = \varepsilon_{\lambda/2} \circ \delta_{\lambda/2}$
Edge sets $X^\times$	even	$\Gamma_{\lambda/2} = \Delta_{\lambda/2} \circ \varepsilon_{\lambda/2}$	$\Phi_{\lambda/2} = \varepsilon_{\lambda/2} \circ \Delta_{\lambda/2}$
-	odd	$\Gamma_{\lambda/2} = \delta_{\lambda/2} \circ \varepsilon_{\lambda/2}$	$\Phi_{\lambda/2} = \epsilon_{\lambda/2} \circ \Delta_{\lambda/2}$

Table 2.1: Summary of the openings and closings resulting from the dilations and erosions of Definition 13.

Given a graph $X(X^\bullet, X^\times)$ in $\mathcal{G}$, the operator $\gamma_{\lambda/2}$ (resp. $\Gamma_{\lambda/2}$) is opening on $X^\bullet$ (resp. $X^\times$), and $\phi_{\lambda/2}$ (resp. $\Phi_{\lambda/2}$) is closing on $X^\bullet$ (resp. $X^\times$). Furthermore, the simultaneous application of these operators for a same size parameter on the vertices and on the edges of a subgraph of $\mathbb{G}$ leads to a subgraph of $\mathbb{G}$, hence morphological filtering on subgraphs (opening $[\gamma, \Gamma]_{\lambda/2}(X)$ and closing $[\phi, \Phi]_{\lambda/2}(X)$).

From a morphological point of view, when $\mathbb{G}^\bullet$ is a subset of the grid point $\mathbb{Z}^d$ and the set of edges $\mathbb{G}^\times$ is derived from a symmetrical structuring element, then the vertex parts in the result of $[\gamma, \Gamma]_{\lambda/2}$ and $[\phi, \Phi]_{\lambda/2}$ correspond to the classical opening and closing of size $\lambda/2$ [14].

By further composing of openings and closings with increasing size parameter (these series of openings and closings, called granulometries), alternate sequential filters (ASF) are obtained. Let $\lambda \in \mathbb{N}$, the operators $ASF_{\lambda/2}$ is defined as follows:

$$ASF_{\lambda/2} = [\gamma, \Gamma]_{\lambda/2} \circ [\phi, \Phi]_{\lambda/2} \circ ASF_{(\lambda-1)/2} \text{ otherwise.} \tag{2.44}$$

A second family of the alternate sequential filter can be defined by replacing the sequence $[\gamma, \Gamma]_{\lambda/2} \circ [\phi, \Phi]_{\lambda/2}$ by the sequence $[\phi, \Phi]_{\lambda/2} \circ [\gamma, \Gamma]_{\lambda/2}$ (anti-granulometries). These families of filters generally outperform the results of elementary openings and closings in applications to image regularization.

As seen in this section to obtain efficient morphological filters on graphs, one needs

to consider iterated versions of elementary operators of dilations and erosions of size λ (Table 2.1). In following, we will provide definition of these iterated operators and we will study their behavior and computation.

2.5.3 Problem statement of iterated dilations and erosions on graphs

Let α be an operator acting on $\mathcal{G}^\bullet$ or on $\mathcal{G}^\times$ and let i be a non negative integer. The operator α^i is defined by the identity when $i = 0$ and by $\alpha \circ \alpha^{i-1}$ otherwise.

In other words, when i is greater than 1, the result of the operator α^i applied to a set X can be obtained by applying i iterations of α from X.

The elementary operators presented in Definition 10 map the elements of $\mathcal{G}^\bullet$ (*i.e.*, subsets of vertices) to those of $\mathcal{G}^\times$ (*i.e.*, subsets of edges) or the elements of $\mathcal{G}^\times$ to those of $\mathcal{G}^\bullet$. Thus, since the input and output sets of these operators are distinct, they cannot be directly iterated. However, any composition of an operator acting from $\mathcal{G}^\bullet$ to $\mathcal{G}^\times$ (resp. from $\mathcal{G}^\times$ to $\mathcal{G}^\bullet$) with an operator from $\mathcal{G}^\times$ to $\mathcal{G}^\bullet$ (resp.from $\mathcal{G}^\bullet$ to $\mathcal{G}^\times$) leads to an operator on $\mathcal{G}^\bullet$ (resp. on $\mathcal{G}^\times$). Then, such composition can be iterated and eventually followed again by an operator from $\mathcal{G}^\bullet$ to $\mathcal{G}^\times$ (resp. from $\mathcal{G}^\times$ to $\mathcal{G}^\bullet$). Therefore, to define iterated operators on graphs, we can distinguish two cases depending whether a final composition with an operator from $\mathcal{G}^\bullet$ to $\mathcal{G}^\times$ (resp. from $\mathcal{G}^\times$ to $\mathcal{G}^\bullet$) is considered or not.

Definition 13 (Iterated dilations/erosions)**.** *Let λ be a nonnegative integer.*

Case 1 (even values of λ). *If λ is even, we define the operators $\delta_{\lambda/2}$ and $\epsilon_{\lambda/2}$ on $\mathcal{G}^\bullet$ and the operators $\Delta_{\lambda/2}$ and $\varepsilon_{\lambda/2}$ on $\mathcal{G}^\times$ by:*

$$\delta_{\lambda/2}(X^\bullet) = (\delta^\bullet \circ \delta^\times)^{\lambda/2}(X^\bullet), \text{ for any } X^\bullet \subseteq \mathbb{G}^\bullet; \tag{2.45}$$

$$\epsilon_{\lambda/2}(X^\bullet) = (\epsilon^\bullet \circ \epsilon^\times)^{\lambda/2}(X^\bullet), \text{ for any } X^\bullet \subseteq \mathbb{G}^\bullet; \tag{2.46}$$

$$\Delta_{\lambda/2}(X^\times) = (\delta^\times \circ \delta^\bullet)^{\lambda/2}(X^\times), \text{ for any } X^\times \subseteq \mathbb{G}^\times; \text{ and} \tag{2.47}$$

$$\varepsilon_{\lambda/2}(X^\times) = (\epsilon^\times \circ \epsilon^\bullet)^{\lambda/2}(X^\times), \text{ for any } X^\times \subseteq \mathbb{G}^\times. \tag{2.48}$$

Case 2 (odd values of λ). *If λ is odd, we define the operators $\delta_{\lambda/2}$ and $\epsilon_{\lambda/2}$ from $\mathcal{G}^\bullet$ to $\mathcal{G}^\times$ and the operators $\Delta_{\lambda/2}$ and $\varepsilon_{\lambda/2}$ from $\mathcal{G}^\times$ to $\mathcal{G}^\bullet$ by:*

$$\delta_{\lambda/2}(X^\bullet) = \delta^\times \circ \delta_{(\lambda-1)/2}(X^\bullet), \text{ for any } X^\bullet \subseteq \mathbb{G}^\bullet; \tag{2.49}$$

$$\epsilon_{\lambda/2}(X^\bullet) = \epsilon^\times \circ \epsilon_{(\lambda-1)/2}(X^\bullet), \text{ for any } X^\bullet \subseteq \mathbb{G}^\bullet; \tag{2.50}$$

$$\Delta_{\lambda/2}(X^\times) = \delta^\bullet \circ \Delta_{(\lambda-1)/2}(X^\times), \text{ for any } X^\times \subseteq \mathbb{G}^\times; \text{ and} \tag{2.51}$$

$$\varepsilon_{\lambda/2}(X^\times) = \epsilon^\bullet \circ \varepsilon_{(\lambda-1)/2}(X^\times), \text{ for any } X^\times \subseteq \mathbb{G}^\times. \tag{2.52}$$

The operators $\epsilon_{\lambda/2}$, $\delta_{\lambda/2}$, $\varepsilon_{\lambda/2}$, and $\Delta_{\lambda/2}$ are illustrated for various (even and odd) values of λ in Figures 2.14, 2.15, 2.16, and 2.17, respectively. In particular, it can be observed

that, when λ is even, the operators $\epsilon_{\lambda/2}$ and $\delta_{\lambda/2}$ map a set vertices to a set of vertices and that the operators $\varepsilon_{\lambda/2}$ and $\Delta_{\lambda/2}$ map a set of edges to a set of edges. On the other hand, when λ is odd, the operators $\epsilon_{\lambda/2}$ and $\delta_{\lambda/2}$ map a set of edges to a set of vertices and the operators $\varepsilon_{\lambda/2}$ and $\Delta_{\lambda/2}$ map a set of vertices to a set of edges.

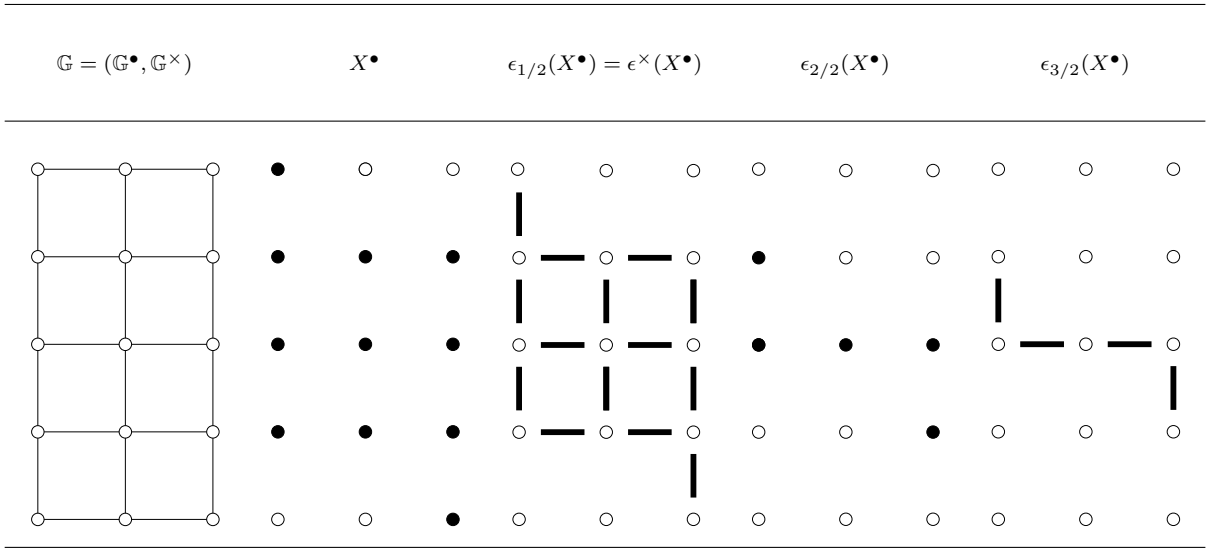


Figure 2.14: Illustration of the operator $\epsilon_{\lambda/2}$ with $\lambda \in \{1, 2, 3\}$. The considered sets are depicted by black dots or line segments.

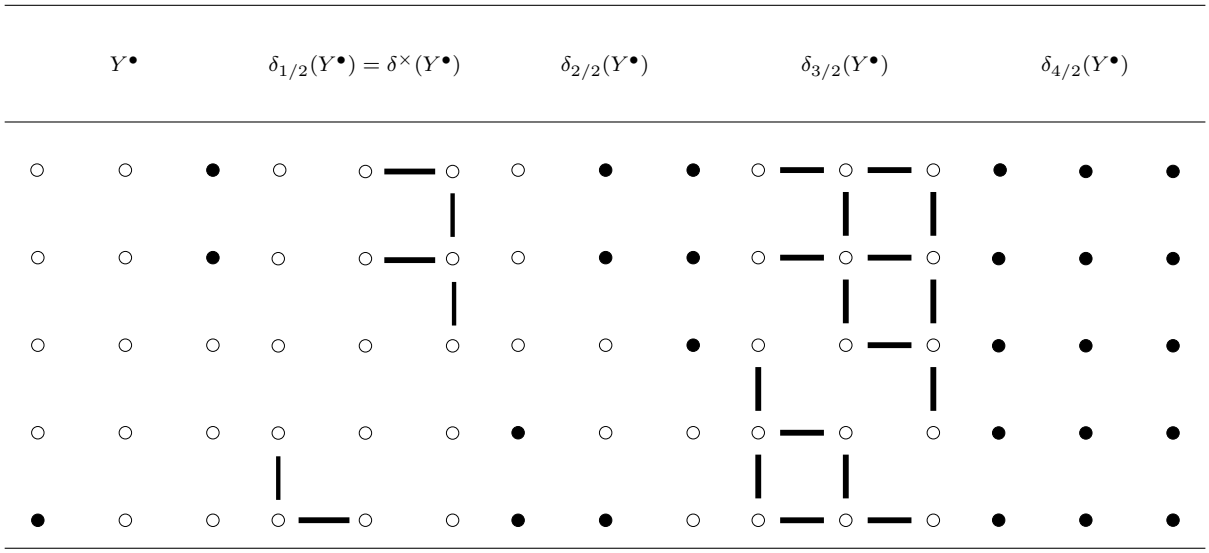


Figure 2.15: Illustration of the operator $\delta_{\lambda/2}$ with $\lambda \in \{1, 2, 3, 4\}$, the workspace being the graph $\mathbb{G}$ depicted in Figure 2.14. The considered sets are depicted by black dots or line segments.

The naive computation of iterated operators presented here consists of performing λ iteration of elementary operators. When the complexity of elementary operator is $O(n)$ where n is the size of the underlying graph. Thus, the time-complexity of the associated

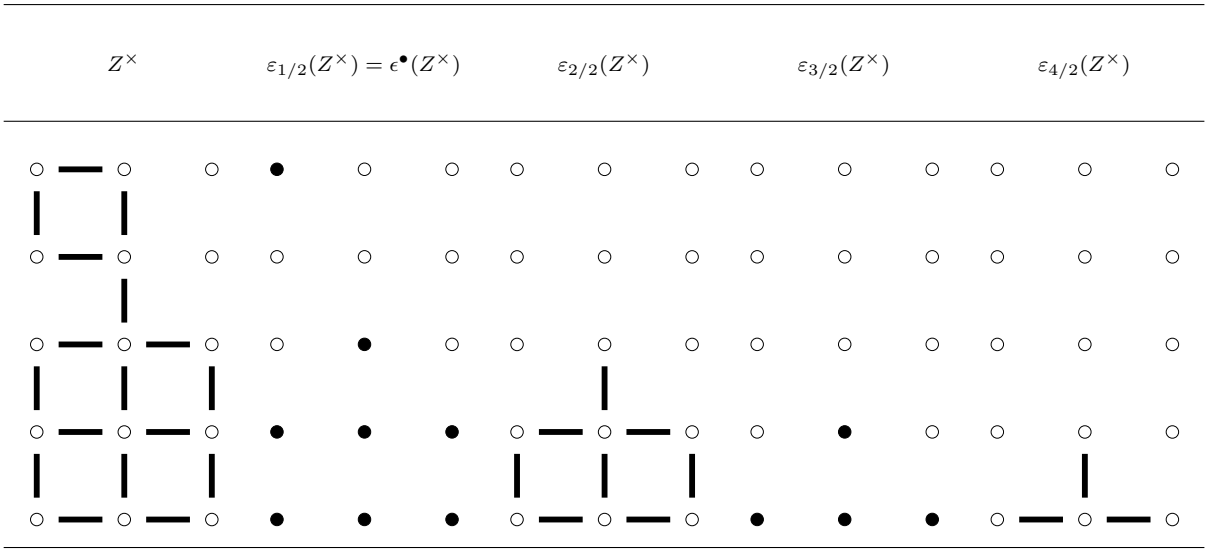


Figure 2.16: Illustration of the operator $\varepsilon_{\lambda/2}$ with $\lambda \in \{1, 2, 3, 4\}$, the workspace being the graph $\mathbb{G}$ depicted in Figure 2.14. The considered sets are depicted by black dots or line segments.

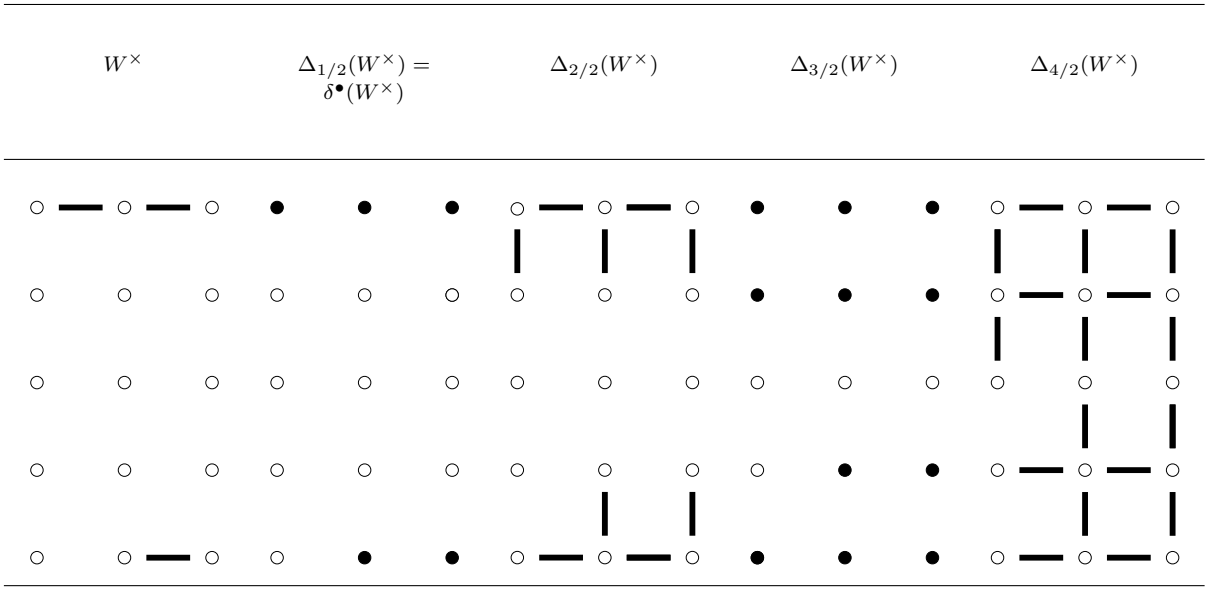


Figure 2.17: Illustration of the operator $\Delta_{\lambda/2}$ with $\lambda \in \{1, 2, 3, 4\}$, the workspace being the graph $\mathbb{G}$ depicted in Figure 2.14. The considered sets are depicted by black dots or line segments.

algorithm increases with the size parameter. More precisely, for a parameter value of λ the algorithm runs in $O(\lambda.n)$ time, where n is the size of the underlying graph. Therefore, there is a dependence between the size parameter λ and the time-complexity.

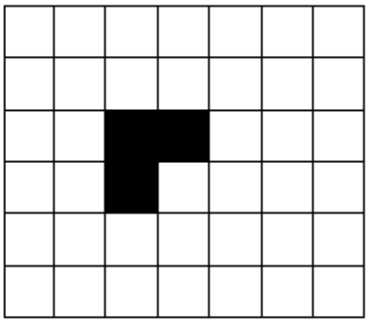
Eight iterated morphological operators on graphs are presented in Definition 13. When dealing with even λ value, the inputs and outputs sets of the two first operators $\delta_{\lambda/2}$ and $\epsilon_{\lambda/2}$ are subsets of vertices of graph $\mathbb{G}$. From a mathematical morphology point of view,

these operators (*i.e.* $\{\delta_{\lambda/2}, \epsilon_{\lambda/2} \mid \lambda \text{ is even }\}$) amount exactly to the classical operators of dilation and erosion defined with symmetric structuring elements that were first studied in [58] for which efficient algorithms exist. These algorithms were introduced by L.Vincent in [58] where a link is established between these two operators and notion of distance map. Such as, the result of these operators can be obtained using interesting property of thresholding of distance map. This link leads to efficiently compute the results of the corresponding dilation and erosion. Contrary, there exist no efficient algorithms for the six remaining iterated operators defined above.

A big challenge in this framework is to obtain efficient and fast computation of the morphological filtering on graphs [14]. For this aim, as a starting point, we need to introduce in our work new distance maps on graphs that allow to obtain linear-time algorithms for the all operators defined in Definition 13. Then, we need also to propose a parallel strategy in order to obtain fast implementation of these distance maps, hence all these morphological operators.

2.6 Distance Map on graphs

The distance map (also called distance transformation) plays an important role with a wide range of applications in image processing, computer vision, graphics and computational geometry, pattern recognition [16]. It is an interesting tool in image analysis and mathematical morphology. Distance map is used for computing the medial axis of digital shapes, and for comparing binary images [20]. Consider a digital binary image, a distance transformation is an operation that converts this binary image to a grey-level image where all pixels have a value corresponding to the distance to the nearest feature pixel [9]. An example of distance map is shown in Figure 2.18.



4	3	2	2	3	4	5
3	2	1	1	2	3	4
2	1	0	0	1	2	3
2	1	0	1	2	3	4
3	2	1	2	3	4	5
4	3	2	3	4	5	6

Figure 2.18: Example of a distance map. To the left is a binary image. To the right is the resulting image using Euclidean distance.

The aim of distance map is to provide the distance from each point to the nearest point. Several distance functions (metrics) have been used for digital image processing

such as euclidean distance, Manhattan distance, Tchebychev distance, city block distance, chessboard distance, chamfer distance and octagonal distance etc.

When the workspace is a graph $\mathbb{G}$, an elementary use of paths is the computation of a distance map: from any vertex of a graph, one can compute the distance (length of the shortest path) to the nearest obstacle vertex. The distance between these vertices can be defined as follows:

Definition 14 (Graph distance)**.** *Let x and y be two vertices in undirected graph $\mathbb{G}$. The graph distance D between x and y is defined by the length of the shortest paths connecting x to y in $\mathbb{G}$ [29]:*

$$\forall x, y \in \mathbb{G}, D(x, y) = min\{L(\pi) \mid \pi \text{ is a path from } x \text{ to } y \text{ in } \mathbb{G} \text{ and from } y \text{ to } x\} \quad (2.53)$$

In the case of undirected graph, the distance D is called geodesic [29] which have the same proprieties of the usual distance [51].

Property 15. *The map D is a geodesic distance on $\mathbb{G}$ that satisfy:*

1. $\forall x \in \mathbb{G}, D(x, x) = 0$ *(identity),*
2. $\forall x, y \in \mathbb{G}, x \neq y \Rightarrow D(x, y) > 0$ *(Positive),*
3. $D(x, y) = D(y, x)$ *(symmetry),*
4. $D(x, y) = 0 \Leftrightarrow x = y$ *(defined) and*
5. $\forall x, y, and\ z \in \mathbb{G}, D(x, z) \leq D(x, y) + D(y, z)$ *(triangle inequality).*

Definition 16 (Distance map on graph)**.** *Given a subset $X \subseteq \mathbb{G}$ and D a distance on X. The distance map to X is the map D_X from X into $\mathbb{R}$ is defined as follows:*

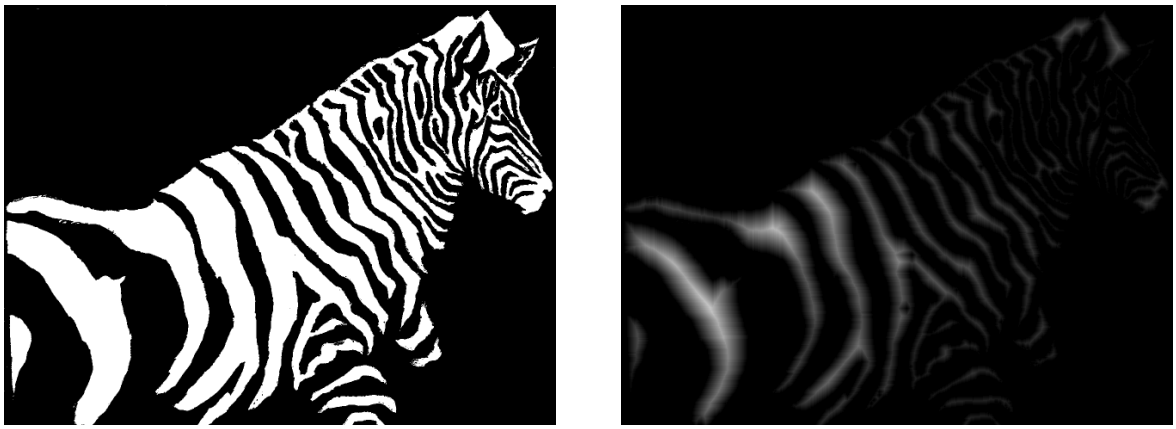
$$\forall y \in X, D_X(y) = \min\{D(x, y) \mid x \in X\}. \quad (2.54)$$

In other words, for any element of X, one can compute the geodesic distance (*i.e.*, the length of a shortest path) to the closest element in X (see example in Figure 2.6).

As seen above, the computation of distance map in unweighted graph can be reduced to the computation of shortest paths. To this end, we study the shortest-path problem in the following section.

2.6.1 Shortest Paths Problem

The shortest-path problem is one of classical well-studied problems in graph theory that has many applications include general traveling problems, path finding in social networks, roads networks optimization, computer games, maze problems, etc. In fact, this problem deals with finding the optimal paths from a source to a destination with respect to a



(b) Original object X in black (b) D_X in the graph induced by the 4-adjacency

Figure 2.19: Illustration of distance map on graph.

distance function. In the following, we review algorithms that are able to solve the shortest path problem.

Dijkstra's algorithm [19] has many variants but the most common one is to compute all shortest paths between a source vertex and all other vertices of a weighted graph with positive real numbers. For a given single source vertex in the graph, the algorithm finds the paths with smallest lengths between the source vertex and every other vertex. the Dijkstra's algorithm visits first the neighbors of the source vertex, and inserts them in an ordered priority queue depending on their distances. Then, in each iteration neighbors of the vertices in queue are analyzed, classified and inserted in the queue. The calculation stops when all vertices of the graph are visited. The complexity of this algorithm is $O(|V|^2)$.

Fredman and Tarjan [23] improve Dijkstra's algorithm using Fibonacci heap in order to implement the priority queue, the complexity can be easily reduced to $O(|E|.|V|log(|V|))$ where $|E|$ is the number of vertices and $|V|$ is the number of edges in the graph .

The breadth-first search(BFS) algorithm [34] can be simply employed to solve the single-source shortest path problem for both directed and undirected graphs. Given an unweighted graph and a source vertex, BFS explores a graph level by level in order to find every vertex reachable to the source vertex. This algorithm explores all non-visited vertices (the neighboring vertices) at level (distance) K from the source vertex x, and, then discovers the next level of vertices at distance $k+1$. In fact, all nodes at level k are expanded before any nodes at level $k+1$. An auxiliary queue data structure managed with a FIFO (First-In-First-Out) property is used to store vertices. In this search, a BFS tree is built with the source vertex x as a root of the tree where for any vertices in this BFS tree, the shortest path from the source vertex x to any vertex is found in a BFS tree. The algorithm stops when there are no vertices in the queue. The complexity of the BFS search is $O(|V|+|E|)$.

Bellman Ford's algorithm [22] is used to find the shortest paths from the single-source

vertex to all other vertices in a weighted graph. The idea of this algorithm is similar to Dijkstra's, it is slower than Dijkstra's algorithm but able to process weighted graph in which edges can have negative weights. It deals by overestimating the length of the path from the source vertex to all other vertices in graph. Then, it iteratively checks if there is an edge which can shorten the calculated shortest path to vertices. After $n-1$ iterations at most, the algorithm returns an array of distances to the reachable vertices. The complexity of Ford-Bellman algorithm is of order of $O(n.m)$ where n and m are the number of vertices and m the number of edges in graph respectively.

Floyd-Warshall's Algorithm [21] is used to find the shortest paths between all pairs of vertices in weighted graph with positive or negative weighted edges. The idea of this algorithm consists of the comparison of all possible paths in graph between each pair of vertices. Thus, the algorithm can calculate all the shortest paths between all pairs vertices in $O(V^3)$, where V is the number of vertices in a graph.

The shortest-path algorithms presented above are divided into two categories [59, 37]. The first one is single source shortest-path, where the idea is to find the shortest-paths from a given single-source vertex to all other vertices in graph. Whereas, the second one is all-pairs shortest-path that find the shortest-paths between all pairs of vertices in a graph. Furthermore, these algorithms processed either weighted or unweighted graphs. It seems that the breadth-first search(BFS) algorithm [34] present a classical approach in graph-based problems that we are interested for our work in particular for (unweighted, undirected) graphs. In fact, it is one of the simplest algorithms for exploring a graph. This exploration strategy is achieved using a suitable data structure. Then, the BFS is considered as queue-based algorithm in linear-time.

2.7 Introduction to Parallel Computing

The aim of the parallel computers is to speed up computations by using multiple processors "CPUs", or to perform larger computations which are not possible on a single processor machines. Multicore and multithreaded CPUs architectures have become an increasingly popular way to implement dynamic, highly asynchronous, concurrent programs. They both exploit concurrency by executing multiple threads at the more fine-grained instruction level. Generally, we can divide parallel computing architectures in two dominant classes with respect to memory layout. The first one is the class of distributed memory machines. The second one is that of shared memory machines.

2.7.1 Distributed Memory Machines

In distributed memory architecture, each processor has its own local memory. All variables of a program are private *i.e.* they locate in the local memory (or cache) of a processor

Figure 2.20. The distributed memory paradigms assume that the computing infrastructure is composed of multiple nodes with distinct memory address spaces. Each compute node can only directly reference its own memory. The communication of data occurs through discrete messages sent from process to process. For communication between processors a message passing library is used. Two commonly used libraries are MPI (Message Passing Interface, or PVM (Parallel Virtual Machine).

The Message Passing Interface (MPI) is the standard language for parallel programs on distributed memory systems. It is a specification for message passing operations which is provided as a library for C, C++ and Fortran [31]. It is based on explicit message passing and collective communications that have non-trivial consequences for performance.

The PVM message-passing interface is designed to allow a network of heterogeneous UNIX machines to be used as a single, large parallel computer. The interface is implemented by the PVM system, which consists of a daemon process (pvmd) that runs on each workstation, and a run-time library (pvmlib) that contains the PVM interface routines [32].

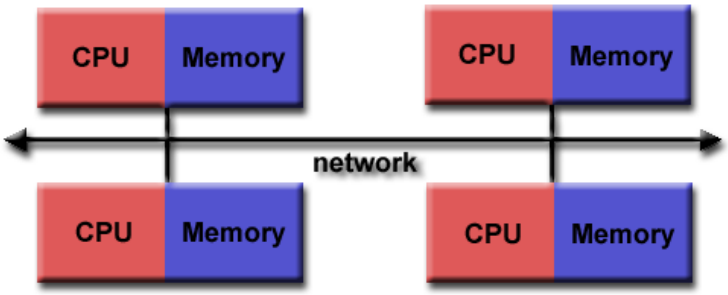


Figure 2.20: Left: Distributed memory architecture.

2.7.2 Shared Memory Architectures

In shared memory architecture every processor has an access to all of the memory *i.e.* there is a shared address space Figure 2.21. The shared memory programming model is often achieved with some form of multithreading, where multiple threads on the same node share the same address space. Threads can easily communicate because they operate in the same address space, so explicit synchronization must be used to synchronize the simultaneously access on shared variables. OpenMP (Open Multi-Processing) and POSIX Threads (Pthreads) are two of the most widely used solutions for shared memory programming.

OpenMP consists of a set of compiler directives. It is a directive-based extension to C, C++ and Fortran languages commonly used in high performance computing. Moreover, OpenMP is becoming more important when the number of cores per system increases. Due to the higher level specification of parallelism with OpenMP, the gain in productivity is

that OpenMP programmers do not have to concern themselves with such details as when or how to create new threads, how to distribute their data between these threads and how to synchronize the computation itself between the threads [31, 25].

The POSIX Threads is a set of C programming language types and procedure calls. Various benefits exist to using Pthreads such as the simplicity, the flexibility, and the portability. Pthreads library provides various primitives for synchronizing concurrent computations and memory accesses of threads. In particular, it provides implementations of mutexes and semaphores [10] for Linux. A mutex is a variable which is either locked or unlocked and can be handled with atomic (non-interruptible) instructions, ensuring that, at a given time, a single thread is accessing to the variable with the atomic instruction. It is mainly used to ensure exclusive access to data shared between threads, whereas semaphores generalize mutexes by allowing a limited number, but possibly greater than one, to access concurrently to critical data [31, 25].

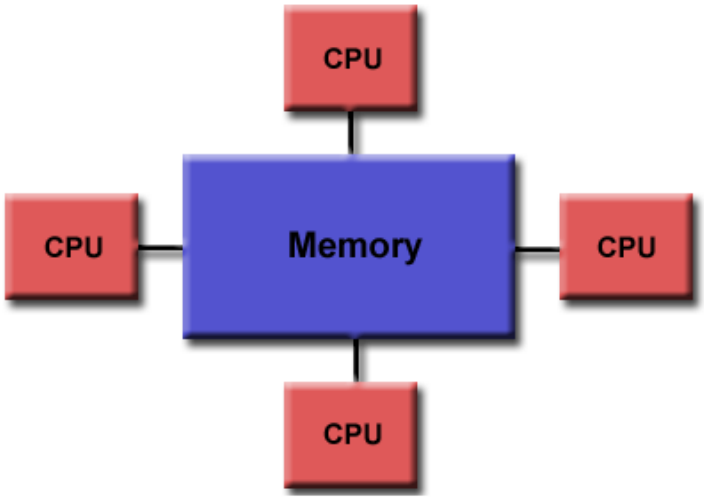


Figure 2.21: Shared memory architecture.

2.7.3 Performance Models of Parallel Computing Systems

It is important to study the performance of an algorithm parallel in order to assess the gain of parallel implementation determine the best algorithm, evaluate hardware platforms, and examine the benefits obtained from parallelism. A number of metrics have been considered for evaluating performance in parallel computing such as execution time, speedup and efficiency.

2.7.3.1 Execution Time

The sequential running time of a program is the time elapsed between the beginning and the end of its execution on a single processor, using the fastest known sequential algorithm.

Whereas, the parallel running time is the execution time of parallel parts of algorithms using a parallel processor. We denote the sequential running time by $T(1)$ and the parallel running time by $T(p)$.

2.7.3.2 Speedup and efficiency

Speedup is an important measure of the quality of a parallel program. This measure expresses how many times a parallel program works faster than a sequential one, where both programs are solving the same problem. For a given number p of processors, the speedup $S(p)$ is the ratio of the execution time $T(1)$ of a program on a single processor over the execution time $T(p)$ obtained with p processors: $S(p) = T(1)/T(p)$. For p processors, the speedup is bounded by the the number of processors: $S(p) \leq p$. The maximum value of speedup that can be obtained is equal to the number of processors $S(p) = p$. This maximum speedup value could be achieved in an ideal multiprocessor system where there are no communication costs and the workload of processors is balanced (although in practice this is rarely achieved). In such a system, every processor needs $T(1)/p$ time units in order to complete its job [3].

One can consider speedup to analyze algorithms either theoretically using asymptotic time complexity or in practice by measuring the execution times of a program.

Another frequently used measure of performance is the efficiency of a parallel program. The efficiency of a program yields the same information as the speedup of a program, since it is computed as: $E(p) = S(p)/p * 100$.

2.8 Overview of parallel implementation for distance map

Literary, several algorithms for fast computing of distance maps on images have been proposed. For instance, Shyu *et al.*[55] proposed an efficient parallel algorithm to compute the Chamfer distance transformation of a binary image. Shyu *et al.*'s parallel implementation requires the decomposition of the input image into bands, each band is distributed to a processor. In each processor, the distance transformation can be computed by preforming two passes over the image : a forward pass to propagate the distance transform from the causal neighbors, followed by a backward pass to propagate the distance transform from anti-causal neighbors. This method computes the distance transformation on a distributed system. Therefore, the intermediate results across several processors must be synchronized using Message Passing Interface (MPI). Pham *et al.*[47] presented a parallel implementation of geodesic distance transformation using OpenMP. In fact, this implementation is based on the parallel execution of the sequential Chamfer distance transformation proposed in [55] using the shared memory model on multicore CPUs. The parallel implementation of this

distance transformation requires more than one iterations of forward and backward passes unlike the Chamfer distance transformation. As a result, the geodesic distance transformation can be propagated from one band to the next in following iteration contrary to [55]. Man *et al.*[38] [39] presented a parallel algorithm for computing Euclidean distance map on both multicore processors and GPU system. The idea of this algorithm consists of performing two steps, in each step both a forward and backward scan is performed. In fact, The input image is partitioned into rows, columns where in the first step columns are scanned and in the second rows are scanned. The scanning of a particular column (resp. row) is independent to the scanning of the others columns (resp. rows). The work of [52] described a separable algorithm to efficiently compute the distance transformation, the separability means that the computations are performed dimension by dimension.

These works are all based on the regular structure of the space. Such parallel computation of distance map uses a static partitioning of space into rows, columns, or blocks processed in parallel. This means that this computation is based on a number of data predefined a prior.

In this work, we aim to develop a new parallel strategy adapted for the new distance map on graphs proposed in the next chapter. In order to generalize our strategy, and its implementations, we have to consider both regular and irregular space.

2.9 Conclusion

After reading the description of the framework of mathematical morphology on graphs [14] presented in this chapter, we can claim that this framework considers four elementary morphological operators to go from one kind of sets to another one. More preciously, these operators map a set of vertices to a set of edges and a set of edges to a set of vertices. Interesting morphological filters on graphs can be obtained by iterated version of these elementary operators (composition). These filters are parameterized by an integer value related to a notion of size of the features to be preserved or removed. Larger structures are removed/preserved when higher numbers of iterations of basic operators are considered. To this end, we discussed in this chapter the problem of computation of iterated morphological operators on graphs. The naive computation of iterated operators consists of performing λ iteration of elementary operators. So, there is a dependence between the size parameter λ and the time-complexity.

However, the behavior of iterated operators depends on the parity of a size parameter called λ. Thus, we distinguished two cases even and odd values of λ. When λ is even, the inputs and outputs sets of the operators is the same which are subsets of vertices or subsets of edges of graph. In this case there exist efficient algorithms for the two operators acting on subsets of vertice which amount to classical morphological operators. These algorithms were introduced by L.Vincent in [58] where a link is established between distance map

and these two operators. When λ is odd, the inputs and the outputs of the operators are distinct: one of them is subset of edges whereas the other one is a subset of vertices. In this case there exist no efficient algorithms for the six remaining iterated operators.

For this, we provide a reminder about notions of distance map on graphs. Because of the computation of distance map is reduced to the computation of shortest paths, we discuss in this chapter the algorithms related to the problem of shortest path. We are interested in our work with the breadth-first search(BFS) algorithm which is one of the simplest algorithms for exploring a (unweighted, undirected)graphs and runs in linear-time.

Finally, to obtain fast computation of distance map, we presented an overview on parallel computing. In especial, we surveyed the multicore and multithreaded CPUs architectures and the performance models of this system, since they are the target architecture and the performance models used in the parallel strategy proposed in this work. we also presented some related work about parallelization of distance map on the regular structure of the space, where, the parallel computation uses a static partitioning of space into rows, columns, or blocks processed in parallel.

In the following of this manuscript, we present how new distance maps are established links with all operators of Definition 13, and are used to propose efficient sequential and parallel algorithms for computing the results of the morphological operators of [14].

Chapter 3

Distance maps for Iterated morphological operators on graphs

Contents

3.1 Introduction

In this chapter, we propose three original distance maps on graphs that called edge-edge, edge-vertex, and vertex-edge distance maps. Given a set of edges, the edge-edge (resp. edge-vertex) distance map provides for each edge (resp. each vertex) of the graph a geodesic distance to the closest edge in the input set. Given a set of vertices, the vertex-edge distance map provides for each edge a geodesic distance to the closest vertex in the

input set. In addition, we present our vertex-vertex distance map based on new notion of length of path which considers both the numbers of vertices and of edges along the path. The vertex-vertex distance map provides for each vertex a geodesic distance to the closest vertex in the input (a set of vertices).

Then, we prove by theorems that all dilations and erosions on graphs presented in Definition 13 in chapter 2 can be characterized with the proposed distance maps. Indeed, the edge-vertex and the vertex-edge distance maps allow us to characterize (by threshoding) the four operators defined when λ is odd. The vertex-vertex distance map allows us to characterize (by thresholding) the two first operators acting on vertices which amount to classical morphology, and the edge-edge distance map allows us to characterize (by thresholding) the two remaining operators acting on edges, when λ is even.

After establishing the links between the proposed distance maps and the operators of Definition 13, in order to efficiently compute these proposed distance maps, we present in a later part variations of linear-time algorithms for distance maps in unweighted graphs which derive from breadth first search. Thus, the dilations and erosions of [14] can be also obtained by linear-time algorithms.

3.2 Proposed distance maps

In this section we define the proposed distance maps on graphs in order to characterize the iterated morphological operators on graphs. In each kind of these distance maps, we start by presenting notions of paths and shortest paths that are necessary to define the proposed distance maps.

Important remark. As seen in the previous chapter, in standard graph textbooks, the length of a path is often considered as the number of edges along the path, whereas, our contributions in this thesis consider both the numbers of vertices and of edges along the path are considered. It can be remarked that the two notions of length are equivalent up to a factor 2. The choice of multiplying by 2 the usual length of paths directly links the length of paths to the operators defined in chapter 2 in Section 2.5.3.

3.2.1 Edge-edge distance maps on graphs

- An **edge-edge path**: let u and v be two edges in $\mathbb{G}^{\times}$. A *(edge-edge) path from u to v* is a sequence $(u_0, x_0, \ldots, u_{\ell-1}, x_{\ell-1}, u_\ell)$ such that $u_0 = u$, $u_\ell = v$, $x_0 \in u_0$, $x_{\ell-1} \in u_\ell$, and $(x_0, u_1, \ldots, u_{\ell-1}, x_{\ell-1})$ is a vertex-vertex path from x_0 to $x_{\ell-1}$.
- The ***length*** of an edge-edge path $(u_0, x_1, \ldots, u_{\ell-1}, x_{\ell-1}, u_\ell)$ is the number of its elements minus one. Hence, it is equal to the integer value 2ℓ.
  Figure 3.1 illustrates in red an example of an edge-edge path {{a,b}, b, {b,d}, d, {d,e}} of length equal to 4.

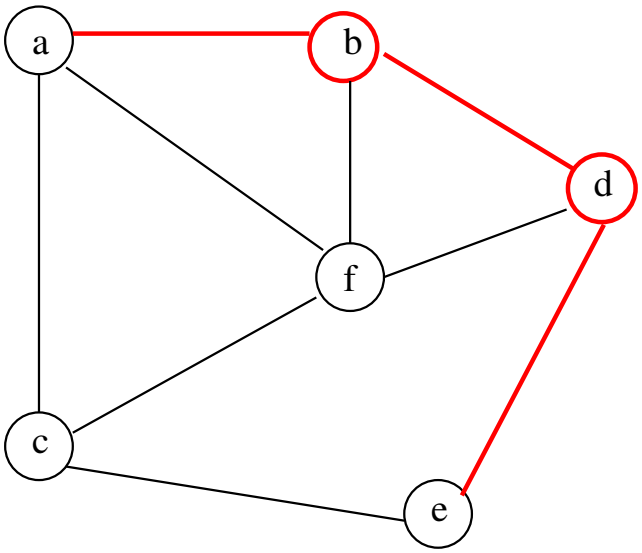


Figure 3.1: An edge-edge path.

– An **Edge-edge shortest path**: let e_1 and e_2 be two edges in $\mathbb{G}^\times$. A *shortest path from an edge e_1 to another edge e_2* is a path of minimal length from e_1 to e_2. We denoted by $L(e_1, e_2)$ the length of a shortest path from e_1 to e_2.

Let us now define the edge-edge distance map.

Definition 17 (Edge-edge distance map). *Let $X^\times$ be a subset of $\mathbb{G}$. The (edge-edge) distance map to $X^\times$ is the map $D_{X^\times}^{(\times,\times)}$ from $\mathbb{G}^\times$ to the set of integers such that:*

$$D_{X^\times}^{(\times,\times)}(u) = \min\{L(v,u) \mid v \in X^\times\}, \text{ for any } u \in X^\times. \tag{3.1}$$

In other words, given a subset $X^\times$ of edges, the edge-edge distance map to $X^\times$ provides for each edge u of $\mathbb{G}$ the minimal length of a path from u to an edge of $X^\times$. An illustration of edge-edge distance map $D_{X^\times}^{(\times,\times)}$ is provided in Figure 3.2.

3.2.2 Vertex-edge distance maps on graphs

– A **vertex-edge path**: let x be a vertex in $\mathbb{G}^\bullet$ and u be an edge in $\mathbb{G}^\times$. A *(vertex-edge) path from x to u* is a sequence $(x_0, u_0, \ldots, x_\ell, u_\ell)$ such that $x_0 = x$, $u_\ell = u$, $x_\ell \in u_\ell$, and $(x_0, u_0, \ldots, x_\ell)$ is a vertex-vertex path from x_0 to x_ℓ.

– As we said in the previous section, the ***length*** of a path is the number of its elements minus one. So, the length of a vertex-edge path $(x_0, u_0, \ldots, x_\ell, u_\ell)$ is equal to the integer value $2\ell + 1$.

We illustrate in red in Figure 3.3 an example of a vertex-edge path {a, {a,b}, b, {b,d}, d, {d,e}} of length equal to 5.

– A **Vertex-edge shortest path**: let e_1 be a vertex in $\mathbb{G}^\bullet$ and e_2 be an edge in $\mathbb{G}^\times$. A *shortest path from a vertex e_1 to an edge e_2* is a path of minimal length from e_1 to

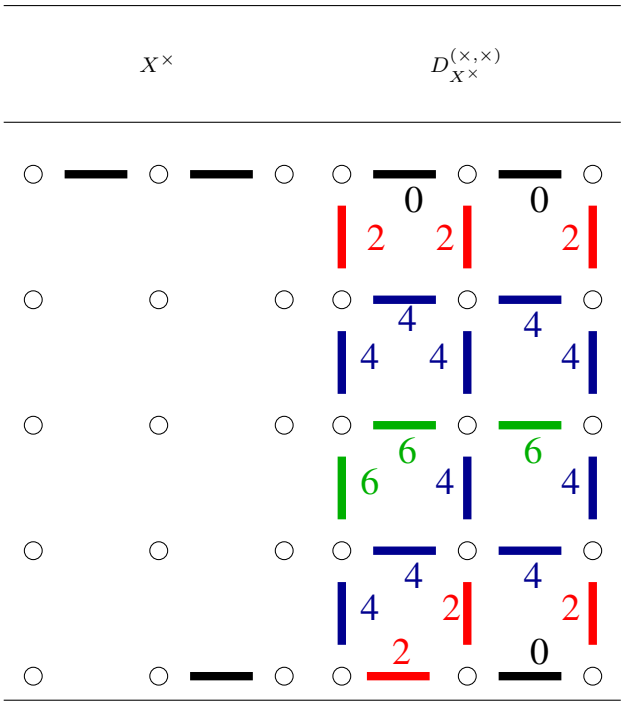


Figure 3.2: Illustration of Edge-edge distance map on graphs. The first subfigure shows the set of edges $X^\times$ in the graph $\mathbb{G}$. The second subfigure shows the edge-edge distance maps to the set $X^\times$.

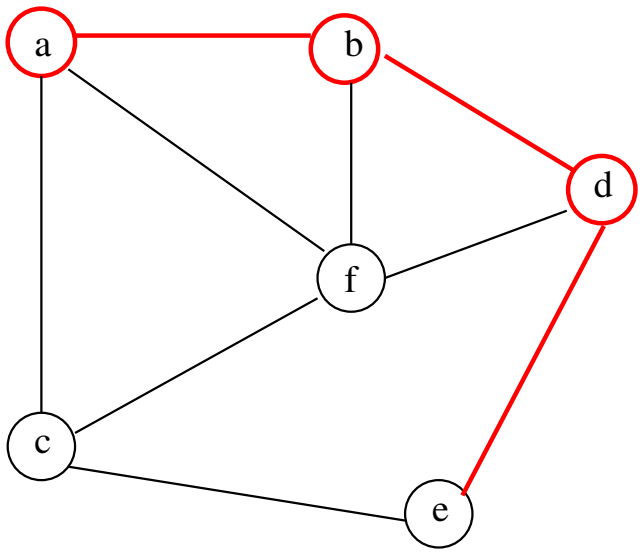


Figure 3.3: A vertex-edge path.

e_2. We denoted by $L(e_1, e_2)$ the length of a shortest path from e_1 to e_2.

Let us now define the Vertex-edge distance maps.

Definition 18 (Vertex-edge distance map). *Let $X^\times$ and $X^\bullet$ be two subsets of $\mathbb{G}$. The (vertex-edge) distance map to $X^\bullet$ is the map $D^{(\bullet,\times)}_{X^\bullet}$ from $\mathbb{G}^\times$ to the set of integers such*

that:

$$D_{X^\bullet}^{(\bullet;\times)}(u) = \min\{L(x,u) \mid x \in X^\bullet\}, \textit{for any } u \in X^\times. \tag{3.2}$$

In other words, given a subset $X^\bullet$ of vertices, the vertex-edge distance map to $X^\bullet$ provides for each edge u of $\mathbb{G}$ the minimal length of a path from u to a vertex in $X^\bullet$.
An illustration of vertex-edge distance map $D_{X^\bullet}^{(\bullet;\times)}$ is provided in Figure 3.4.

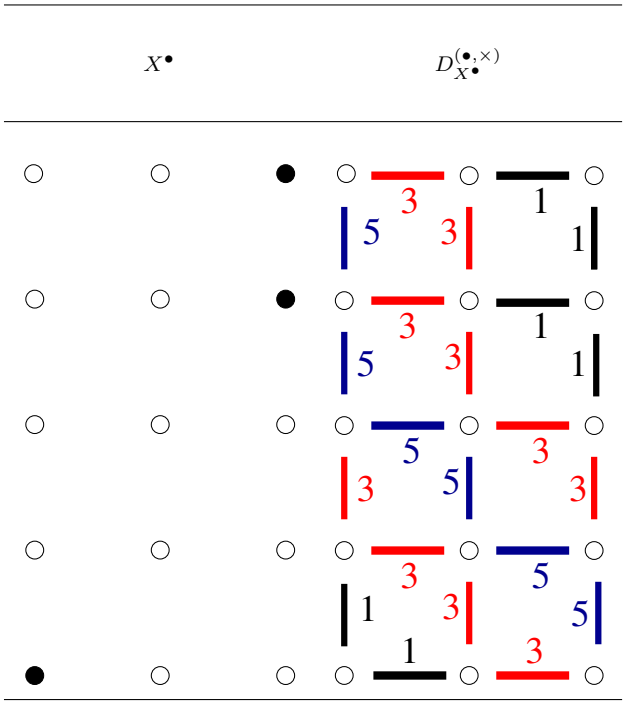


Figure 3.4: Illustration of Vertex-edge distance map on graphs. The first subfigure shows the set of vertices $X^\bullet$ in the graph $\mathbb{G}$. The second subfigure shows the vertex-edge distance maps to the set $X^\bullet$.

3.2.3 Edge-vertex distance maps on graphs

- An **edge-vertex path**: let u be an edge in $\mathbb{G}^\times$ and let x be a vertex in $\mathbb{G}^\bullet$. A *(edge-vertex) path from u to x* is a sequence $(u_0, x_0, \ldots, u_\ell, x_\ell)$ such that $(x_\ell, u_\ell, \ldots, x_0, u_0)$ is a vertex-edge path from x to u.
- The **length** of an edge-vertex path $(u_0, x_0, \ldots, u_\ell, x_\ell)$ is the number of its elements minus one which is equal to the integer value $2\ell + 1$.
  We illustrate in red in Figure 3.5 an example of an edge-vertex path { {a,f}, f, {f,d}, d, {d,e}, e} of length equal to 5.
- An Edge-vertex shortest path: let e_1 be an edge in $\mathbb{G}^\times$ and e_2 be a vertex in $\mathbb{G}^\bullet$. A *shortest path from a vertex e_1 to an edge e_2* is a path of minimal length from e_1 to e_2. We denoted by $L(e_1, e_2)$ the length of a shortest path from e_1 to e_2.

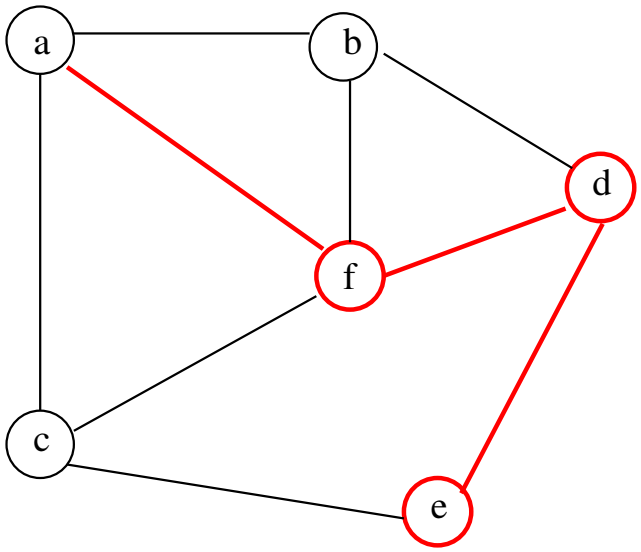


Figure 3.5: An edge-vertex path.

Let us now present the vertex-edge distance maps.

Definition 19 (Edge-vertex distance map). *Let $X^\times$ and $X^\bullet$ be two subsets of $\mathbb{G}$. The (edge-vertex) distance map to $X^\times$ is the map $D^{(\times,\bullet)}_{X^\times}$ from $\mathbb{G}^\bullet$ to the set of integers such that:*

$$D^{(\times,\bullet)}_{X^\times}(x) = \min\{L(u,x) \mid u \in X^\times\}, \text{ for any } x \in X^\bullet. \tag{3.3}$$

In other words, given a subset $X^\times$ of edges, the edge-vertex distance map to $X^\times$ provides for each vertex x of $\mathbb{G}$ the minimal length of a path from x to an edge of $X^\times$. An illustration of edge-vertex distance map $D^{(\times,\bullet)}_{X^\times}$ is provided in Figure 3.6.

3.2.4 Vertex-Vertex Distance map on graphs

- A **vertex-vertex path**: let x and y be two vertices in $\mathbb{G}^\bullet$. A *(vertex-vertex) path from x to y* is a sequence $(x_0, u_0, \ldots, x_{\ell-1}, u_{\ell-1}, x_\ell)$ such that $x_0 = x$, $x_\ell = y$, and, for any i in $\{0, \ldots, \ell-1\}$, and we have $u_i = \{x_i, x_{i+1}\}$ where u_i is an edge of $\mathbb{G}$.
- The **length** of a vertex-vertex path $(x_0, u_0, \ldots, x_{\ell-1}, u_{\ell-1}, x_\ell)$ in our proposition is considered as the number of its elements minus one, *i.e.*, the integer value 2ℓ. Observe that the length of any vertex-vertex path is even, which is not usual standard graph textbooks.
  Figure 3.7 illustrates in red an example of a vertex-vertex path { a, {a,f}, f, {f,d}, d, {d,e}, e} of length equal to 6.
- A **Vertex-vertex shortest path**: let e_1 and e_2 be two vertices in $\mathbb{G}^\bullet$. A *shortest path from a vertex e_1 to another vertex e_2* is a path of minimal length from e_1 to e_2. We denote by $L(e_1, e_2)$ the length of a shortest path from e_1 to e_2.

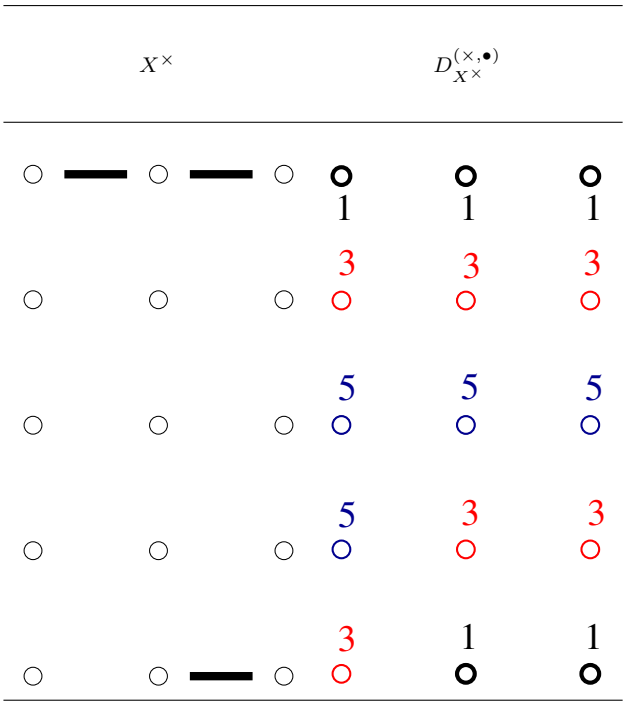


Figure 3.6: Illustration of edge-vertex distance map on graphs. The first subfigure shows the set of edges $X^{\times}$ in the graph $\mathbb{G}$. The second subfigure shows the edge-vertex distance maps to the set $X^{\times}$.

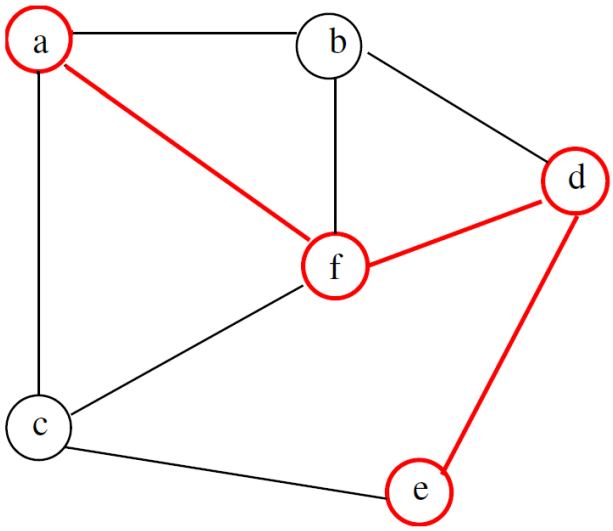


Figure 3.7: A vertex-vertex path.

Definition 20 (vertex-vertex distance map). *Let $X^{\bullet}$ be a subset of $\mathbb{G}$. The* (vertex-vertex) distance map to $X^{\bullet}$ *is the map $D_{X^{\bullet}}^{(\bullet,\bullet)}$ from $\mathbb{G}^{\bullet}$ to the set of integers such that:*

$$D_{X^{\bullet}}^{(\bullet,\bullet)}(x) = \min\{L(x,y) \mid y \in X^{\bullet}\}, \text{ for any } x \in X^{\bullet}. \tag{3.4}$$

In other words, given a subset $X^\bullet$ of edges, the vertex-vertex distance map to $X^\bullet$ provides for each vertex x of $\mathbb{G}$ the minimal length of a path from x to a vertex of $X^\bullet$. An illustration of vertex-vertex distance map $D_{X^\bullet}^{(\bullet,\bullet)}$ is provided in Figure 3.8.

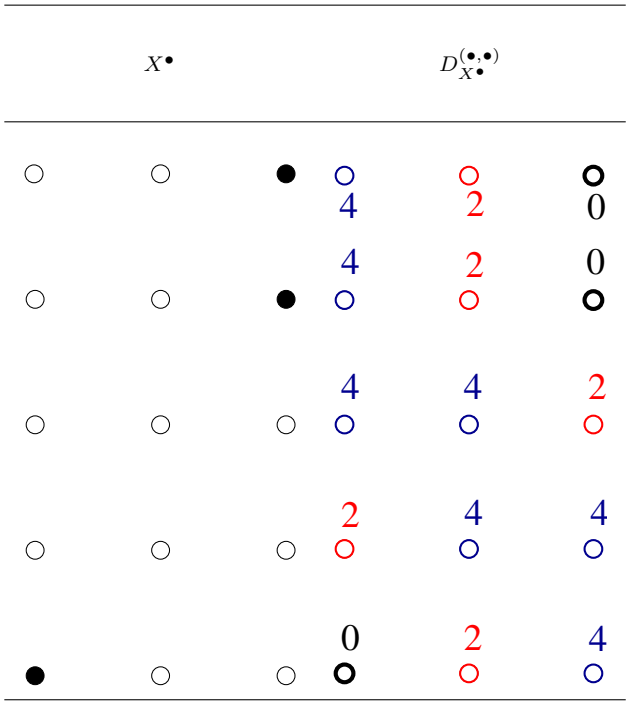


Figure 3.8: Illustration of Vertex-vertex distance map on graphs. The first subfigure shows the set of vertices $X^\bullet$ in the graph $\mathbb{G}$. The second subfigure shows the vertex-vertex distance maps to the set $X^\bullet$.

As we presented in chapter 2 a link is introduced by L.Vincen in [58] in order to compute the classical dilation and erosion using distance map on graph. In the following section, we will prove that all operators of erosions and dilations defined in Definition 13 can be characterized thanks to these proposed distance maps.

3.3 Iterated Morphological operators based on distance maps

In this section, we demonstrate using characterization theorems that the result of the operators defined in Definition 13 can be obtained from our proposed distance maps. More precisely, any dilation or erosion of a set, for any given size parameter λ, can be obtained by thresholding a distance map at value λ according to the relations established by the following theorems.

3.3.1 Edge-edge Dilation and Erosion

When the value of the parameter λ is even, the operators of dilation $\Delta_{\lambda/2}$ and of erosion $\varepsilon_{\lambda/2}$ map a set of edges to a set of edges in the graph $\mathbb{G}$. In order to characterize these operators, we establish in this part a link between the proposed *edge-edge* distance map $D_{X^\times}^{(\times,\times)}$ and these two operators.

The following Theorem 21 states that to obtain the edge-edge dilation and erosion, one needs to compute an *edge-edge* distance map and to deduce the result of these operators by considering thresholds of distance map at value λ.

Theorem 21. *Let λ be any even positive integer. The two following relations hold true.*

$$\Delta_{\lambda/2}(X^\times) = \{u \in \mathbb{G}^\times \mid D_{X^\times}^{(\times,\times)}(u) \leq \lambda\}, \text{ for any } X^\times \in \mathcal{G}^\times; \text{ and} \tag{3.5}$$

$$\varepsilon_{\lambda/2}(X^\times) = \{u \in \mathbb{G}^\times \mid D_{\overline{X^\times}}^{(\times,\times)}(u) > \lambda\}, \text{ for any } X^\times \in \mathcal{G}^\times. \tag{3.6}$$

where $\overline{X^\times}$ is the complement of $X^\times$ in $\mathbb{G}^\times$, that is $\overline{X^\times} = \mathbb{G}^\times \setminus X^\times$.

Proof. Let us now prove that the relation (3.5) holds true. Let λ be even and let $X^\times \subseteq \mathbb{G}^\times$. The following statements are equivalent.

$\Delta_{\lambda/2}(X^\times) = (\delta^\times \circ \delta^\bullet)^{\lambda/2}(X^\times)$, (by Equation 2.47)

$\Delta_{\lambda/2}(X^\times) = \delta^\times \circ (\delta^\bullet \circ \delta^\times)^{(\lambda-2)/2} \circ \delta^\bullet(X^\times)$

$\Delta_{\lambda/2}(X^\times) = \delta^\times \circ \delta_{(\lambda-2)/2} \circ \delta^\bullet(X^\times)$, (by Equation 2.45)

$\Delta_{\lambda/2}(X^\times) = \delta_{(\lambda-1)/2} \circ \delta^\bullet(X^\times)$, (by Equation 2.49)

$\Delta_{\lambda/2}(X^\times) = \{u \in \mathbb{G}^\times \mid D_{\delta^\bullet(X^\times)}^{(\bullet,\times)}(u) \leq \lambda - 1\}$, (by Equation 3.7)

$\Delta_{\lambda/2}(X^\times) = \{u \in \mathbb{G}^\times \mid \exists\, x \in \delta^\bullet(X^\times), \exists$ a path from x to u of length not greater than $\lambda - 1\}$, (by Equation 3.2)

$\Delta_{\lambda/2}(X^\times) = \{u \in \mathbb{G}^\times \mid \exists v \in X^\times, \exists$ a path from v to u of length not greater than $\lambda\}$, (by Equation 2.28)

$\Delta_{\lambda/2}(X^\times) = \{u \in \mathbb{G}^\times \mid D_{X^\times}^{(\times,\times)}(u) \leq \lambda\}$, (by Equation 3.1)

The relation (3.6) will be deduced by duality from the relation (3.5). Using the property of duality provided in [14], we have $\Delta_{\lambda/2}(X^\times) = \overline{\varepsilon_{\lambda/2}(\overline{X^\times})}$. Therefore, by relation (3.5), we have the equality $\Delta_{\lambda/2}(X^\times) = \overline{\{u \in \mathbb{G}^\times \mid D_{\overline{X^\times}}^{(\times,\times)}(u) \leq \lambda\}}$ which is equivalent to $\varepsilon_{\lambda/2}(X^\times) = \{u \in \mathbb{G}^\times \mid D_{\overline{X^\times}}^{(\times,\times)}(u) > \lambda\}$. □

In other words, Theorem 21 states that, when λ is even, the set $\Delta_{\lambda/2}(X^\times)$ contains any edge with a value not greater than λ for the distance map $D_{X^\times}^{(\times,\times)}$. In addition, it states that the set $\varepsilon_{\lambda/2}(X^\times))$ contains any edge with a value greater than λ for the distance map $D_{\overline{X^\times}}^{(\times,\times)}$.

For example, the dilations $\{\Delta_{\lambda/2}(X^\times)\}$ (resp. the erosions $\{\varepsilon_{\lambda/2}(X^\times)\}$) with even value of λ in $\{2, 4\}$, which are shown in Figure 3.9 (resp. Figure 3.10), can be obtained by thresholding the edge-edge distance map $D^{(\times,\times)}_{X^\times}$ (resp. $D^{(\times,\times)}_{\overline{X^\times}}$) presented in the second column Figure 3.9 (resp. Figure 3.10) at the values 2 and 4 (see the third and the fourth column of the Figure 3.9, respectively (resp. Figure 3.10)).

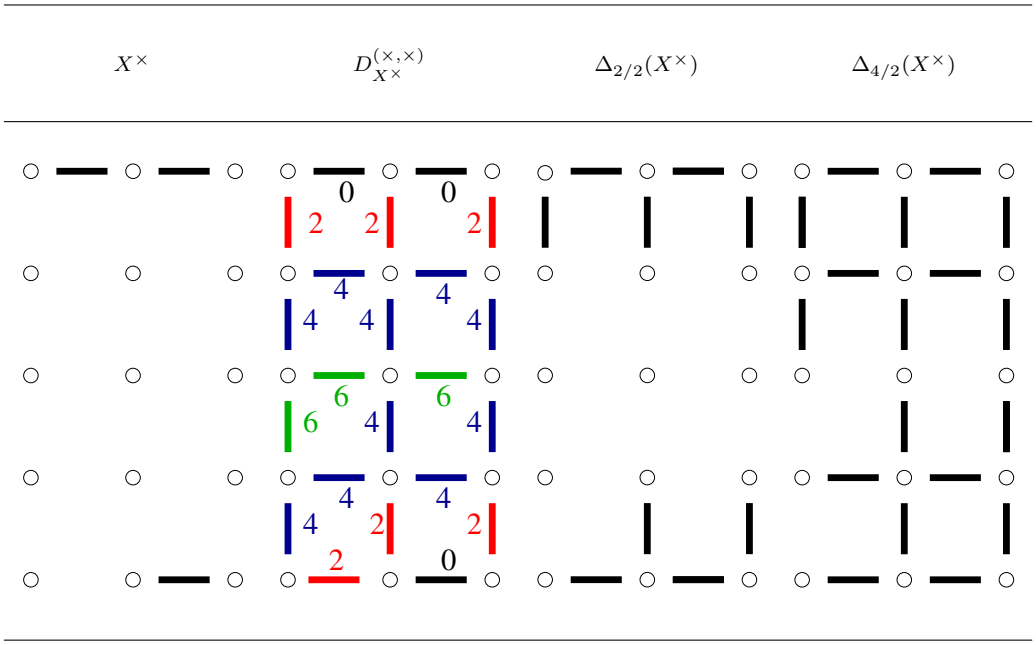


Figure 3.9: Illustration of the operator $\Delta_{\lambda/2}(X^\times)$ with $\lambda \in 2, 4$. The considered sets are depicted by black line segments.

3.3.2 Vertex-edge Dilation and erosion

When dealing with odd value of the parameter λ, the operators of dilation $\delta_{\lambda/2}$ and of erosion $\epsilon_{\lambda/2}$ map a set of vertices to a set of edges. In order to characterize these operators, relations are established between the proposed *vertex-edge* distance map $D^{(\bullet,\times)}_{X^\bullet}$ and these two operators. Indeed, the following theorem (Theorem 22) states that, one needs to compute an *vertex-edge* distance map and to deduce the result of these operators by considering thresholds of distance map at value λ.

Theorem 22. *Let λ be any positive odd integer. The two following relations hold true.*

$$\delta_{\lambda/2}(X^\bullet) = \{u \in \mathbb{G}^\times \mid D^{(\bullet,\times)}_{X^\bullet}(u) \leq \lambda\}, \text{ for any } X^\bullet \in \mathcal{G}^\bullet; \tag{3.7}$$

$$\epsilon_{\lambda/2}(X^\bullet) = \{u \in \mathbb{G}^\times \mid D^{(\bullet,\times)}_{\overline{X^\bullet}}(u) > \lambda\}, \text{ for any } X^\bullet \in \mathcal{G}^\bullet; \tag{3.8}$$

where $\overline{X^\times}$ is the complement of $X^\times$ in $\mathbb{G}^\times$, that is $\overline{X^\times} = \mathbb{G}^\times \backslash X^\times$ and $\overline{X^\bullet}$ is the complement of $X^\bullet$ in $\mathbb{G}^\bullet$, that is $\overline{X^\bullet} = \mathbb{G}^\bullet \setminus X^\bullet$.

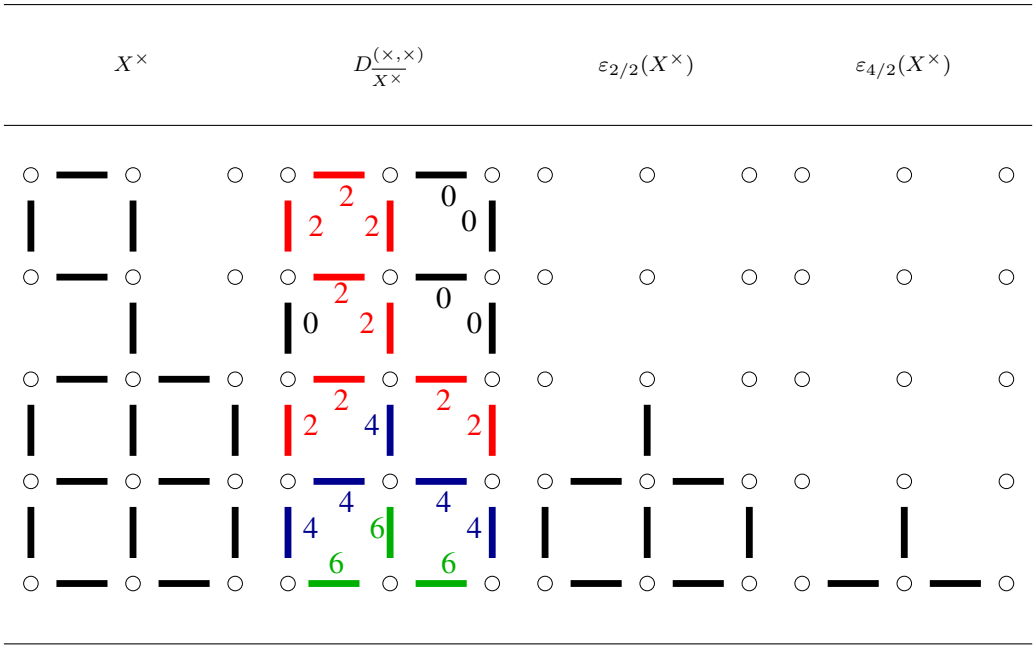


Figure 3.10: Illustration of the operator $\varepsilon_{\lambda/2}(X^\times)$ with $\lambda \in 2, 4$. The considered sets are depicted by black line segments.

Proof. We move now to prove that the relation (3.7) holds true. Let λ be odd and let $X^\bullet \subseteq \mathbb{G}^\bullet$. The following statements are equivalent.

$\delta_{\lambda/2}(X^\bullet) = \delta^\times \circ \delta_{(\lambda-1)/2}(X^\bullet)$, (by Equation 2.49)

$\delta_{\lambda/2}(X^\bullet) = \delta^\times(\{x \in \mathbb{G}^\bullet \mid D^{(\bullet,\bullet)}_{X^\bullet}(x) \leq \lambda - 1\})$, (by Equation 3.11, since $\lambda - 1$ is even)

$\delta_{\lambda/2}(X^\bullet) = \delta^\times(\{x \in \mathbb{G}^\bullet \mid \exists y \in X^\bullet, \exists$ a path from y to x of length not greater than $\lambda - 1\})$, (by Definition 20)

$\delta_{\lambda/2}(X^\bullet) = \{\{x, z\} \in \mathbb{G}^\times \mid \exists y \in X^\bullet, \exists$ a path from y to x of length not greater than $\lambda - 1\})$, (by Equation 2.31)

$\delta_{\lambda/2}(X^\bullet) = \{u \in \mathbb{G}^\times \mid \exists y \in X^\bullet, \exists$ a path from y to u of length not greater than $\lambda\})$, (by definition of a path)

$\delta_{\lambda/2}(X^\bullet) = \{u \in \mathbb{G}^\times \mid D^{(\bullet,\times)}_{X^\bullet}(u) \leq \lambda\}$, (by Equation 3.2)

The relation (3.8) will be deduced by duality from the relation (3.7). Using the property of duality provided in [14], we have $\delta_{\lambda/2}(X^\bullet) = \overline{\epsilon_{\lambda/2}(\overline{X^\bullet})}$. Therefore, by relation (3.7), we have the equality $\delta_{\lambda/2}(X^\bullet) = \overline{\{u \in \mathbb{G}^\times \mid D^{(\bullet,\times)}_{\overline{X^\bullet}}(u) \leq \lambda\}}$ which is equivalent to $\epsilon_{\lambda/2}(X^\bullet) = \{u \in \mathbb{G}^\times \mid D^{(\bullet,\times)}_{\overline{X^\bullet}}(u) > \lambda\}$. □

In other words, Theorem 22 states that, when λ is odd, the set $\delta_{\lambda/2}(X^\bullet)$ contains any edge of value not greater than λ (resp. greater than λ for the distance map $D^{(\bullet,\times)}_{X^\bullet}$. Also, the theorem states that the set $\epsilon_{\lambda/2}(X^\bullet)$ contains any edge of value greater than λ for the

distance map $D^{(\bullet,\times)}_{\overline{X^\bullet}}$.

For instance, the dilations $\{\delta_{\lambda/2}(X^\bullet)\}$ (resp. the erosions $\{\epsilon_{\lambda/2}(X^\bullet)\}$) for odd values of λ in $\{1, 3\}$, which are shown in Figure 3.11 (resp. Figure 3.12), can be obtained and characterized by thresholding the vertex-edge distance map $D^{(\bullet,\times)}_{X^\bullet}$ (resp. $D^{(\bullet,\times)}_{\overline{X^\bullet}}$) presented in the second column of Figure 3.11 (resp. Figure 3.12) at the values 1 and 3 (see the third and the fourth column of the Figure 3.11, respectively (resp. Figure 3.12)).

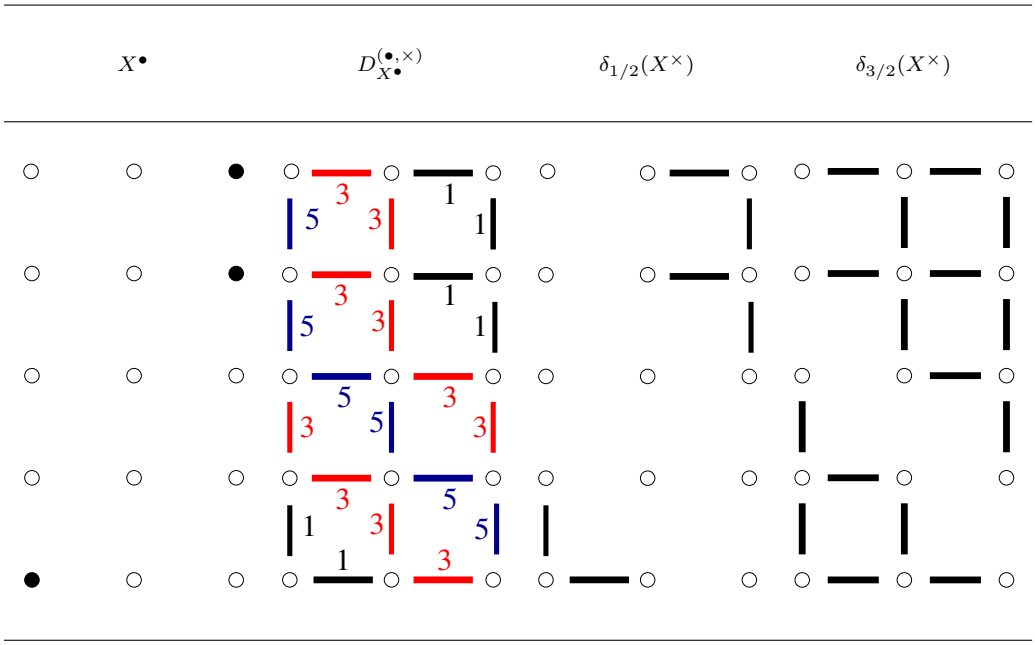


Figure 3.11: Illustration of the operator $\delta_{\lambda/2}(X^\times)$ with $\lambda \in 1, 3$. The considered sets are depicted by black dots segments.

3.3.3 Edge-vertex Dilation and erosion

When the parameter λ is odd value, the operators of dilation $\Delta_{\lambda/2}$ and of erosion $\varepsilon_{\lambda/2}$ map a set of edges to a set of vertices. In order to obtain these operators, relations are established between these operators and the proposed *edge-vertex* distance map $D^{(\times,\bullet)}_{X^\times}$. Indeed, as stated by the following theorem (Theorem 23), one needs to compute an *edge-vertex* distance map and to deduce the result of these operators of dilation and erosion by considering thresholds of distance map at value λ.

Theorem 23. *Let λ be any odd positive integer. The four following relations hold true.*

$$\Delta_{\lambda/2}(X^\times) = \{x \in \mathbb{G}^\bullet \mid D^{(\times,\bullet)}_{X^\times}(x) \leq \lambda\}, \text{ for any } X^\times \in \mathcal{G}^\times; \text{ and} \tag{3.9}$$

$$\varepsilon_{\lambda/2}(X^\times) = \{x \in \mathbb{G}^\bullet \mid D^{(\times,\bullet)}_{\overline{X^\times}}(x) > \lambda\}, \text{ for any } X^\times \in \mathcal{G}^\times, \tag{3.10}$$

where $\overline{X^\times}$ is the complement of $X^\times$ in $\mathbb{G}^\times$, that is $\overline{X^\times} = \mathbb{G}^\times \setminus X^\times$ and $\overline{X^\bullet}$ is the complement of $X^\bullet$ in $\mathbb{G}^\bullet$, that is $\overline{X^\bullet} = \mathbb{G}^\bullet \setminus X^\bullet$.

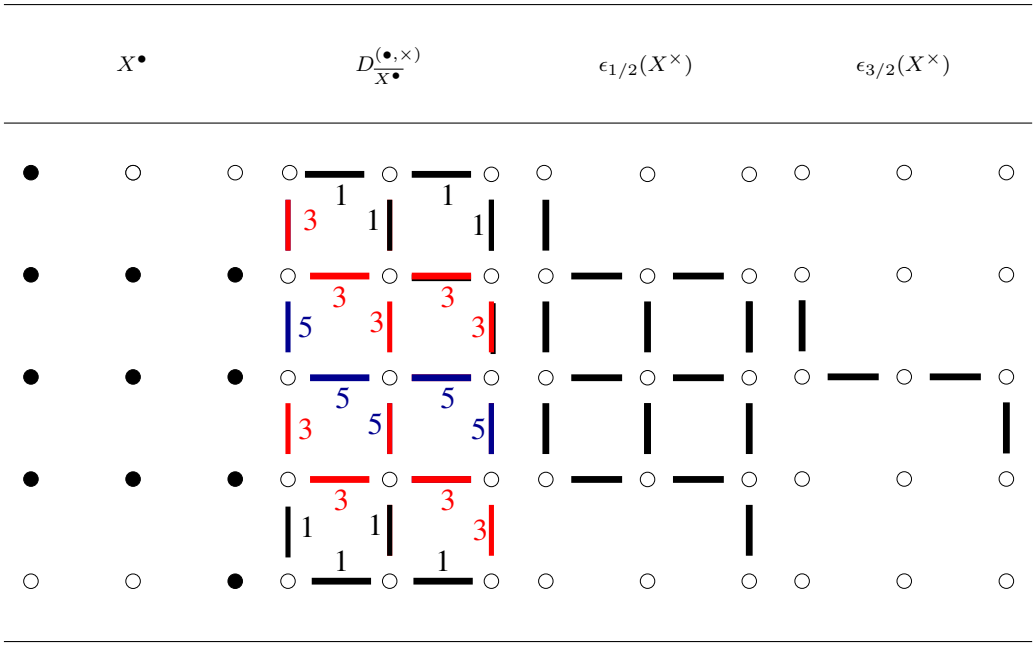


Figure 3.12: Illustration of the operator $\epsilon_{\lambda/2}(X^\times)$ with $\lambda \in 1, 3$. The considered sets are depicted by black dots segments.

Proof. We are now going to prove that relation (3.9) holds true. Let λ be odd and let $X^\times \subseteq \mathbb{G}^\bullet$. The following statements are equivalent.

$$\Delta_{\lambda/2}(X^\times) = \delta^\bullet \circ \Delta_{(\lambda-1)/2}(X^\times), \quad \text{(by Equation 2.51)}$$

$$\Delta_{\lambda/2}(X^\times) = \delta^\bullet \circ (\delta^\times \circ \delta^\bullet)^{(\lambda-1)/2}(X^\times), \quad \text{(by Equation 2.47)}$$

$$\Delta_{\lambda/2}(X^\times) = (\delta^\bullet \circ \delta^\times)^{(\lambda-1)/2} \circ \delta^\bullet(X^\times)$$

$$\Delta_{\lambda/2}(X^\times) = \delta_{(\lambda-1)/2} \circ \delta^\bullet(X^\times), \quad \text{(byEquation 2.45)}$$

$$\Delta_{\lambda/2}(X^\times) = \{x \in \mathbb{G}^\bullet \mid D^{(\bullet,\bullet)}_{\delta^\bullet(X^\times)}(x) \leq \lambda - 1\}, \quad \text{(by Equation 3.11)}$$

$$\Delta_{\lambda/2}(X^\times) = \{x \in \mathbb{G}^\bullet \mid \exists y \in \delta^\bullet(X^\times), \exists \text{ a path from } y \text{ to } x \text{ of length not greater than } \lambda - 1\},$$
(by Definition 20)

$$\Delta_{\lambda/2}(X^\times) = \{x \in \mathbb{G}^\bullet \mid \exists u \in X^\times, \exists \text{ a path from } u \text{ to } x \text{ of length not greater than } \lambda\}$$
(by Equation 2.28)

$$\Delta_{\lambda/2}(X^\times) = \{x \in \mathbb{G}^\bullet \mid D^{(\times,\bullet)}_{X^\times}(x) \leq \lambda\} \quad \text{(by Equation 3.3)}$$

The relation (3.10) will be deduced by duality from the relation (3.9). Using the property of duality provided in [14], we have $\Delta_{\lambda/2}(X^\times) = \overline{\varepsilon_{\lambda/2}(\overline{X^\times})}$. Therefore, by relation (3.9), we have the equality $\Delta_{\lambda/2}(X^\times) = \overline{\{u \in \mathbb{G}^\times \mid D^{(\times,\bullet)}_{\overline{X^\times}}(u) \leq \lambda\}}$ which is equivalent to $\varepsilon_{\lambda/2}(X^\times) = \{u \in \mathbb{G}^\times \mid D^{(\times,\bullet)}_{\overline{X^\times}}(u) > \lambda\}$. □

In other words, Theorem 23 states that, when λ is odd, the set $\Delta_{\lambda/2}(X^\times)$ contains any vertex of value not greater than λ for the distance map $D^{(\times,\bullet)}_{X^\times}$. And, the set $\varepsilon_{\lambda/2}(X^\times)$

contains any vertex of value greater than λ for the distance map $D^{(\times,\bullet)}_{\overline{X^\times}}$.

For example, the dilations $\{\Delta_{\lambda/2}(X^\times)\}$ (resp. the erosions $\{\varepsilon_{\lambda/2}(X^\times)\}$) with odd value of λ in $\{1,3\}$, which are shown in Figure 3.13 (resp. Figure 3.14), can be obtained by thresholding the edge-vertex distance map $D^{(\times,\bullet)}_{X^\times}$ (resp. $D^{(\times,\bullet)}_{\overline{X^\times}}$) presented in Figure 3.13 (resp. Figure 3.14) presented in the second column Figure 3.13 (resp. Figure 3.14)at the values 1 and 3 (see the third and the fourth column of the Figure 3.13 respectively (resp. Figure 3.14)).

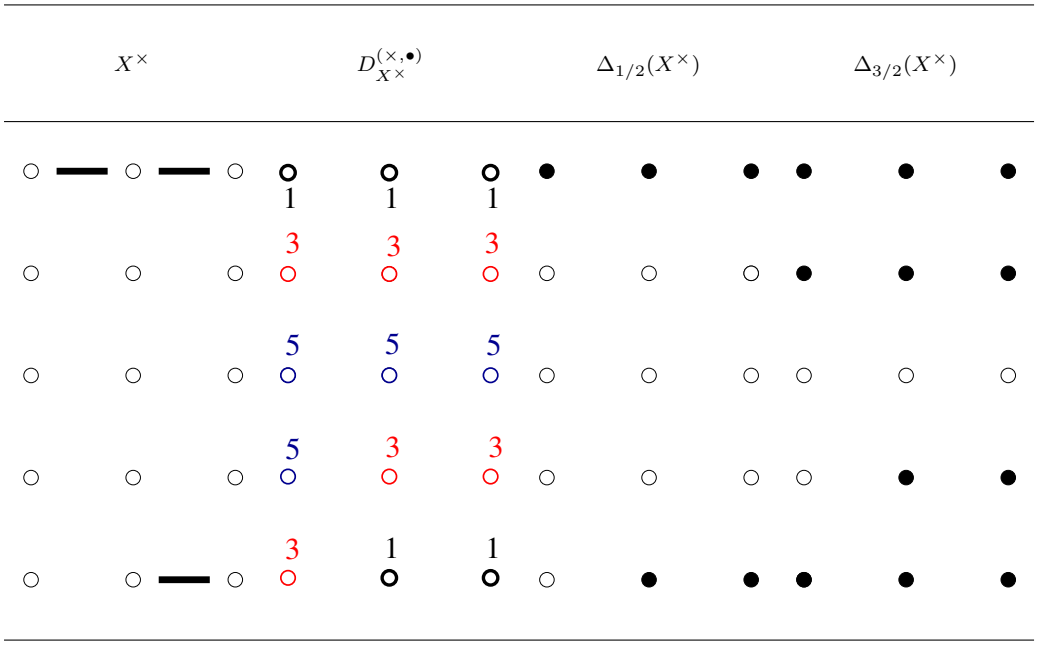


Figure 3.13: Illustration of the operator $\Delta_{\lambda/2}(X^\times)$ with $\lambda \in 1, 3$. The considered sets are depicted by black dots.

3.3.4 Vertex-vertex Dilation and erosion

When the parameter λ is even value, the operators of dilation $\delta_{\lambda/2}$ and of erosion $\epsilon_{\lambda/2}$ map a set of vertices to a set of vertices.

Based on the link proposed in [58], the following property states that one can compute a the *vertex-vertex* distance map $D^{(\bullet,\bullet)}_{X^\times}$ and deduce the result of the dilation or erosion by considering thresholds of that distance map at value λ.

Property 24 (from [58]). *Let $X^\bullet$ be a subset of $\mathbb{G}$. Then, the following relations hold true:*

$$\delta_{\lambda/2}(X^\bullet) = \{x \in \mathbb{G}^\bullet \mid D^{(\bullet,\bullet)}_{X^\bullet}(x) \leq \lambda\}, \text{ for any } X^\bullet \in \mathcal{G}^\bullet; \tag{3.11}$$

$$\epsilon_{\lambda/2}(X^\bullet) = \{x \in \mathbb{G}^\bullet \mid D^{(\bullet,\bullet)}_{\overline{X^\bullet}}(x) > \lambda\}, \text{ for any } X^\bullet \in \mathcal{G}^\bullet, \tag{3.12}$$

where $\overline{X^\bullet}$ is the complement of $X^\bullet$ in $\mathbb{G}^\bullet$, that is $\overline{X^\bullet} = \mathbb{G}^\bullet \setminus X^\bullet$.

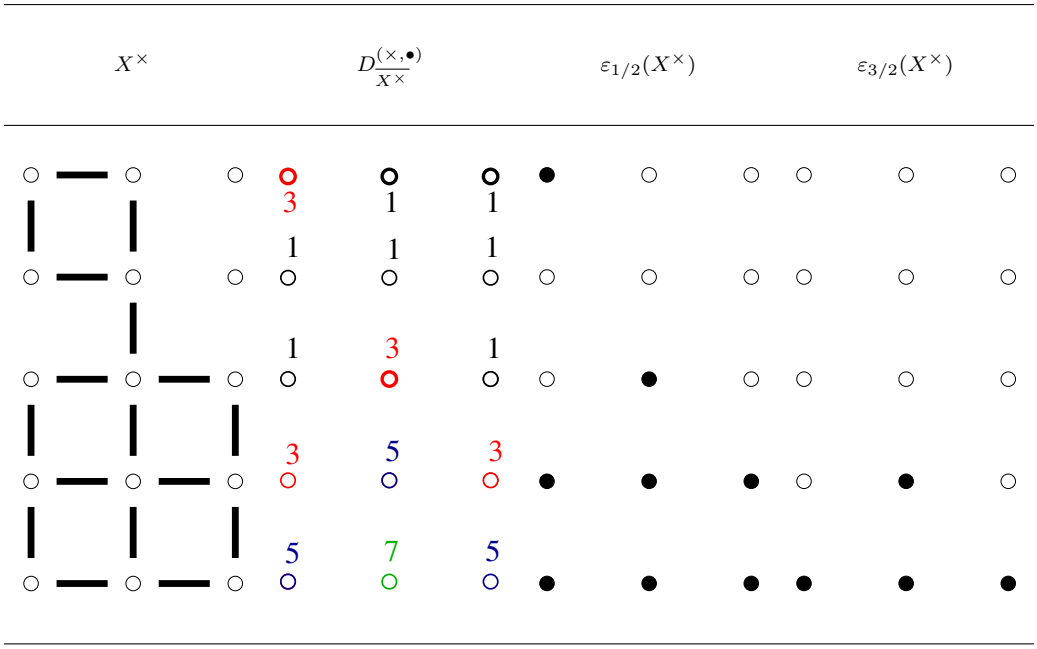


Figure 3.14: Illustration of the operator $\varepsilon_{\lambda/2}(X^\times)$ with $\lambda \in 1, 3$. The considered sets are depicted by black dots.

In other words, the set $\delta_{\lambda/2}(X^\bullet)$ contains any vertex with a value not greater than λ for the distance map $D^{(\bullet,\bullet)}_{X^\bullet}$. The set $\epsilon_{\lambda/2}(X^\bullet)$ contains any vertex with a value greater than λ for the distance map $D^{(\bullet,\bullet)}_{\overline{X^\bullet}}$ to the complementary set $\overline{X^\bullet}$ of $X^\bullet$.

For example, the dilations $\{\delta_{\lambda/2}(X^\bullet)\}$ (resp. the erosions $\{\epsilon_{\lambda/2}(X^\bullet)\}$) with odd value of λ in $\{2, 4\}$, which are shown in Figure 3.15 (resp. Figure 3.16), can be obtained by thresholding the vertex-vertex distance map $D^{(\bullet,\bullet)}_{X^\times}$ (resp. $D^{(\bullet,\bullet)}_{\overline{X^\bullet}}$) presented in Figure 3.15 (resp. Figure 3.16) presented in the second column Figure 3.15 (resp. Figure 3.16) at the values 2 and 4 (see the third and the fourth column of the Figure 3.15 respectively (resp. Figure 3.16)).

As a result, based on Theorems 21, 22, 23 and Property 24 which are provided in this section, the result of any operator of Definition 13 can be characterized by thresholding a distance map at value λ, leading to efficient algorithms to compute the results of these operators. Note that, thresholding distance maps that weights the vertices $X^\bullet$ (resp. edges $X^\times$) of a graph $\mathbb{G}$ can be done in linear-time with respect to the number of vertices $|X^\bullet|$ (resp. edges $|X^\times|$) of the graph $\mathbb{G}$ with a sequential algorithm. Therefore, in the following section linear-time sequential algorithms are devoted for computing distance maps on graphs, hence all dilations and erosions.

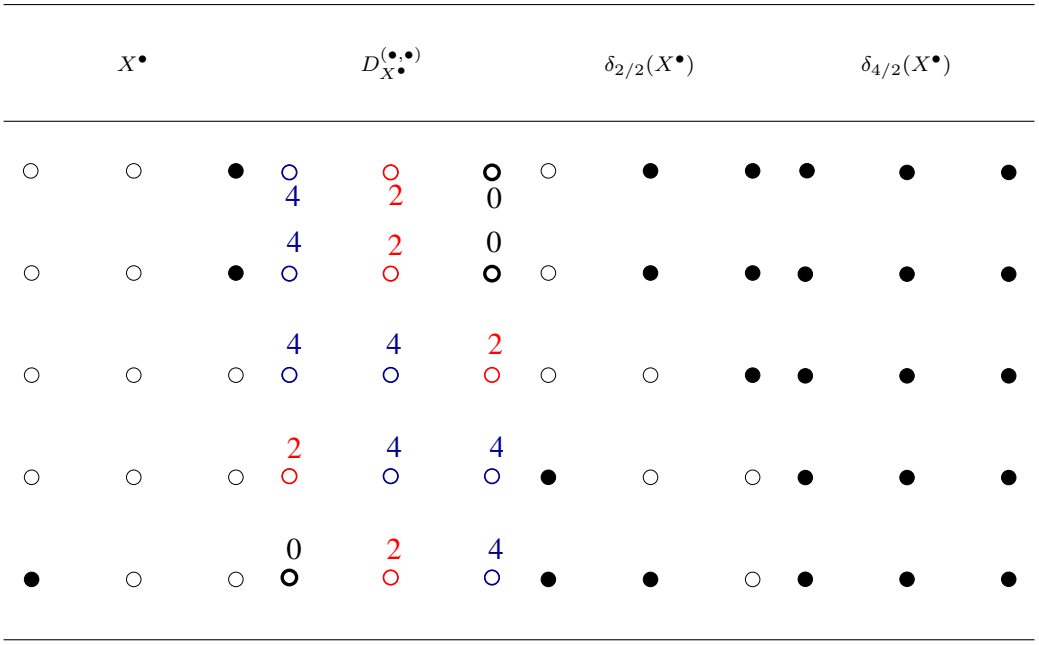


Figure 3.15: Illustration of the operator $\delta_{\lambda/2}(X^\bullet)$ with $\lambda \in 2, 4$. The considered sets are depicted by black dots.

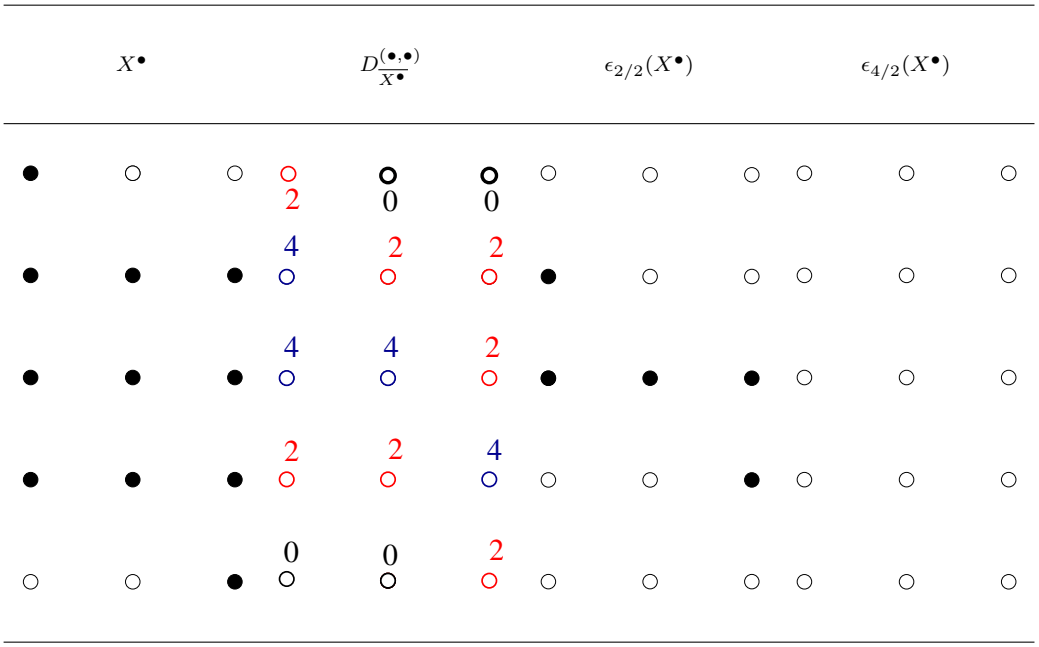


Figure 3.16: Illustration of the operator $\epsilon_{\lambda/2}(X^\bullet)$ with $\lambda \in 2, 4$. The considered sets are depicted by black dots.

3.4 Proposed sequential algorithms for morphological operators on graphs

After establishing links between our proposed distance maps and the iterated morphological operators on graphs (see Definition 13), we proposed efficient algorithms to compute

vertex-edge, vertex-vertex, edge-edge, and edge-vertex distance maps in linear time with respect to the size of the graph $\mathbb{G}$, hence all morphological operators on graphs. In particular, when the considered graph is unweighted, the lengths of shortest paths to a given vertex can be obtained with a breadth-first search algorithm in linear $O(|\mathbb{G}^\bullet| + |\mathbb{G}^\times|)$ time complexity (see Chapter 2). Therefore, our sequential algorithms proposed in this section are variations on this breadth-first search algorithm.

3.4.1 Vertex-edge and Vertex-vertex distance map algorithm

We present in this part Algorithm 1 which allows us to compute both the vertex-edge distance map $D_{X^\bullet}^{(\times;\bullet)}$ and the vertex-vertex distance map $D_{X^\bullet}^{(\bullet;\bullet)}$ to a given subset $X^\bullet$ of vertices. The basic idea of this algorithm is to perform a breadth-first exploration of the edges and vertices of $\mathbb{G}$ from the elements of $X^\bullet$. The distance map value of each vertex (resp. edge) is computed when the vertex is first encountered during exploration. The breadth-first exploration, which ensures correct distance values, is made possible thanks to the auxiliary queue $\mathcal{Q}$ that is managed with a FIFO (First-In-First-Out) property.

Algorithm 1: Vertex-edge and vertex-vertex distance maps.

Data: a connected graph $\mathbb{G} = (\mathbb{G}^\bullet, \mathbb{G}^\times)$, and subset $X^\bullet$ of $\mathbb{G}^\bullet$.
Result: the distance maps $D_{X^\bullet}^{(\bullet;\bullet)}$ and $D_{X^\bullet}^{(\bullet;\times)}$ to the set $X^\bullet$.

1 $\mathcal{Q}$:= an empty queue with FIFO property;
2 **foreach** *vertex* x *in* $\mathbb{G}^\bullet$ **do**
3 **if** $x \in X^\bullet$ **then** $\mathcal{Q}$.push(x); $D_{X^\bullet}^{(\bullet;\bullet)}(x) := 0$;
4 **else** $D_{X^\bullet}^{(\bullet;\bullet)}(x) := \infty$;
5 **foreach** *edge* $\{x, y\}$ *in* $\mathbb{G}^\times$ **do** $D_{X^\bullet}^{(\bullet;\times)}(\{x, y\}) := \infty$;
6 **while** $\mathcal{Q}$*.isNotEmpty()* **do**
7 $x := \mathcal{Q}$.pop();
8 **foreach** *vertex* y *adjacent to* x *in* $\mathbb{G}$ **do** // *i.e.*, when $\{x, y\} \in \mathbb{G}^\times$
9 **if** $D_{X^\bullet}^{(\bullet;\times)}(\{x, y\}) = \infty$ **then** $D_{X^\bullet}^{(\bullet;\times)}(\{x, y\}) := D_{X^\bullet}^{(\bullet;\bullet)}(x) + 1$;
10 **if** $D_{X^\bullet}^{(\bullet;\bullet)}(y) = \infty$ **then** $\mathcal{Q}$.push(y); $D_{X^\bullet}^{(\bullet;\bullet)}(y) := D_{X^\bullet}^{(\bullet;\bullet)}(x) + 2$;

We provide now an overview of Algorithm 1. Given a subset $X^\bullet$ of $\mathbb{G}^\bullet$, this Algorithm 1 can be sketched as follows:

1. An initialization step consists of inserting the vertices that belong to $X^\bullet$ into the queue $\mathcal{Q}$, and set their distance value to 0 (see lines 2 to 4) whereas set the distance value of the remaining vertices in $\mathbb{G}^\bullet$ to infinity. It consists also to set the distance value of edges in $\mathbb{G}^\times$ to infinity.
2. Browse the neighbors (*i.e.* the adjacent vertices and adjacent edges) of each vertex popped from $\mathcal{Q}$, and set the distance value of the adjacent vertices (resp. adjacent edges) not yet traversed to the distance value of the popped vertex +1 (resp. +2).

3. Repeat the step 2 until $\mathcal{Q}$ is empty.

Observe that in Algorithm 1, the distance value between two neighbors is equal to two. This is indeed correct with respect to the above definition of length of a path.

3.4.1.1 The correctness of Vertex-edge and Vertex-vertex distance map algorithm

In order to establish the correctness of Algorithm 1, let us analyze four invariants of Algorithm 1, *i.e.* four properties that hold true at every iterations of the main loop (line 6): i) every finite value associated to a point or an edge is the correct distance map value to the input set $X^\bullet$; ii) every vertex of $\mathbb{G}$ with a finite value that has an adjacent edge or vertex with an infinite value belongs to $\mathcal{Q}$; iii) the vertices of $\mathcal{Q}$ are mapped to finite values and they are stored in increasing order, the value of the first element of $\mathcal{Q}$ being $\mathcal{Q}_{\min}$ and the one of the last element being not greater than $\mathcal{Q}_{\min}+2$; and iv) any element e of $\mathbb{G}^\bullet$ or of $\mathbb{G}^\times$ such that there is a vertex x in $X^\bullet$ with $L(x,e) < \mathcal{Q}_{\min}$ is mapped to a finite value.

These invariants trivially hold true after the initialization step (lines 1-5). At every iteration of the main loop, a vertex x is popped from $\mathcal{Q}$ (line 6). Let λ be the distance map value of this vertex. In order to preserve properties i)-iv) above, all vertices and edges adjacent to x are analyzed thanks to the foreach loop at line 8. If an adjacent edge (resp. vertex) with an infinite value is found, then we can easily deduced that this element can be reach from $X^\bullet$ (through x) with a path of length $\lambda+1$ (resp. $\lambda+2$). Furthermore, such path is a shortest one (otherwise, due to invariant ii) and iv)), there would be an adjacent vertex in $\mathcal{Q}$ with a value smaller than λ, a contradiction with invariant iii)). Thus, the new value is correct and invariant i) is preserved. The newly discovered vertex is pushed in $\mathcal{Q}$ at line 10 in order to preserve invariant ii). Note that invariant iii) is trivially preserved since the newly inserted vertex has a value of $\lambda+2$. Let us finally establish that invariant iv) holds true at the end of the while loop iteration. It can be seen that any element e in $\mathbb{G}^\bullet \cup \mathbb{G}^\times$ such that there is a vertex y in $X^\bullet$ with $L(y,e) < \mathcal{Q}_{\min}$ is adjacent to a vertex z such that $L(y,z) < Q_{\min} - 2$. But it can be observed that we have $\mathcal{Q}_{\min} \in \{\lambda, \lambda+2\}$. Thus, we deduce that $L(y,z) < \lambda$. Hence, since invariant iv) holds true at the beginning of the while loop iteration, we deduce that z is mapped to a finite value. Furthermore, since $L(y,z) < \lambda$, we deduce that z is not in $\mathcal{Q}$. Therefore, by (the contraposition of) ii), we may affirm that e is mapped to a finite value, which proves that invariant iv) holds true at the end of the iteration of the while loop. Thus, invariants i)-iv) hold true when the foreach loop at line 8 terminates. Hence, we deduce that when the algorithm halts the produced maps $D^{(\bullet,\bullet)}_{X^\bullet}$ and $D^{(\bullet,\times)}_{X^\bullet}$ are indeed the vertex-vertex distance and the the vertex-edge distance map to $X^\bullet$.

3.4.1.2 Complexity analysis of Vertex-edge and Vertex-vertex distance map algorithm

Let us now analyze the time complexity of Algorithm 1. First, the initialization of the queue $\mathcal{Q}$ (line 1) is done in constant time. Since a push operation on a queue can be done in constant time, the overall complexity of the initialization loop at lines 1-4 is done in $O(|\mathbb{G}^\bullet|)$ and the edge distance map initialization is done in $O(|\mathbb{G}^\times|)$ by the loop at line 5. Then, in order to analyze the complexity of the main loop of Algorithm 1, it is important to note that every vertex of the graph is pushed at most once in $\mathcal{Q}$. This is ensured by immediately setting to a finite value a point which is inserted in $\mathcal{Q}$ and by only inserting in $\mathcal{Q}$ points with an infinite value (see lines 3 and 10). Thus, since a vertex is popped from $\mathcal{Q}$ at every iteration of the while loop, we deduce that there are at most $|V|$ iterations of this loop. At each iteration of the while loop the vertices and edges adjacent to the vertex which is popped are explored (foreach loop at line 8). Thus, if the graph $\mathbb{G}$ is represented as an array of lists associating to each vertex the list of its adjacent edges, the complexity of line 8 is $O(|V| + |E|)$. The instructions, inside this foreach loop are executed at most $2 \times |E|$ times, each of them being individually performed in constant time. Therefore, the overall complexity of these instructions is $O(|E|)$. Thus, we deduce that the overall complexity of the algorithm is $O(|V| + |E|)$.

3.4.2 Edge-edge and Edge-vertex distance map algorithm

In this part, we provide Algorithm 2 that allows us to compute both the edge-edge distance map $D_{X^\times}^{(\times,\times)}$ and edge-vertex distance maps $D_{X^\times}^{(\times,\bullet)}$ to a given subset $X^\times$ of edges instead of the vertex-vertex and vertex-edge distance maps to a subset of vertices. The initialization steps of algorithm presented hereafter (lines 2 to 8), which has to be made from a set of edges rather than from a set of vertices, is the main difference with Algorithm 1. However, the same arguments can be used to establish the correctness and the complexity of Algorithm 2.

As a conclusion, the following property resumes the main ideas of last sections.

Property 25. *Algorithm 1 outputs two maps $D_{X^\bullet}^{(\bullet,\bullet)}$ and $D_{X^\bullet}^{(\bullet,\times)}$ that are the vertex-vertex and vertex-edge distance maps, respectively, to the input subset $X^\bullet$ of $\mathbb{G}^\bullet$. Algorithm 2 outputs two maps $D_{X^\times}^{(\times,\times)}$ and $D_{X^\times}^{(\times,\bullet)}$ that are the edge-edge and edge-vertex distance maps, respectively, to the input subset $X^\times$ of $\mathbb{G}^\times$. Furthermore Algorithms 1 and 2 both run in linear time with respect to $|V| + |E|$.*

3.5 Conclusion

In this chapter, we show that all operators of erosions and dilations defined in [14] (see Definition 13) can be characterized thanks to our proposed distance maps, leading to

Algorithm 2: Edge-edge and edge-vertex distance maps.

Data: a connected graph $\mathbb{G} = (\mathbb{G}^\bullet, \mathbb{G}^\times)$ and a subset $X^\times$ of $\mathbb{G}^\times$.
Result: the distance maps $D_{X^\times}^{(\times,\times)}$ and $D_{X^\times}^{(\times,\bullet)}$ to the set $X^\times$.

$\mathcal{Q}$:= an empty queue with FIFO property;
foreach *vertex* x *in* $\mathbb{G}^\bullet$ **do** $D_{X^\times}^{(\times,\bullet)}(x) := \infty$;
foreach *edge* $\{x, y\}$ *in* $\mathbb{G}^\times$ **do**
    if $\{x, y\} \in X^\times$ **then**
        if $D_{X^\times}^{(\times,\bullet)}(x) == \infty$ **then** $\mathcal{Q}$.push(x); $D_{X^\times}^{(\times,\bullet)}(x) := 1$;
        if $D_{X^\times}^{(\times,\bullet)}(y) == \infty$ **then** $\mathcal{Q}$.push(y); $D_{X^\times}^{(\times,\bullet)}(y) := 1$;
        $D_{X^\times}^{(\times,\times)}(\{x, y\}) := 0$;
    else $D_{X^\times}^{(\times,\times)}(\{x, y\}) := \infty$;
while $\mathcal{Q}$.*isNotEmpty()* **do**
    $x := \mathcal{Q}$.pop();
    foreach *vertex* y *adjacent to* x *in* $\mathbb{G}$ **do** // *i.e.*, when $\{x, y\} \in \mathbb{G}^\times$
        if $D_{X^\times}^{(\times,\times)}(\{x, y\}) == \infty$ **then** $D_{X^\times}^{(\times,\times)}(\{x, y\}) = D_{X^\times}^{(\times,\bullet)}(x)+1$;
        if $D_{X^\times}^{(\times,\bullet)}(y) == \infty$ **then** $\mathcal{Q}$.push(y); $D_{X^\times}^{(\times,\bullet)}(y) := D_{X^\times}^{(\times,\bullet)} + 2$;

efficient computation of these operators. First, theorems are provided to demonstrate that the result of any operator of Definition 13 can be obtained by thresholding a distance map at value λ. These characterizations lead to obtain all operators of dilations and erosions on graphs with a single iteration and without any dependence with the parameter λ. Then, linear-time sequential algorithms are proposed to compute the proposed distance maps, hence the morphological operators on graphs.

The next chapter is devoted to propose a parallelization strategy leading to a fast computation of the proposed distance maps, hence the morphological operators of [14].

.

Chapter 4

Parallel strategy for distance maps on graphs

Contents

4.1 Introduction

In order to exploit the parallel machines and to benefit from the multi-core processors architecture, we propose a parallel strategy that leads to a fast computation of distance maps introduced in previous chapter, hence the mathematical morphology operators on graphs of [14]. To this end, a description of the proposed parallel strategy is provided in Section 4.2 and necessary auxiliary functions for this strategy are described in Section 4.2.2. Then, we state the complexity of our parallel algorithms under some proved assumptions about the graph and set under consideration in Section 4.3.

4.2 Proposed parallel strategy

The parallel strategy proposed in our work is based on dynamic partitioning. The partition method depends on the input set (the set of vertices or set of edges) and is iteratively computed during the execution. More precisely, our strategy iteratively considers

the successive level-sets of the distance maps where each level set being partitioned and then traversed in parallel.

In this section, we provide a precise description of our parallel strategy for vertex-vertex and vertex-edge distance maps (i.e. a parallelization of Algorithm 1) which can also be adapted to edge-edge and and edge-vertex distance maps computations to obtain a parallelization of Algorithm 2.

4.2.1 Description of parallel algorithms

Let us first present our strategy from a high level point of view. To this end, we recall the notion of a level set: Given an integer λ and a (distance) map D from $\mathbb{G}^\bullet$ in the set of integers, the λ-*level set* of D is the set of all elements of value λ for D (*i.e.*, the set $\{x \in \mathbb{G}^\bullet \mid D(x) = \lambda\}$).

We are now ready for providing the overview of our parallel algorithm for vertex-vertex and vertex-edge distance maps (see Algorithm 3). This description is similar for the parallel algorithm which compute edge-vertex and edge-edge distance maps to a given subset of edges (see Algorithm 4). The main difference with Algorithm 3 is in the initialization steps of Algorithm 4 (lines 7 to 12), which has to be made from a set of edges rather than from a set of vertices. So, for the sake of simplicity, hereafter we only focus on the Algorithm 3.

Given a subset $X^\bullet$ of $\mathbb{G}^\bullet$, after an initialization step where an integer variable λ is set to 0 and where the elements of $X^\bullet$ are inserted in a variable set E (hence E is the $(\lambda = 0)$-level-set of $D_{X^\bullet}^{(\bullet;\bullet)}$), our algorithm can be sketched as follows:

1. Partition E (*i.e.*, the λ-level set of $D_{X^\bullet}^{(\bullet;\bullet)}$) into p balanced subsets $E_1, \ldots, E_p$ (line 10 of Algorithm 3).
2. Assign each of the p subsets $E_1, \ldots, E_p$ to one of the p processors (line 11 of Algorithm 3).
3. Let, in parallel, each processor browse on the neighbors of the elements in its assigned subset E_i, insert those which are not yet traversed into a private variable set S_i, and set their distance map values to $\lambda + 2$, also set the distance map values of the edges linked the elements in E_i with their neighbors not yet traversed to $\lambda + 1$ (lines 13 to 18 of Algorithm 3).
4. Merge the private sets $\{S_i \mid i \in \{1, \ldots, p\}\}$ and store the result in E so that E becomes the $(\lambda + 2)$-level set of $D_{X^\bullet}^{(\bullet;\bullet)}$ or the $(\lambda + 1)$-level set of $D_{X^\bullet}^{(\bullet;\times)}$ (line 19 of Algorithm 3).
5. Increment λ and repeat steps 1-4 until E becomes empty (line 20).

In Step 3, in order to concurrently check if a vertex has been already traversed, we need to equip each vertex with a synchronization Boolean variable that is handled with an atomic *test-and-set* instruction. The test-and-set instruction sets a given variable to true and returns its old value as a single atomic (*i.e.*, non-interruptible) instruction.

Algorithm 3 provides the precise description of our parallel strategy. It uses two auxiliary functions called *Partition* and *Union* (see Section 4.2.2 for an extensive study) . The function *Partition* takes two arguments. The first one is the set E to be partitioned and the second one is an integer p corresponding to the number of classes of the returned partition. The function *Partition* returns a balanced partition of E into p classes, the partition being balanced in the sense that any of its classes contains either $|E|/p$ or $|E|/p+1$ elements. The function *Union* is simply computing the union of its arguments.

Algorithm 3: Parallel vertex-vertex and vertex-edge distance maps.

Data: A connected graph $(\mathbb{G}^\bullet, \mathbb{G}^\times)$, a subset $X^\bullet$ of $\mathbb{G}^\bullet$, the number p of processors.
Result: The distance maps $D^\bullet_{X^\bullet}$ and $D^\times_{X^\bullet}$ to the set $X^\bullet$.

```
1  E := ∅; λ :=0;
2  Set to False all elements of a shared Boolean array Traversed of size |𝔾•|
3  (E_1, ...,E_p):= Partition(X•, p);
4  (F_1, ...,F_p):= Partition(𝔾×, p);
5  foreach processor i in {1, ...,p} do in parallel
6      foreach vertex x ∈ E_i do D•_{X•}(x) := λ; Traversed[x] := True ;
7      foreach edge e ∈ F_i do D×_{X•}(e) := ∞ ;
8  E:=Union(p, E_1, ..., E_p);
9  while E ≠ ∅ do
10     (E_1, ...,E_p):= Partition(E, p);
11     foreach processor i in {1, ..., p} do in parallel
12         S_i:=∅;
13         foreach x in E_i do
14             foreach vertex y adjacent to x in 𝔾 do // i.e., when {x, y} ∈ 𝔾×
15                 if D×_{X•}({x, y}) = ∞ then D×_{X•}({x, y}) := λ + 1;
16                 if test-and-set(Traversed[y]) = False then
17                     S_i:=S_i ∪ {y};
18                     D•_{X•}(y) := λ;
19     E:=Union(p, S_1, ..., S_p);
20     λ:=λ + 2;
```

4.2.2 Parallel partition and disjoint union algorithms

Let us now present the two auxiliary algorithms for the *Partition* and *Union* functions used in our parallel strategy.

The parallel partition algorithm (see Algorithm 5) consists of computing in parallel, with p processors, a *balanced partition* $\{E_1, \ldots, E_p\}$ of a set E. The partition is balanced in the sense that the k first sets of the partition contain $|E|/p+1$ elements whereas the following ones contain $|E|/p$ elements, where k is the remainder in the integer division of $|E|$ by p and where a/b denotes the quotient of the integer division of a by b. The

Algorithm 4: Parallel edge-edge and edge-vertex distance maps.

Data: A connected graph $(\mathbb{G}^\bullet, \mathbb{G}^\times)$, a subset $X^\times$ of $\mathbb{G}^\times$, the number p of processors.
Result: The distance map $D_{X^\times}^{(\times,\times)}$ and $D_{X^\times}^{(\times,\bullet)}$ to the set $X^\times$.

1 $E := \emptyset$; λ :=1;
2 Set to False all elements of a shared Boolean array $Traversed$ of size $|\mathbb{G}^\bullet|$
3 $(E_1, \ldots, E_p)$:= Partition$(\mathbb{G}^\bullet, p)$;
4 $(F_1, \ldots, F_p)$:= Partition$(\mathbb{G}^\times, p)$;
5 **foreach** *processor i in* $\{1, \ldots, p\}$ **do in parallel**
6 **foreach** *vertex* $x \in E_i$ **do** $D_{X^\times}^{(\times,\bullet)}(x) := \infty$;
7 **foreach** *edge* $e = \{x, y\} \in F_i$ **do**
8 **if** $\{x, y\} \in X^\times$ **then**
9 **if** $D_{X^\times}^{(\times,\bullet)}(x) == \infty$ **then** $Traversed[x] := True$; $D_{X^\times}^{(\times,\bullet)}(x) := 1$;
10 **if** $D_{X^\times}^{(\times,\bullet)}(y) == \infty$ **then** $Traversed[y] := True$; $D_{X^\times}^{(\times,\bullet)}(y) := 1$;
11 $D_{X^\times}^{(\times,\times)}(\{x, y\}) := 0$;
12 **else** $D_{X^\times}^{(\times,\times)}(\{x, y\}) := \infty$;
13 E:=Union$(p, E_1, \ldots, E_p)$;
14 **while** $E \neq \emptyset$ **do**
15 $(E_1, \ldots, E_p)$:= Partition(E, p);
16 **foreach** *processor i in* $\{1, \ldots, p\}$ **do in parallel**
17 S_i:=$\emptyset$;
18 **foreach** *x in* E_i **do**
19 **foreach** *vertex y adjacent to x in* $\mathbb{G}$ **do** // *i.e.*, when $\{x, y\} \in \mathbb{G}^\times$
20 **if** $D_{X^\times}^{\times}(\{x, y\}) = \infty$ **then** $D_{X^\times}^{\times}(\{x, y\}) := \lambda + 1$;
21 **if** *test-and-set*$(Traversed[y])$ = *False* **then**
22 S_i:=$S_i \cup \{y\}$;
23 $D_{X^\times}^{(\times,\bullet)} := \lambda + 2$;
24 E:=Union$(p, S_1, \ldots, S_p)$;
25 λ:=$\lambda + 2$;

elements of E, stored in an array of size $|E|$, are moved to arrays previously allocated for the subsets $E_1, \ldots, E_p$ in the order of their indices: the first set receives the first elements of the array E and so on (see Figure 4.1). Thus, each processor computes the index of the first and of the last element that must be copied (lines 2 to 6) before actually copying the elements of E located between the computed indices (line 7). The computation of the first and of the last indices can be done in constant time and the copying step is done in linear time with respect to $|E|/p$ (each processor moves at most $|E|/p + 1$ elements according to its number elements). For these steps, there is no dependence between processors. Thus, all processors can perform these steps in parallel.

The parallel *Union* algorithm (see Algorithm 6) computes the union of p disjoint sets $\{S_1, \ldots, S_p\}$ with p processors. The elements of each set are stored in an array and each processor p_i ($i \in \{1 \ldots, p\}$) copies the elements of the array S_i in the array E. The

Algorithm 5: Partition.

Data: An array E of $n = |E|$ elements, the number p of processors.
Result: A balance partition $(E_1, \ldots, E_p)$ of E.

1 **foreach** *processor* i *in* $\{1, \ldots, p\}$ **do in parallel**
2 **if** $i \leq (n \mod p)$ **then**
3 start$[i]$:= $(i-1) * (n/p+1)$; end$[i]$:= start$[i]$ + n/p;
4 **else**
5 start$[i]$:= $(n \mod p) * (n/p+1) + (i-1-(n \mod p)) * (n/p)$;
6 end$[i]$:= start$[i]$ + n/p - 1;
7 **foreach** j_i *in* $\{start[i], \ldots, end[i]\}$ **do** $E_i[j_i - start[i]] := E[j_i]$;

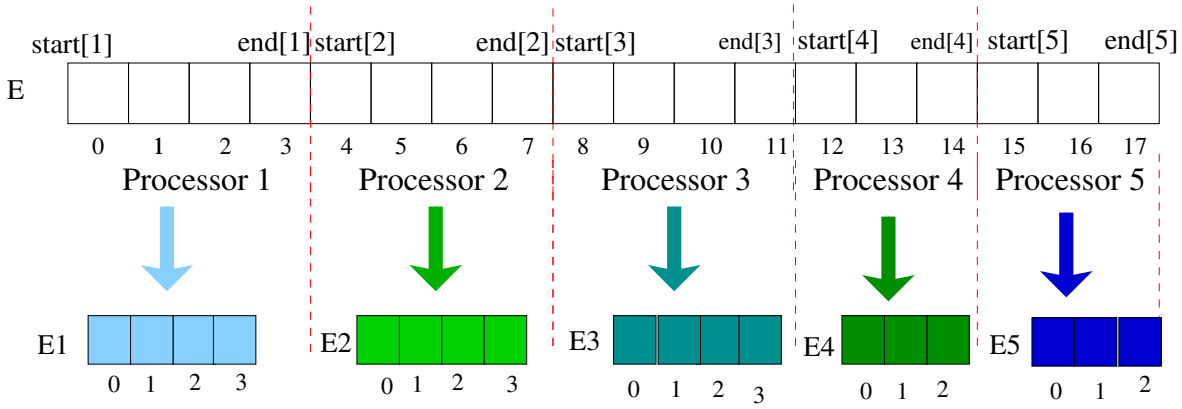


Figure 4.1: Illustration of the *Partition* algorithm with $p = 5$ processors.

elements of S_i are stored consecutively in the resulting array E from the index $start[i]$, where $start[i]$ is the sum of the cardinalities of the sets $S_1, \ldots, S_{i-1}$ (see Figure 4.2): $start[i] = \sum_{j \in \{1,\ldots,i-1\}} |S_j|$. Thus, our algorithm first computes the values $start[i]$ for any i in $\{1, \ldots, p\}$ (line 1) before actually copying the elements into E (line 3). Given the cardinalities $|S_1|, \ldots, |S_p|$, computing the values $start[i]$ for any i in $\{1, \ldots, p\}$ is known as the prefix-sum problem. It can be solved in parallel with p processors with a $O(\log_2 p)$ running-time algorithm [33]. Then, each processor i copies in parallel (line 3) the elements of S_i into E at the correct position.

Algorithm 6: Union.

Data: A series $S_1, \ldots, S_p$ of p sets, and the number p of processors.
Result: An array E whose elements constitutes the union of $\{S_1, \ldots, S_p\}$.

1 start = ParallelPrefixSum($|S_1|, \ldots, |S_p|$);
2 **foreach** *processor* i *in* $\{1, \ldots, p\}$ **do in parallel**
3 **foreach** j_i *in* $\{0, \ldots, |S_i| - 1\}$ **do** $E[start[i] + j_i] := S_i[j_i]$;

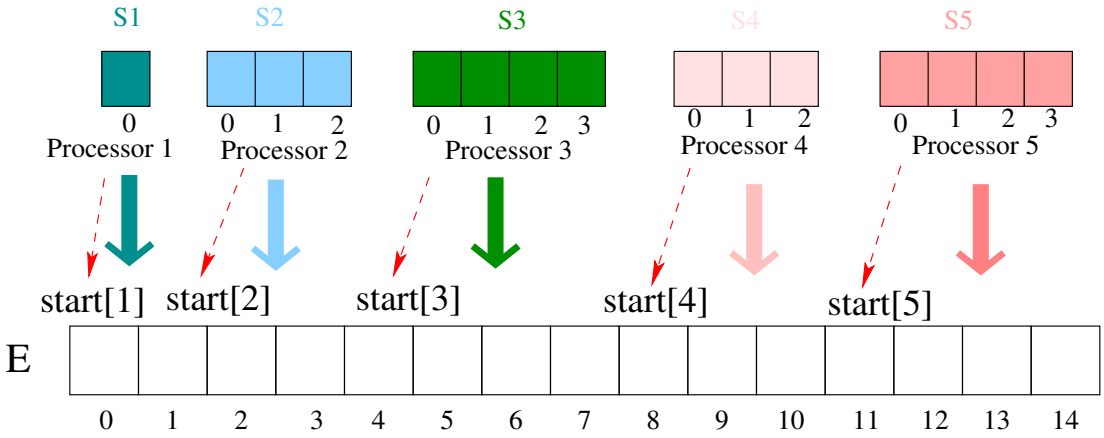
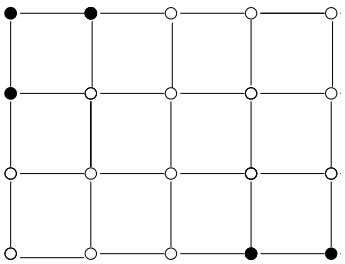


Figure 4.2: Illustration of the *Union* algorithm with $p = 5$ processors.

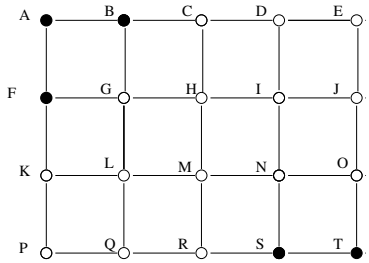
4.2.3 Example of step illustration of parallel algorithm

Before further studying Algorithm 3, let us analyze an execution example. To this end, the graph $\mathbb{G} = (\mathbb{G}^\bullet, \mathbb{G}^\times)$ and the set $X^\bullet = \{A, B, F, S, T\}$ shown Figure 4.3(a) are considered. For the sake of simplicity, in this example, let us focus only on the distance map $D^\bullet_{X\bullet}$ and let us discard the instructions concerning $D^\times_{X\bullet}$. Figure 4.3(b) illustrates the iterations of the main loop of Algorithm 3 with three processors, *i.e.*, with $p = 3$. After the initialization, we have $E = X^\bullet = \{A, B, F, S, T\}$, $\lambda = 0$, and $D^\bullet_{X\bullet}(A) = D^\bullet_{X\bullet}(B) = D^\bullet_{X\bullet}(F) = D^\bullet_{X\bullet}(S) = D^\bullet_{X\bullet}(T) = 0$. In the first iteration, the first step consists of partitioning the input set $E = \{A, B, F, S, T\}$ into 3 balanced subsets $E_1 = \{A, B\}$, $E_2 = \{F, S\}$, $E_3 = \{T\}$. Then, the sets E_1, E_2, and E_2 are assigned to the processors 1, 2, and 3, respectively. Each processor i, in parallel, explores the non-already traversed neighbors of the vertices in its subset E_i and inserts these neighbors into a private set. As shown in Figure 4.3, the resulting private sets S_1, S_2, and S_3 are such that $S_1 = \{C, G\}$, $S_2 = \{K, R, N\}$, $S_3 = \{O\}$. Observe that the vertex G (written in bold in the figure) was concurrently found by two distinct processors (numbered 1 and 2) since it is a successor of B, which belongs to E_1, and a successor of F, which belongs to E_2. But thanks to the test on the synchronized Boolean array *Traversed*, it was actually inserted only into S_1 and not into S_2. When a element is inserted into a set S_i, its distance map value is updated with the current value of λ plus 2 which here is equal to 2 since we are at the first iteration. Thus, we now have $D^\bullet_{X\bullet}(C) = D^\bullet_{X\bullet}(G) = D^\bullet_{X\bullet}(K) = D^\bullet_{X\bullet}(R) = D^\bullet_{X\bullet}(N) = D^\bullet_{X\bullet}(O) = 2$. Then, once the parallel neighbor search on E_1, E_2, and E_3 is over, the sets S_1, S_2, and S_3 are merged thanks to a call to the *Union* function resulting into the updated set $E = \{C, G, K, R, N, O\}$. This updated set E, the 2-level set of $D^\bullet_{X\bullet}$, is considered for the next iteration of the main while loop of the algorithm. Before starting a new iteration the value of λ is updated and therefore set to 2. After this first iteration, since E is nonempty, a second iteration of the main while loop is considered. In fact, on this example, four iterations of the main loop are

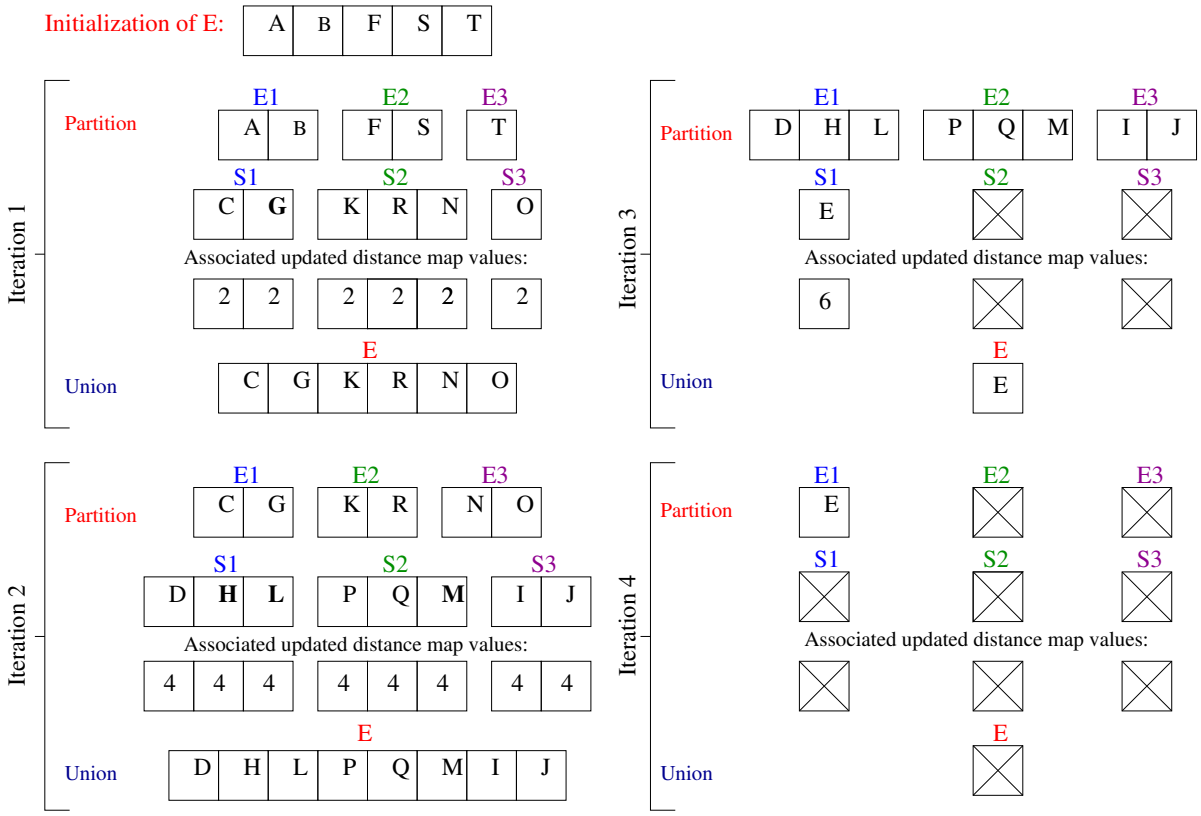
necessary to compute the resulting distance maps, as shown on Figure 4.3.



(a) A subset $X^{\bullet}$ of vertices (bold)



(b) Labeling of the vertices of $\mathbb{G}$.



(c) The different steps of our parallel algorithm.

Figure 4.3: Illustration of the proposed parallel algorithm (Algorithm 3): a step-by-step execution example with $p = 3$ processors.

4.3 Complexity analysis

In this part, the worst-case time complexity of our parallel algorithms is analyzed. This complexity depends of the ratio, called the unbalancing balancing factor of the graph (Definition 27), of the maximum degree of a vertex of the graph over the minimum one. As seen above, for the sake of simplicity, we only focus on Algorithm 3, but all the study can also be adapted to Algorithm 4.

The complexity of Algorithm 3 is analyzed under an assumption about the graph and set under scrutiny. Indeed, in some "degenerated" cases, which we seek to avoid by this assumption, the distance map computation problem becomes sequential. This is the case, in particular, when the graph depicts a "linear shape", *e.g.*, when each vertex is adjacent to at most two other vertices. An example of such graph is shown in Figure 4.4a. In order to handle such problematic situations, the following notion of a regular set is proposed.

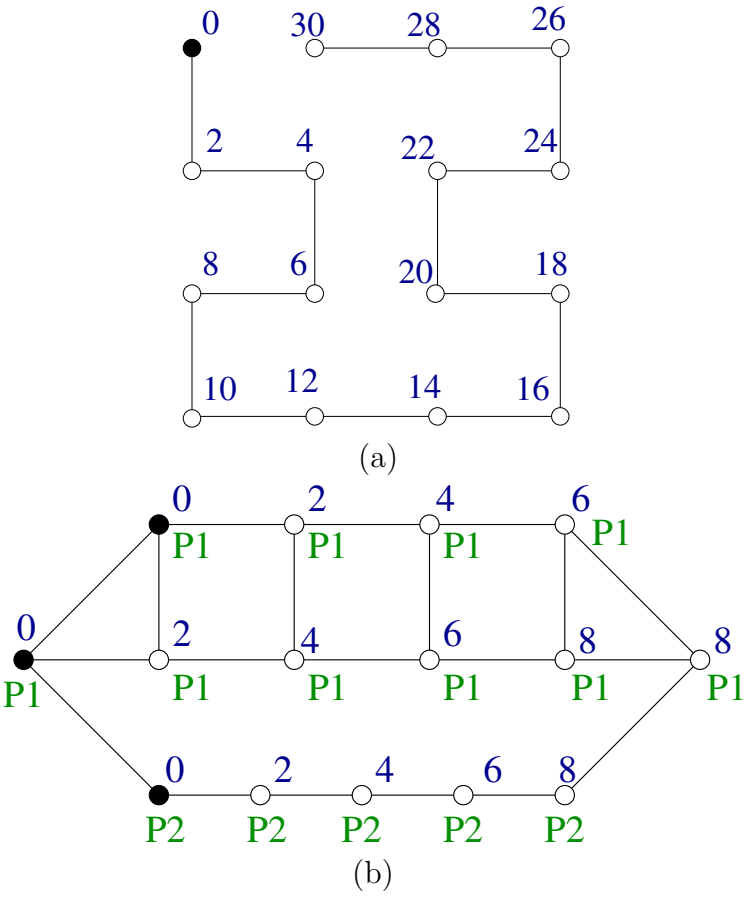


Figure 4.4: (a) A graph $\mathbb{G}$ and a subset $X^\bullet$ (in black) of vertices that is 1-regular but not 2-regular. The distance map $D^\bullet_{X^\bullet}$ is given by the blue numbers. (b) A graph whose unbalancing factor is $\frac{3}{2}$ used to illustrate a situation of unbalanced workload obtained with Algorithm 3 (see text).

Definition 26 (regular set)**.** *Let p be a positive integer and let $X^\bullet$ be a subset of $\mathbb{G}^\bullet$. We say that $X^\bullet$ is* p-regular *(for $\mathbb{G}$) if every nonempty level-set of $D^\bullet_{X^\bullet}$ contains at least p elements.*

For instance, if we consider the graph $\mathbb{G}$ of Figure 4.4(a), the set $X^\bullet$ (black vertices in Figure 4.4(a)) is 1-regular but not 2-regular since every level set of $D^\bullet_{X^\bullet}$ (represented by blue numbers in the figure) contains a single vertex. It can be seen that when Algorithm 3 is applied to the graph $\mathbb{G}$ and the set $X^\bullet$ of this example, then, at each iteration, there is a single nonempty set among the sets $E_1, \ldots, E_p$ obtained after the call to the *Partition*

function. Thus, it can be observed, that, at each iteration, there is no parallel calculus done in the parallel loop line 11. Hence, in order to study the complexity of Algorithm 3, it is assumed in the following that the input set $X^\bullet$ is p-regular, p-being the number of available processors. This ensures that any of the sets $E_1, \ldots, E_p$, obtained after the call to the *Partition* function, is nonempty. Hence, every processor processes a nonempty set E_i at every iteration of the parallel loop line 11.

Another important issue for studying the time-complexity of Algorithm 3 is related to the degrees of the vertices of $\mathbb{G}$. Indeed, after line 10 of Algorithm 3, the sets $\{E_i \mid i \in \{1, \ldots, p\}\}$ are balanced and each processor must traverse all the neighbors of exactly one of these sets. Thus, if all vertices of the graph have the same degree (*i.e.*, if they all belong to the same number of edges), the workload is well balanced over the processors. On the other hand, if the degree varies from one vertex to another one, then, one may easily end up in a situation which is unfair for one of the processors. An example of such situation is provided in Figure 4.4(b) which illustrates a possible execution of Algorithm 3 on the set $X^\bullet$ of black vertices with two processors called P1 and P2. It can be seen that every level set of $D^\bullet_{X^\bullet}$ contains three vertices which are partitioned at line 10 of Algorithm 3 into two sets E_1 and E_2, handled respectively by P1 and P2. In the figure, we indicate in green the processor which traverse every vertex. It can be seen that the obtained partition is, at each iteration, such that E_1 contains two vertices and E_2 contains one vertex. Furthermore, each vertex in E_1 belongs to three edges, whereas each vertex in E_2 belongs to a single edge. Overall, it can thus be checked that P1 must browse over 30 edges (10 vertices contained in 3 edges) in the loop line 14 whereas P2 browses over only 10 edges (5 vertices contained in two edges), leading to an unbalanced workload between P1 and P2. The occurrence of such situation and its amplitude can be bounded thanks to the notion of unbalancing factor of a graph, that is introduced hereafter (Definition 27). Then, the time-complexity of Algorithm 3, established in Theorem 29, can be expressed as a function of this unbalancing factor.

We recall that, if x is a vertex of $\mathbb{G}$, the *degree of* x, denoted by $d(x)$, is the number of edges that contain x, *i.e.*, $d(x) = |\{\{x, y\} \in \mathbb{G}^\times\}|$. Furthermore, we denote by $d_{\min(\mathbb{G})}$ (resp. $d_{\max(\mathbb{G})}$) the minimum (resp. maximum) of the degrees of the vertices of $\mathbb{G}$: $d_{\min}(\mathbb{G}) = \min\{d(x) \mid x \in \mathbb{G}^\bullet\}$ (resp. $d_{\max}(\mathbb{G}) = \max\{d(x) \mid x \in \mathbb{G}^\bullet\}$).

Definition 27 (unbalancing factor)**.** *The* unbalancing factor of $\mathbb{G}$*, denoted by* $\beta(\mathbb{G})$*, is the fraction:*

$$\beta(\mathbb{G}) = \frac{d_{\max}(\mathbb{G})}{d_{\min}(\mathbb{G})}. \tag{4.1}$$

For instance, the unbalancing factor of the graph $\mathbb{G}$ shown in Figure 4.4(b) is equal to $3/2$ since $d_{\min}(\mathbb{G}) = 2$ and $d_{\max}(\mathbb{G}) = 3$. The unbalancing factor of the 4-adjacency graph of Figure 4.3 is 2. Note that if we modify this graph by identifying the vertices of the first

and last lines and those of the first and last columns (so that a torus is obtained), then the degree of every vertex in the obtained graph is equal to 4, leading to an unbalancing factor of 1.

The time complexity of Algorithm 3 depends of the unbalancing factor of $\mathbb{G}$. The next lemma is an important result in order to establish this time complexity.

Lemma 28. *Let p be a positive integer and let $X^\bullet$ be a p-regular subset of $\mathbb{G}^\bullet$. When Algorithm 3 is executed on $\mathbb{G}$, $X^\bullet$, and p, then the following statements hold true.*

1. *The instruction lines 15 and 16 are executed less than $\frac{4\beta(\mathbb{G})}{2\beta(\mathbb{G})+p+1}|\mathbb{G}^\times|$ times by each processor;*
2. *Using an array of linked lists to represent the graph $\mathbb{G}$ (*i.e.*, one linked list for any vertex to store its adjacent edges), the continuation condition in the for each loop at line 14 is tested less than $\frac{2}{p+1}|\mathbb{G}^\bullet| + \frac{4\beta(\mathbb{G})}{2\beta(\mathbb{G})+p+1}|\mathbb{G}^\times|$ times by each processor.*

Proof. At every iteration of the main loop line 9 the set E is the λ-level set of $D^\bullet_{X^\bullet}$. Let n_λ be the cardinality of E. Then, by definition of the *Partition* function, for any i in $\{1,\ldots,p\}$, the number $n_{\lambda,i}$ of elements in the set E_i obtained at line 10 is either equal to n_λ/p or to $n_\lambda/p+1$, where a/b denotes the quotient in the integer division of a by b. Let $\delta^\times(E)$ (resp. $\delta^\times(E_i)$) be the set that contains any edge with a vertex in E (resp. in E_i). Let us also denote by m_λ and $m_{\lambda,i}$ the cardinalities of $\delta^\times(E)$ and $\delta^\times(E_i)$, respectively. Note that for a given value of λ, the instruction lines 15 and 16 are executed exactly $m_{\lambda,i}$ times by the processor i. In order to establish Lemma 28. 1, we are going to prove that $m_{\lambda,i} \leq \frac{2\beta(\mathbb{G})}{2\beta(\mathbb{G})+p-1} m_\lambda$. To this, end we are going to distinguish two cases.

– Let us first assume that $n_\lambda > p$. Thus, we have $q \leq n_{\lambda,i} \leq 2q$, where q is the quotient in the integer division of n_λ by $(p+1)$. Since $n_\lambda > p$, the quotient q is nonnull. By definition of $d_{\min}(\mathbb{G})$ and $d_{\max}(\mathbb{G})$, we have $d_{\max}(\mathbb{G}).n_{\lambda,i} \leq m_{\lambda,i} \leq d_{\max}(\mathbb{G}).n_{\lambda,i}$. Hence, we deduce that $d_{\min}(\mathbb{G})q \leq m_{\lambda,i} \leq 2d_{\max}(\mathbb{G})q$. By definition of m_λ, we have:

$$\frac{m_{\lambda,i}}{m_\lambda} = \frac{m_{\lambda,i}}{m_{\lambda,i} + \sum_{j\in\{1,\ldots,p\}\setminus\{i\}} m_{\lambda,j}} = 1 - \frac{\sum_{j\in\{1,\ldots,p\}\setminus\{i\}} m_{\lambda,j}}{m_{\lambda,i} + \sum_{j\in\{1,\ldots,p\}\setminus\{i\}} m_{\lambda,j}}. \tag{4.2}$$

Hence, since $m_{\lambda,i} \leq 2d_{\max}(\mathbb{G})q$, we deduce that:

$$\frac{m_{\lambda,i}}{m_\lambda} \leq 1 - \frac{\sum_{j\in\{1,\ldots,p\}\setminus\{i\}} m_{\lambda,j}}{2d_{\max}(\mathbb{G})q + \sum_{j\in\{1,\ldots,p\}\setminus\{i\}} m_{\lambda,j}} \quad \text{, which can be rewritten as:} \tag{4.3}$$

$$\frac{m_{\lambda,i}}{m_\lambda} \leq \frac{2d_{\max}(\mathbb{G})q}{2d_{\max}(\mathbb{G})q + \sum_{j\in\{1,\ldots,p\}\setminus\{i\}} m_{\lambda,j}} \tag{4.4}$$

Since, $d_{\min}(\mathbb{G})q \leq m_{\lambda,j}$, for any $j \in \{1,\ldots,p\}$ we deduce that:

$$\frac{m_{\lambda,i}}{m_\lambda} \leq \frac{2d_{\max}(\mathbb{G})q}{2d_{\max}(\mathbb{G})q + (p-1)d_{\min}(\mathbb{G})q} \quad \text{, which can be simply rewritten as:} \tag{4.5}$$

$$\frac{m_{\lambda,i}}{m_\lambda} \leq \frac{2d_{\max}(\mathbb{G})}{2d_{\max}(\mathbb{G}) + (p-1)d_{\min}(\mathbb{G})} \quad \text{, which can again be rewritten as:} \tag{4.6}$$

$$\frac{m_{\lambda,i}}{m_\lambda} \leq \frac{2\frac{d_{\max}(\mathbb{G})}{d_{\min}(\mathbb{G})}}{2\frac{d_{\max}(\mathbb{G})}{d_{\min}(\mathbb{G})} + (p-1)\frac{d_{\min}(\mathbb{G})}{d_{\min}(\mathbb{G})}} \quad \text{, which can be simplified as:} \tag{4.7}$$

$$\frac{m_{\lambda,i}}{m_\lambda} \leq \frac{2\beta(\mathbb{G})}{2\beta(\mathbb{G}) + p - 1}. \tag{4.8}$$

Thus, since m_λ is positive, we deduce that:

$$m_{\lambda,i} \leq \frac{2\beta(\mathbb{G})}{2\beta(\mathbb{G}) + p - 1} m_\lambda. \tag{4.9}$$

– Let us now assume that $n_\lambda = p$. In this case, for any $i \in \{1,\ldots,p\}$, we have $n_{\lambda,i} = 1$ and $d_{\min}(\mathbb{G}) \leq m_{\lambda,i} \leq d_{\max}(\mathbb{G})$. Using similar arguments as in the previous case, we deduce that

$$m_{\lambda,i} \leq \frac{\beta(\mathbb{G})}{\beta(\mathbb{G}) + p - 1} m_\lambda \tag{4.10}$$

Hence, since $2\beta(\mathbb{G}) \geq \beta(\mathbb{G}) \geq 0$, we also have in this second case:

$$m_{\lambda,i} \leq \frac{2\beta(\mathbb{G})}{2\beta(\mathbb{G}) + p - 1} m_\lambda. \tag{4.11}$$

Let us set $m_i = \sum_{\lambda\in\mathbb{N}} m_{\lambda,i}$. It can be seen that m_i is the overall number of executions of the instruction lines 15 and 16 by the processor i. Since the set of all level-sets of $D^{\bullet}_{X^{\bullet}}$ partitions $\mathbb{G}^{\bullet}$ and since every edge contains exactly two vertices of $\mathbb{G}$, we deduce that $\sum_{\lambda\in\mathbb{N}} m_\lambda = 2.|\mathbb{G}^{\times}|$. Hence, from Equations 4.9 and 4.11, we deduce the following inequation that establishes Lemma 28.1:

$$m_i \leq \frac{4\beta(\mathbb{G})}{2\beta(\mathbb{G}) + p - 1} |\mathbb{G}^{\times}|. \tag{4.12}$$

Let us now establish Lemma 28.4.3. To this end, it can be seen that there are, for each processor $i \in \{1,\ldots,p\}$, m_i positive continuation tests at line 14 of Algorithm 3. It can also be seen that, for every value of λ, there is one negative test for each element in E_i. Hence, for any processor i and any value of λ, there is at most $\frac{2n_\lambda}{p+1}$ negative tests. Hence, since the level-set of $D^{\bullet}_{X^{\bullet}}$ partitions $\mathbb{G}^{\bullet}$, and since n_λ is the number of elements in the λ-level set of $D^{\bullet}_{X^{\bullet}}$, we deduce that, for any processor, there are at most $\frac{2|\mathbb{G}^{\bullet}|}{p+1}$ negative tests at line 14 of Algorithm 3. Hence, the condition in the for each loop at line 14 is tested

less than $\frac{2}{p+1}|\mathbb{G}^\bullet| + \frac{4\beta(\mathbb{G})}{p+1+2.\beta(\mathbb{G})}|\mathbb{G}^\times|$ times by each processor.

□

Let us now analyze the complexity of Algorithm 3. At every iteration of the main loop (line 9), the set E is the λ-level set of $D^\bullet_{X^\bullet}$. Since the *Partition* function (Algorithm 5) presented in the previous section runs in linear time with respect to $|E|/p$ and since the level sets of $D^\bullet_{X^\bullet}$ partitions $\mathbb{G}^\bullet$, the time complexity of line 10 is $O(|\mathbb{G}|/p)$. Furthermore, after the execution of line 10, we always have $|E_i| \leq |E|/p + 1$, for any $i \in \{1, \ldots, p\}$. Thus, the complexity of line 13 is $O(|\mathbb{G}|/p)$. From Lemma 28.4.3, we deduce that the time complexity of line 14 is $O(\frac{\beta(\mathbb{G})}{\beta(\mathbb{G})+p}|\mathbb{G}^\times| + \frac{1}{p}|\mathbb{G}^\bullet|)$ and, from Lemma 28.1, we deduce that the complexity of lines 15 and 16 is $O(\frac{\beta(\mathbb{G})}{\beta(\mathbb{G})+p}|\mathbb{G}^\times|)$. Let us finally analyze the complexity of line 19. By Lemma 28.1, the overall number of insertions in a set S_i is less than $\frac{4\beta(\mathbb{G})}{2\beta(\mathbb{G})+p+1}|\mathbb{G}^\times|$, for any i in $\{1, \ldots, p\}$. In the function *Union* (Algorithm 6), each element of S_i is copied once in E (line 3) by the processor numbered i. Thus, when *Union* is called by Algorithm 3, the overall complexity of line 3 in Algorithm 6 is $O(\frac{\beta(\mathbb{G})}{\beta(\mathbb{G})+p}|\mathbb{G}^\times|)$. Furthermore, since there is one call to *ParallelPrefixSum* for each call to *Union*, there is one call of *ParallelPrefixSum* for each level set of $D^\bullet_{X^\bullet}$. Therefore, if K is the number of level set of $D^\bullet_{X^\bullet}$, the time complexity of line 19 is $O(\frac{\beta(\mathbb{G})}{\beta(\mathbb{G})+p}|\mathbb{G}^\times| + K \log_2 p)$ since the complexity of *ParallelPrefixSum* is $O(\log_2 p)$. It can be also seen that the worst-case time complexity of the initialization steps (lines 1 to 8) is $O(\frac{\beta(\mathbb{G})}{\beta(\mathbb{G})+p}|\mathbb{G}^\times| + \log_2 p)$. Hence, the following theorem can be stated.

Theorem 29. *Let p be a positive integer and let $X^\bullet$ be a p-regular subset of $\mathbb{G}^\bullet$. Then, Algorithm 3 terminates in $O(\frac{1}{p}|\mathbb{G}^\bullet| + \frac{\beta(\mathbb{G})}{p+\beta(\mathbb{G})}|\mathbb{G}^\times| + K \log_2 p)$ time, where K is the number of distinct level sets of $D^\bullet_{X^\bullet}$.*

Observe that, when the unbalancing factor of $\mathbb{G}$ is less than a small constant, the complexity of Algorithm 3 reduces to $O(\frac{|\mathbb{G}^\bullet|+|\mathbb{G}^\times|}{p} + K \log_2 p)$. Considering the complexity of the sequential distance map algorithm (Algorithm 1), the best possible theoretical complexity for a parallelization of Algorithm 1 with p processors would be $O(\frac{|\mathbb{G}^\bullet|+|\mathbb{G}^\times|}{p})$. This complexity is reached with Algorithm 3 (in the case of small unbalancing factor) up to a $K \log_2 p$ term, which is due to the computation of the parallel dynamic partition. On the other hand, it can be observed that, when the unbalancing factor of $\mathbb{G}$ is high compared to the number p of processors, the complexity of Algorithm 3 tends to $O(\frac{1}{p}|\mathbb{G}^\bullet| + |\mathbb{G}^\times| + K \log_2 p)$. Hence, in this case the complexity gain over the sequential algorithm is less interesting. Fortunately, in practical cases, the considered graphs have a small unbalancing factor. For instance, as mentioned before in this section, when an image is structured with the 4-adjacency relation, the unbalancing factor of the resulting graph is two (or even one if the image borders are identified). Hence, in the next chapter we aim to confirm that in practical case, the proposed parallel algorithm offers an interesting speedup over the sequential version.

4.4 Conclusion

In this chapter, we proposed a parallel strategy that leads to a fast computation of our new distance maps on graphs, hence all morphological operators of [14]. Our parallelization strategy is based on dynamic partitioning of the input set which acts on successive level-sets of the distance maps. Each level set being partitioned and then traversed in parallel. Under some proved assumptions about the graph (unbalancing factor) and set (p-regular) under consideration, the parallel algorithms run in $O(n/p + K \log_2 p)$ time complexity, where n, p and k are the size of the graph, the number of available processors, and the number of distinct level sets of the distance map, respectively.

In the following experimental chapter, we will confirm in practical cases that our parallel algorithms offer an interesting speedup over the sequential version.

Chapter 5

Evaluation and Experimental results of parallel implementation of Distance maps on graph on shared memory architectures

Contents

5.1 Introduction

In this chapter, we start by presenting the target platform used for multithreaded implementation of our proposed parallel strategy. Then, we provide in Section 5.3 a description of the 2D image and 3D textured meshes datasets used for experimental evaluations. In

order to assess the regularity assumptions of Theorem 29 on the considered datasets, an analysis is given in Section 5.4. After, we analyze in Section 5.5 the results obtained by applying the implementations of the proposed sequential and parallel algorithms on the target architecture to the considered datasets. First, Section 5.5.1 describes the measures assessing the performance of the proposed implementations (execution times and speedups). Then, Section 5.5.2 presents the evaluation results.

5.2 Target Platform and Multithreaded Implementation

Today, most modern computer systems include multicore chips that support shared memory in hardware. Therefore, parallel computing on shared memory multicores architecture is now very popular. Because of the high availability of this architecture, the large amount of available computing paradigms for these architectures and the parallelism in our proposed strategy, we decided to carry out experiments on a two quadcore Intel(R) Xeon(R) E5630 processors, *i.e.*, on a machine with 8 cores. The Intel Xeon E5630 (Westmere-EP which have the same microarchitecture of the Nehalem-EP) is a 32nm quadcore processor at 2.53 GHz. Its high level overview is provided in Figure 5.1. Each core has a private 32KB L1 and 256 KB L2 cache, while the 12 MB L3 cache is shared across the cores on a socket. Each core supports Simultaneous Multi-Threading (SMT), allowing two threads to share processing resources in parallel on a single core. The Nehalem-EP has three DDR3 memory channels per chip, each one is capable of performing simultaneous read and write transactions, effectively doubling memory bandwidth. The 3-channel Quick Path Interconnect (QPI) at 5.86 GigaTransactions per second gives over a 25.6 GB/sec bandwidth to the other neighboring processors in a blade [1].

With such multicore-multithreaded shared memory architecture, up to 8 threads can run in parallel, emulating the p processors (with $p \leq 8$) of the theoretical parallel model used in the previous Chapter. Therefore, a multithreaded implementation of our algorithms is devised for the above described target architecture. POSIX Threads (Pthreads) library on a Linux system is considered for this implementation since it allows to handle low-level synchronization between threads. We remind that such synchronization is required to implement the proposed parallelization strategy which controls precisely the load balancing between the available processors. It can also be noticed that the use of MPI and PVM is not adapted to our target architecture (shared memory) and that the use of OpenMP does not allow one to control precisely the load balancing between processors as proposed in our algorithms.

Despite the distribution of data in our parallel algorithms, each thread has access to an overlap data (the Boolean array $Traversed$) in order to check if a vertex has been already

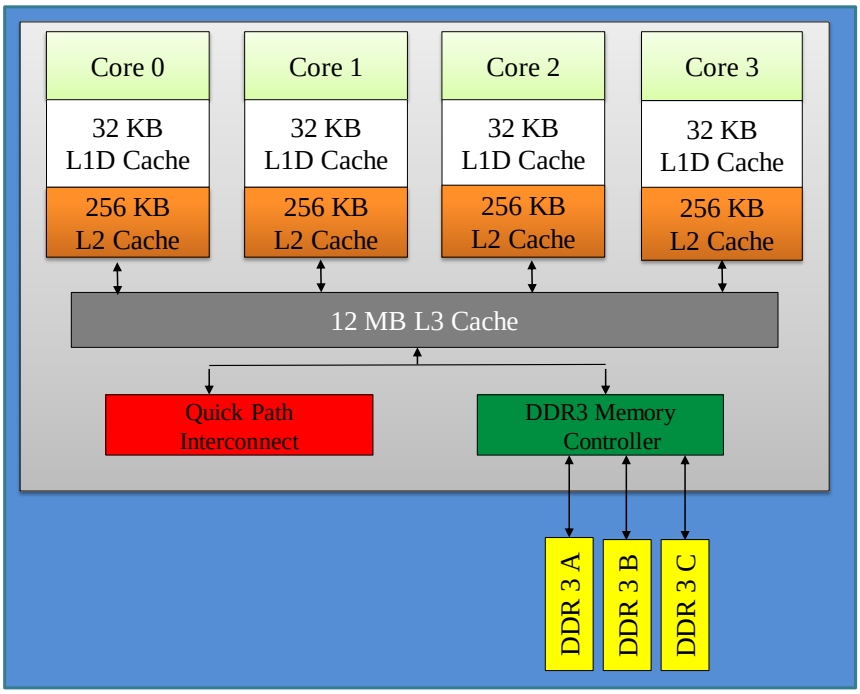


Figure 5.1: High-Level Overview of Intel Nehalem-EP.

traversed or not. The only access to this shared memory is done with the *test-and-set* instruction at line 16 (resp. line 21) of Algorithm 3 (resp. Algorithm 4). This instruction involves a synchronization between processors or threads. The *test-and-set* instruction can be implemented with Pthreads library using a mutex variable. To this end, one must consider a mutex variable instead of a Boolean one. Then, the *test-and-set* instruction is implemented due to a Pthreads function. This function is non-interruptible, locks a mutex and returns 0 if the mutex is unlocked at the time of the call or returns a positive value otherwise (*i.e.*, if a mutex is already locked when the function is called). With this test-and-set implementation, a Boolean array *Traversed* of Algorithms 3 and 4 is implemented with an array of mutexes that are initially unlocked.

5.3 Datasets description

5.3.1 2D image datasets

We consider two 2D image datasets in the experimental evaluations,. The first image dataset is the one of [14]. It is a set of 33 binary images of different size (786 x 540, 800 x 600, 700 x 1285) and various contents (see some samples in Figure 5.2.a and Figure 5.3.a). The images of this dataset have been obtained by adding black and white random noise of

different sizes and shapes to bitmap images of simple shapes (mainly letters). The second one is the *Standard* dataset which is a set of 12 images found frequently in the literature such as Lena, peppers, cameraman, lake, *etc.* All images of this dataset are grayscale and of the same size (512 x 512) (see some examples in Figure 5.4(a)).

Elementary mathematical morphology operators are often considered as a post processing step after the binarization of images since it allows to reduce noise and regularize contours. Therefore, we apply Otsu thresholding method [36] to binarize the images of the standard dataset (see Figure 5.4(b)). Otsu thresholding method considers as a very popular technique, which applies an automatic threshold to reduce the variance inter-classes between foreground and background. Then, after this step, two databases of typical binary images are obtained. These binary images are processed with the mathematical morphological operators described in chapter 2 (see examples of results in Figure 5.2(b), in Figure 5.3(b) and Figure 5.4(c)). In order to filter an image with the operators studied in this thesis, the image domain is equipped with a graph. For each image, the vertex set of the considered graphs is made of the image pixels and the edge set is given by the well known 4-adjacency relation on these pixels. For each of these two image datasets, Table 5.1 provides the image sizes, the number of images of each size, the average number of white pixels (*i.e.*, the vertices $X^\bullet$), and the average number of level-sets in the considered distance maps.

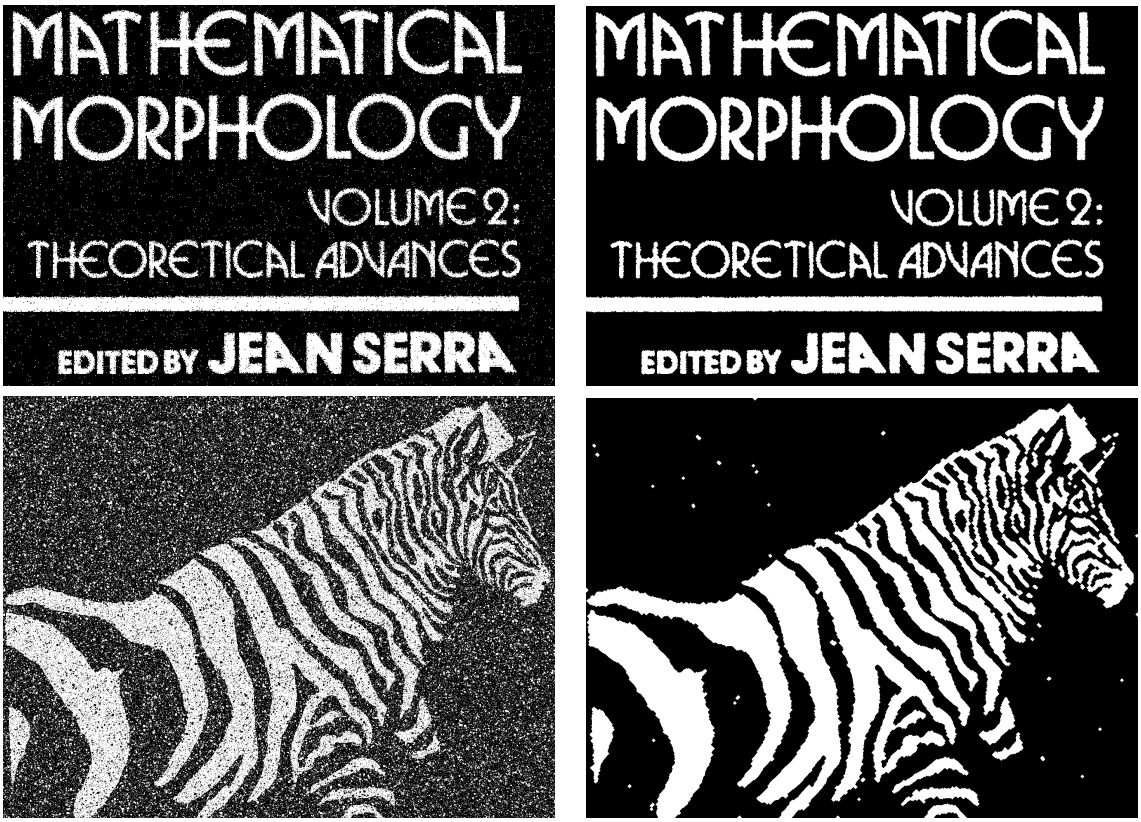
Image DSs:	Standard DS	DS of [14]		
Images size	512 × 512	786 × 540	800 × 600	700 × 1285
Number of images	12	2	29	2
Avg. number of vertices in $X^\bullet$	17 090	321 499	372 044	663 259
Avg. number of level-sets of $D_{X^\bullet}^{(\bullet,\bullet)}$	66	4	11	6

Table 5.1: Main features of the image datasets used for experimental evaluation. In the table, DS stands for dataset.

5.3.2 3D triangular meshes dataset

We also consider for the experimental evaluation a dataset of 3 three-dimensional triangular meshes. Each mesh is equipped with two binary black and white textures, one being obtained by adding random noise to another. Hence, overall, this dataset is made of 6 binary-textured meshes (see, *e.g.*, Figure 5.5).

In order to filter textured meshes with operators on graphs (see Figure 5.5), texture domains are equipped with graphs. More precisely, given a mesh, each corner of each triangle is a vertex of the associated graph and two vertices of the graph are linked by an edge if the associated triangle corners belong to a same triangle or if the associated triangle corners are at the same spatial position and belong to two adjacent triangles. Table 5.2



(a) initial binary images. (b) results of graph ASF.

Figure 5.2: Illustrations of mathematical morphology ASF on graphs on some images of the dataset used in [14].

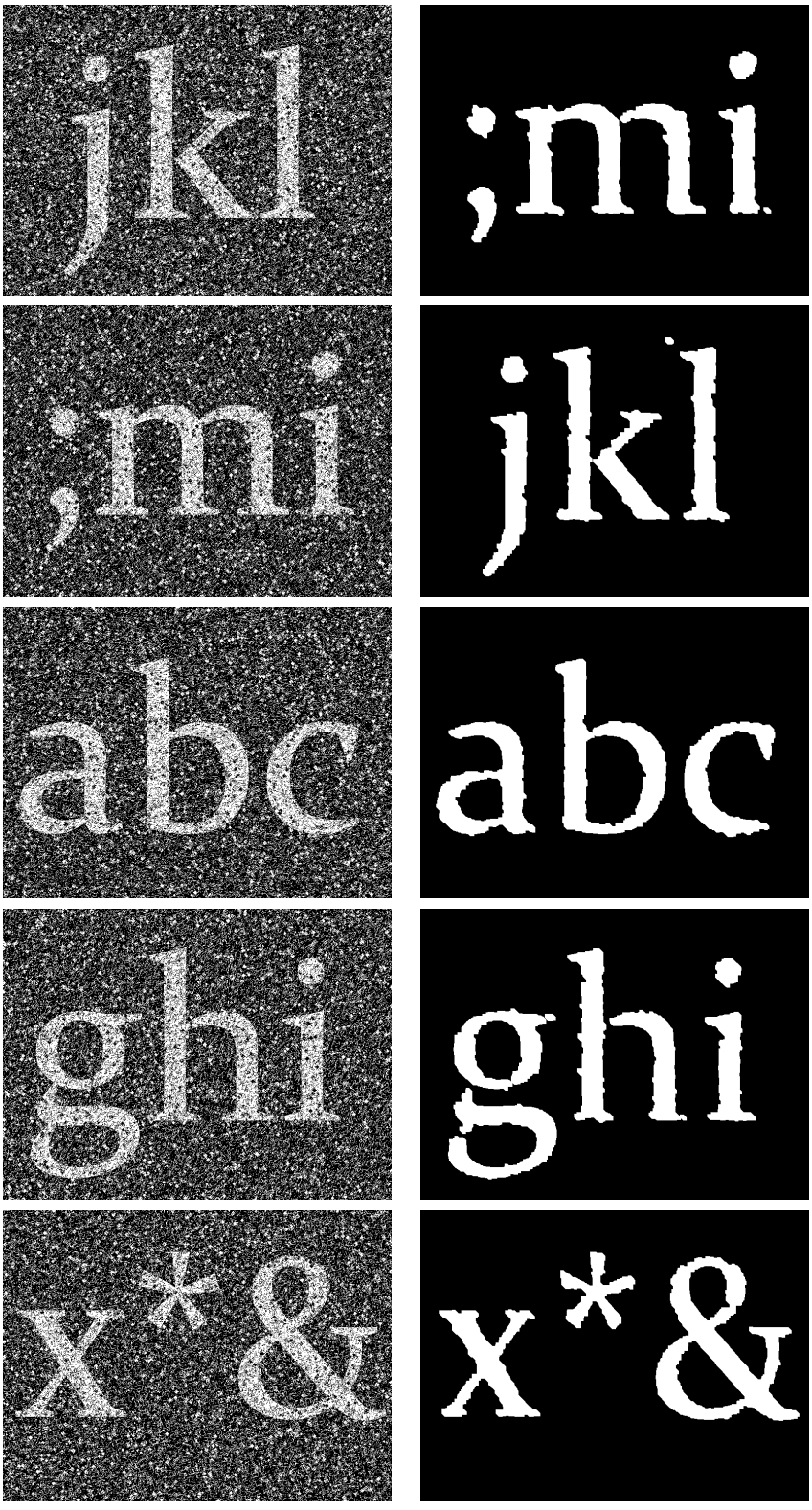
provides the number of vertices and of edges of the considered graphs, the average number of white vertices in the textures, and the average number of level-sets in the distance maps associated with the textures.

Number of textured meshes	6
Number of vertices/edges per graph	98 226 / 392 904
Average number of white vertices (vertices in $X^\bullet$)	45 102
Average number of level-sets in $D_X^{(\bullet;\bullet)}$	31

Table 5.2: Main features of the binary-textured mesh dataset used for experimental evaluations.

5.4 Experimental-assessment dataset and regularity-assumption assessment

Let us now analyze the datasets described above in order to assess the regularity assumptions of Theorem 29 defined in chapter 4. Indeed, Theorem 29 asserts that if



(a) initial binary images. (c) results of graph ASF

Figure 5.3: Illustrations of mathematical morphology ASF on graphs on some images of the dataset used in [14].

i) the unbalancing factor of the graph $\mathbb{G}$ is lower than a small constant and if ii) the considered set of vertices is p-regular, then the time complexity of our algorithms is $O((|\mathbb{G}^\bullet|+|G^\times|)/p+K\log_2 p)$, where K is the number of nonempty level-sets of the distance maps and where p is the number of available processors.

5.4.1 Case of 2D image datasets

Our parallel algorithms are applied to sets of pixels of 2D binary images equipped with the 4-adjacency relation. As mentioned in chapter 4 (see Section 4.3), the unbalancing factor of the graphs associated to the images is 2. Thus, assumption i) is practically verified for the considered datasets. In order to assess assumption ii), we analyze the number of vertices in each level-set of the considered distance maps and compare these numbers with the numbers of processors used in the experiments presented in Section 5.5. Due to the chosen target architecture, this number is always lower than 8. For each dataset, the proportion of level-sets with less than k vertices is estimated for $k \in \{2, 4, 6, 8\}$. The result of this analysis is presented in Table 5.3. To perform this analysis a total of 797 (resp. 716) level-sets are analyzed for the 12 (resp. 33) images of the standard dataset (resp. the dataset of [14]. Observe that the average number of level set is significantly less for the dataset of [14] than for the standard dataset. This can be explained by the addition of random black and white noise in the later images which implies that every image pixel is close to white pixel and to a black one. For the standard dataset, 3 level-sets contain less than 2 vertices (0.37%), 10 level-sets contain less than 4 vertices (1.25%), 16 level-sets contain less than 6 vertices (2 %), and 23 level-sets contain less than 8 vertices (2.88%). For the dataset of [14], 6 level-sets contain less than 2 vertices (0.83%), 11 level-sets contain less than 4 vertices (1.53%), 18 level-sets contain with less than 6 vertices (2.51%) and 20 level-sets contain less than 8 vertices (2.79%).

k	2	4	6	8
Standard dataset	0.37%	1.25%	2%	2.88%
Image dataset used in [14]	0.83%	1.53%	2.51%	2.79%

Table 5.3: Proportion of level-sets of the vertex-vertex distance maps with less than k vertices, the total number of analyzed level-sets being equal to 797 (resp. 716) for the 12 images (resp. 33 images) of the standard dataset (resp. the dataset of [14]).

5.4.2 Case of textured mesh dataset

In this case, by construction the unbalancing factor of the graph associated to the meshes is 1 because each vertex is exactly linked to four other vertices. Thus, assumption i) is practically verified. To assess assumption ii), we follow the same process applied to the

2D image datasets. The result of the analysis is presented in Table 5.4. To perform this analysis a total of 187 level-sets are analyzed for the 6 binary-textured meshes. We obtain no level-set contains less than 2 vertices (0%), 3 level-sets contain less than 4 vertices (1.6%), 5 level-sets contain with less than 6 vertices and less than 8 vertices (2.67%). Hence, the number and the proportion of level-sets with less than $k \in 2, 4, 6, 8$ vertices is relatively small. It is also observed that the level-sets with few vertices tend to appear for the highest distance values.

k	2	4	6	8
Binary-textured mesh dataset	0%	1.6%	2.67%	2.67%

Table 5.4: Proportion of level-sets of the vertex-vertex distance maps with less than k vertices, the total number of analyzed level-sets being equal to 187 for 6 textured meshes of the textured mesh dataset.

These analysis confirm that the assumptions defined in Theorem 29 are reasonable for the considered datasets on experiments in this work.

5.5 Performance evaluation and experimental results

5.5.1 Performance Evaluation

The performance of the proposed sequential and parallel algorithms (chapters 3 and 4) is assessed through the execution times obtained with the implementation and target platform described above. For assessing the gain of the parallel implementation over the sequential one, execution time and speedup measure are considered in our evaluations.

The reported execution times are obtained with Gprof. Gprof is a profiling software available on GNU Linux. This profiler has been introduced in 1982 [24] by group of researchers of the Berkeley University. Gprof measures the time spent on each function for a particular execution of a program as well as the number of times each function is called. Each measurement reported in Section 5.5.2 is the average of the executions times of forty runs of the same program on the same data.

5.5.2 Experimental results

The results presented in this section are obtained by applying the proposed sequential and parallel algorithms (namely, Algorithms 1 and 3) on the datasets presented in Section 5.3. We tested our implementations on two different types of sets: the sets of white vertices and their complements, which are the sets of black vertices. The distance map to a set of white vertices is used to obtain the dilations of this set whereas the distance

map to its complementary set is used to obtain the erosions of the set (see Property 24, Theorems 21, 22 and 23).

Tables 5.5 and 5.6 and Figures 5.6 and 5.8 show the average execution times of the sequential and of the parallel implementations on 1, 2, 4, 6, and 8 cores for the different datasets. Clearly, as expected from the theoretical analysis of our parallel algorithms in chapter 4 (Section 4.3), the obtained running times are decreasing with the number of cores, whatever the considered dataset.

		Execution times (in seconds)							
		original images				complemented images			
Images size		512 × 512	786 × 540	800 × 600	700 × 1285	512 × 512	786 × 540	800 × 600	700 × 1285
Sequential algorithm		0.026	0.055	0.057	0.060	0.035	0.04	0.044	0.07
Parallel algorithm	1 core	0.025	0.052	0.054	0.063	0.032	0.038	0.042	0.069
	2 cores	0.020	0.040	0.035	0.050	0.024	0.029	0.028	0.056
	4 cores	0.014	0.030	0.026	0.040	0.020	0.020	0.019	0.040
	6 cores	0.011	0.018	0.018	0.030	0.017	0.016	0.012	0.035
	8 cores	0.009	0.013	0.014	0.015	0.012	0.011	0.008	0.018

Table 5.5: Average execution times of the vertex-vertex distance map algorithms for the 2D image datasets.

The speedup values achieved by the parallel implementation are presented in Tables 5.7 and 5.8. We observe that, in all the cases, a good speedup over the sequential implementation is reached. Figures 5.7 and 5.8 plot the obtained speedups when the number of threads varies from 2 to 8. On the image datasets (resp. textured-mesh dataset) the best speedup value, obtained with 8 cores, is equal to 5.5 (resp. 5.33). These values are obtained for the set of black pixels of the images of size 800 × 600 and for the set of black vertices of the textured meshes, respectively.

The previous experiments was designed to asses the gain of the parallel distance map implementation over the sequential one for processing binary 2D images and textured meshes. Let us now move to assess the results obtained using distance maps to compute the results of the morphological operators presented in Definition 13. We recall that the computation of results of the dilations and erosions on graphs for any size parameter λ can

		Execution times (in seconds)	
		original textures	complemented textures
Sequential algorithm		0.015	0.016
Parallel algorithm	1 core	0.014	0.014
	2 cores	0.008	0.009
	4 cores	0.005	0.006
	6 cores	0.004	0.004
	8 cores	0.003	0.003

Table 5.6: Average execution times of the vertex-vertex distance map algorithms for the textured-mesh dataset.

	Speedup							
	original images				complemented images			
Images size	2 cores	4 cores	6 cores	8 cores	2 cores	4 cores	6 cores	8 cores
512 × 512	1.3	1.85	2.36	2.88	1.45	1.75	2.05	2.91
786 × 540	1.37	1.83	3.05	4.23	1.37	2	2.5	3.63
800 × 600	1.62	2.19	3.16	4.07	1.57	2.31	3.66	5.5
700 × 1285	1.2	1.5	2	4	1.25	1.75	2	3.88

Table 5.7: Average speedups of the parallel vertex-vertex distance map algorithm on the 2D image datasets.

be computed with a single iteration by thresholding a distance map instead of applying λ iterations of elementary dilation as one would do by straightforwardly applying the definition. However, such straightforward implementation, was, as far as we know, up to the present work, the only available implementation of these operators.

We compare the execution times of the parallel dilation implementation based on distance map on 8 cores with the sequential version also based on distance map and with the straightforward iterative implementation. Figure 5.9 displays the resulting execution

	Speedup							
	original textures				complemented textures			
	2 cores	4 cores	6 cores	8 cores	2 cores	4 cores	6 cores	8 cores
Speedup	1.87	3	3.75	5	1.77	2.66	4	5.33

Table 5.8: Average speedups of the parallel vertex-vertex distance map algorithm on the textured-mesh dataset.

times. It can be seen that the running times of the implementations based on distance maps remain constant while the size parameter increases whereas, with the straightforward implementation, the execution times increase linearly with the size parameter. In particular, when $\lambda = 2$, the sequential implementation based on distance maps and the straightforward implementation runs in about the same time (0.060 sec) but when λ is equal to 24 the implementation based on distance map is more than 10 times faster than the straightforward implementation. Observe also that, due to the speedup of 5.5 achieved by the parallel implementation for 8 cores machine, the result of the dilation of size 2/2 is obtained 5.5 times faster with the parallel implementation than with the straightforward implementation and more than 55 times faster when the size parameter is equal to 24/2.

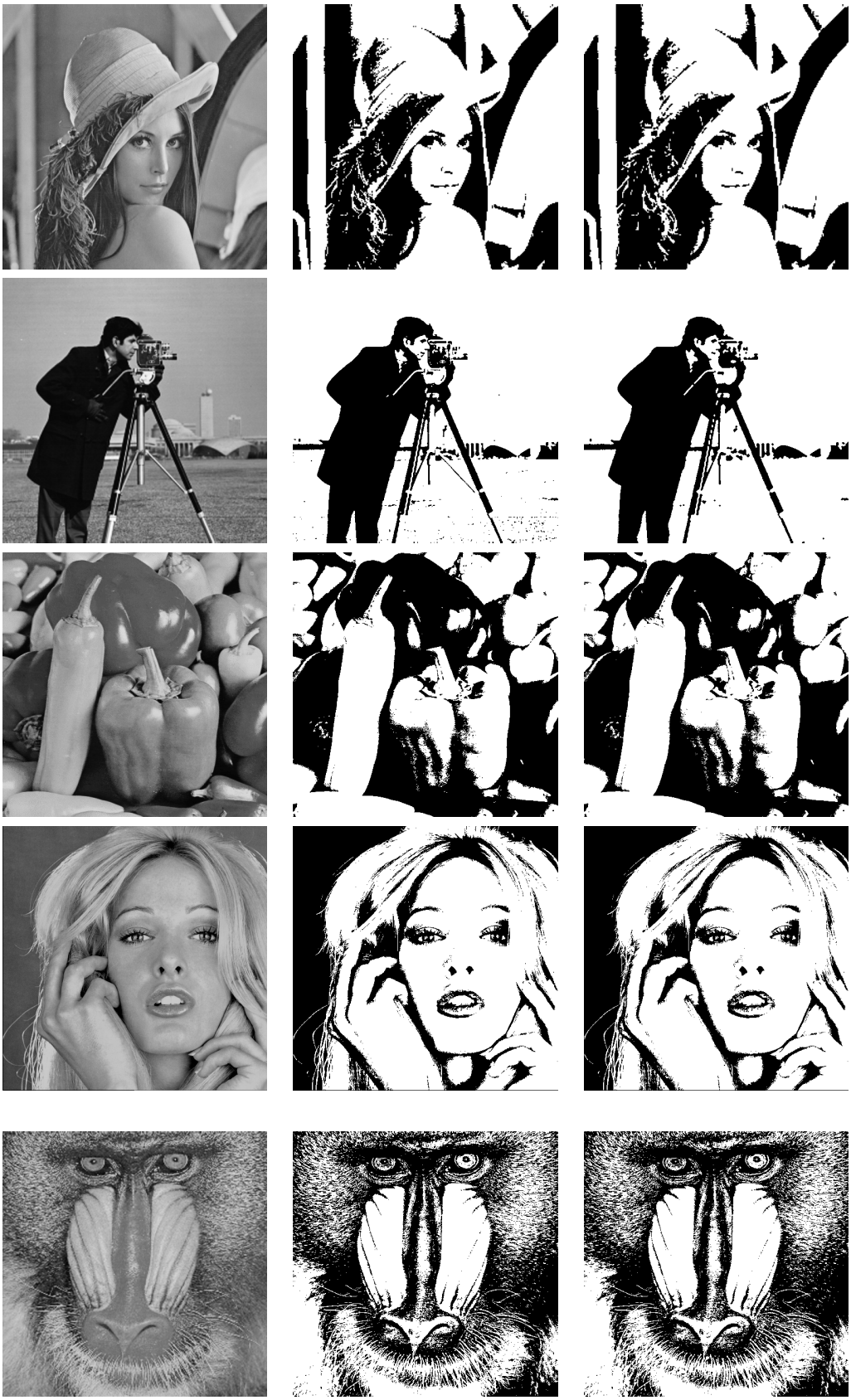
5.6 Conclusion

In this chapter, we presented the target shared memory architecture and we precised the motivation of using multithreaded implementation in our proposed parallel strategy. We also described the various datasets considered for experimental evaluations in this work. Then, we analyzed the cardinalities of the level-sets of the considered distance maps to assess the regularity assumptions of the proposed Theorem 29 on the experimental 2D and 3D textured meshs datasets. Indeed, this theorem asserts that if the unbalancing factor of the graph $\mathbb{G}$ is lower than a small constant and if the considered set of vertices is p-regular, then the time complexity of our algorithms is $O((|\mathbb{G}^\bullet|+|G^\times|)/p+K\log_2 p)$, where K is the number of nonempty level-sets of the distance maps and where p is the number of available processors. Consequently, the analysis carried out confirmed that these assumptions are reasonable for the considered datasets.

After, we assessed the gain of the parallel distance map implementation over the sequential one for processing binary 2D images and textured meshes. The obtained results show clearly, as expected from the theoretical analysis of our algorithms, that the obtained running times are decreasing with the number of cores, whatever the considered datasets. Furthermore, in all presented cases, a good speedup over the sequential implementation is

achieved. In the case of 2D image (resp. the case of 3D textured meshes) dataset, the best speedup value is equal to 5.5 (resp. 5.33) with 8 cores.

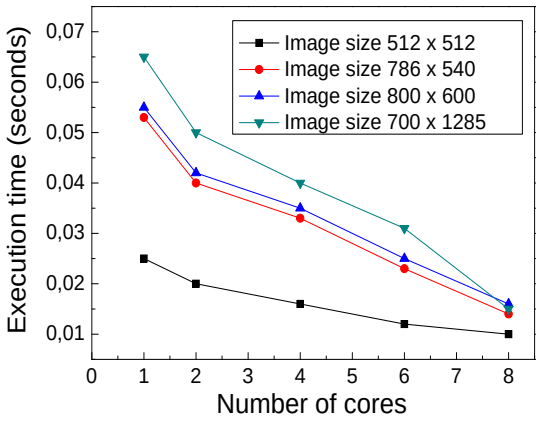
More after, we assessed the results obtained using distance maps to compute the results of the morphological operators presented in Definition 13, we compared the execution times of the parallel dilation implementation based on distance map on 8 cores with the sequential version also based on distance map and with the iterative implementation. The obtained running times of both the sequential and parallel implementations based on distance maps remain constant while the size parameter increases whereas, with the straightforward implementation (iterative), the execution times increase linearly with the size parameter. We reached a faster operator with the parallel implementation of distance map than the other ones. Therefore, implementations of the proposed algorithms on a shared-memory multicore architecture on datasets of 45 images and 6 textured 3-dimensional meshes, showing a reduction of the processing time by a factor up to 55 over the previously available implementations on a 8 core architecture.



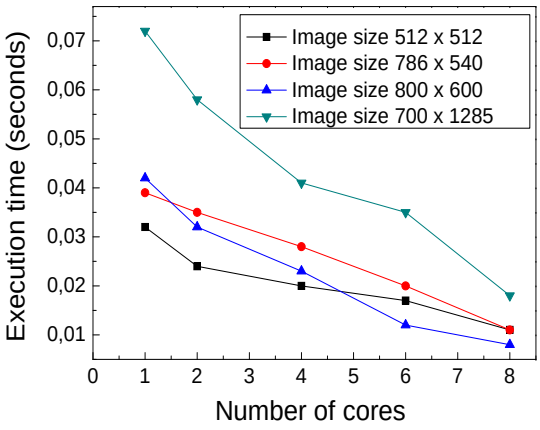
(a) grayscale images (b) results of Otsu binarization (c) results of graph ASF

Figure 5.4: Illustration of Otsu binarization followed by mathematical morphology filtering on graphs from two images of the *Standard* image dataset.

Figure 5.5: Mathematical morphology filtering of a texture defined over a 3 dimensional mesh; (left) two renderings of a 3 dimensional mesh M equipped with a binary texture T; and (right) two renderings of the ASF-filtered version of the texture T on a graph associated with the mesh M.

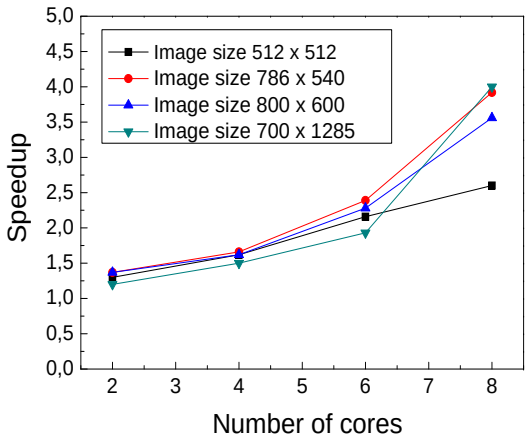


(original images)

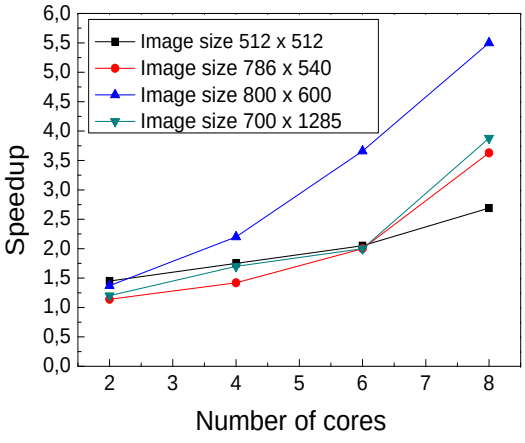


(complemented images)

Figure 5.6: Average execution times, for the 2D image datasets, of the parallel vertex-vertex distance map algorithm, plotted as a function of the number of cores.

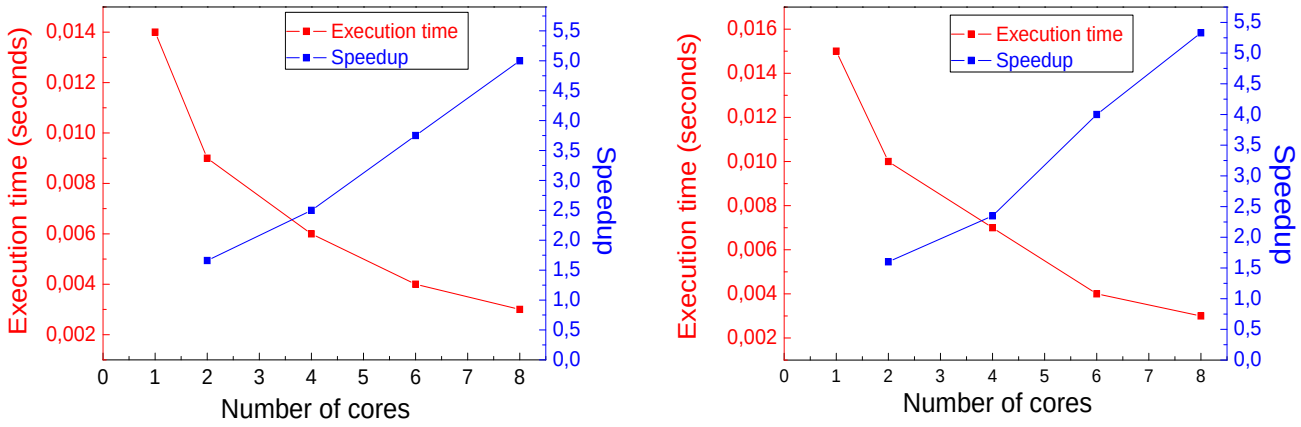


(original images)



(complemented images)

Figure 5.7: Average speedups, on the image datasets, of the parallel vertex-vertex distance map algorithm, plotted as a function of the number of cores.



(original textures) (complemented textures)

Figure 5.8: Average execution times and speedups, on the textured-mesh dataset, of the parallel vertex-vertex distance map algorithm, plotted as a function of the number of cores.

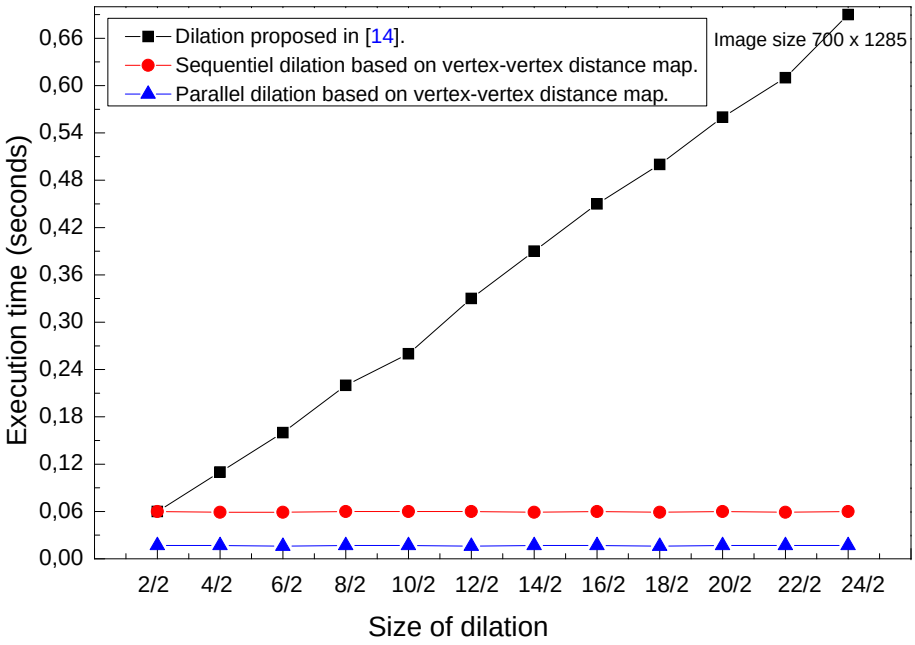


Figure 5.9: Experimental evaluation of iterated dilations $\delta_{\lambda/2}$ when λ is even.

Chapter 6

General Conclusions and Perspectives

Contents

In this thesis, we focused on the framework of mathematical morphology acting on graphs defined in *[J. Cousty* et al*, " Morphological filtering on graphs ", CVIU 2013]*. The main aims of this work is to provide efficient computation and implementation of these (binary) morphological operators on graphs. To reach this goal, our contributions are divided into threefold: i) new three distance maps that lead to original characterizations of the operators of [14]; ii) based on these distance maps, first linear-time sequential algorithms for all dilations/erosions on graphs; and iii) a parallelization strategy leading to fast computation of these distance maps on multicore shared memory architectures.

Distance maps for morphological operators on graphs

In this thesis, we have studied the background notions about mathematical morphology on graphs [14]. This work introduces a framework of mathematical operators where the input and output are both graphs. They investigate four elementary dilations and erosions that map a set of vertices to a set of edges and a set of edges to a set of vertices. We have discussed the problem of the composition of these elementary operators (iterated versions) defined in [14] in order to obtain interesting morphological filters (opening, closings, and associated alternate sequential filers) on graph. These filters are parameterized by a integer value related to a notion of size of the features to be preserved or removed. Larger structures are removed/preserved when higher numbers of iterations of basic operators are

considered. Because of these elementary operators map from a kind of subsets to another kind, they cannot be directly iterated to built such composition. So, there exist two cases (even and odd) of iterated operators that are depending whether a final composition with an operator from a subset of vertices to a subset of edges (resp. from a subset of edges to a subset of vertices) is considered or not. More precisely, eight iterated morphological operators on graphs were introduced in [14]. When λ is even, the inputs and outputs of the operators are subsets of vertices or subsets of edges of graph. When λ is odd, the inputs and the outputs of the operators are distinct: one of them is subset of edges whereas the other one is a subset of vertices.

The naive computation of these iterated operators consists of performing λ iterations of elementary operators. When the complexity of an elementary operator is $O(n)$ where n is the size of the underlying graph. Thus, the time-complexity of the associated algorithm increases with the size parameter. More precisely, for a parameter value of λ the algorithm runs in $O(\lambda.n)$ time, where n is the size of the underlying graph. Therefore, there is a dependence between the size parameter λ and the time-complexity.

The two first operators act on the set of vertices in the case of even value of λ amount to classical morphological operators for which efficient algorithms exist. These algorithms are introduced by L.Vincent in [58] by linear-time algorithms based on thresholding of distance map. Contrary, there exist no efficient algorithms for the six remaining operators.

As first contribution of this thesis, we have introduced three original notions of distance maps on graphs called edge-edge, edge-vertex, and vertex-edge distance maps. Given a set of edges, the edge-edge (resp. edge-vertex) distance map provides for each edge (resp. each vertex) of the graph a geodesic distance to the closest edge in the input set. Given a set of vertices, the vertex-edge distance map provides for each edge a geodesic distance to the closest vertex in the input set. Then, we have proved by characterization theorems that the edge-edge distance map allows us to characterize (by thresholding) the two remaining operators when λ is even, and the edge-vertex and the vertex-edge distance maps allow us to characterize (by thresholding) the four operators defined when λ is odd.
In addition, in standard graph textbooks the length of any path is often considered as the number of edges along the path. whereas, in this thesis, we have considered both the numbers of vertices and of edges along the path. Therefore, we have directly established the link between the length of paths and the iterated operators of [14]. Furthermore, based on this notion of length, we have presented our vertex-vertex distance map in order to obtain the two first operators act on the set of vertices in the case of even value of λ.

Sequential algorithms for morphological operators on graphs

We have proposed efficient sequential algorithms lead to compute vertex-vertex, edge-edge, vertex-edge and edge-vertex distance maps. In particular, these algorithms are derived

from the breadth first search algorithm. Then, we have established the correctness of our algorithms in order to verify if our algorithms really solve our problem and for any valid input they produce the distance maps required. Furthermore, we have analyzed their time complexity which runs in linear time with respect to the size of graph.
This contribution allowed us to efficiently computing the results of all operators of dilations and erosions defined in [14] by thresholding distance maps whatever the size parameter in linear time with respect to the size of the graph, with a single iteration and without any dependence to this size parameter.

Parallel strategy for distance maps on graphs

In order to exploit the parallel multi-core processors architecture, we have proposed a first parallelization strategy leading to a fast computation of the proposed distance maps, hence all morphological operators of [14]. Our parallel strategy handle the regular and non-regular structure of a graph. It is based on dynamic partitioning which depends on the input set and which is iteratively computed during the execution. In our strategy, we have considered the successive level-sets of distance maps where each level set being partitioned and then traversed in parallel. To manage the dynamic partitions, we have proposed two auxiliary functions. The first one is the parallel partition function that compute in parallel p balanced partitions where p is the available processors. The second one is the parallel union function that computes the union of p disjoint sets with p processors.
To state the complexity of our parallel algorithms, we have proposed and proved a theorem that asserts if the unbalancing factor of the graph $\mathbb{G}$ is lower than a small constant and if the considered set of vertices is p-regular, the time complexity of our parallel algorithms is $O((|\mathbb{G}^\bullet| + |G^\times|)/p + K \log_2 p)$, where K is the number of nonempty level-sets of the distance maps and where p is the number of available processors.

Based on our contributions, the parallel implementation on shared memory multi-core/multithreaded architecture is proposed. We have confirmed in practice the regularity assumptions of the proposed theorem on 2D images and 3D textured meshes datasets. Then, we have assessed the gain of the sequential implementation of proposed distance maps over the parallel one for processing binary considered datasets. The obtained results on the target architecture have showed clearly, as expected from the theoretical analysis of our algorithms, that the running times was decreasing with the number of cores, whatever the considered datasets. The sequential implementation based on the proposed sequential distance map algorithm shows a significant improvement over the previous existing implementation of the operators defined in [14] since it runs about ten times faster on the tested images for a dilation size of 24/2. Furthermore, the parallel implementation based on distance maps yields another significant improvement over the sequential one:

a speedup factors up to 5.5 are achieved for 8 cores machine, leading to execution times 55 times smaller than with the previous existing implementation of the operators of [14]. Note that the implementation of the proposed parallel algorithms and its assessment are also completely original.

6.1 Perspective

The issues studied in this thesis open the doors for many avenues of research for the future work. Our contributions is about the field of elementary mathematical morphology operators on graphs for processing binary data (*i.e.* subsets of vertices or of edges of a graph). Even if the definition of these operators can be straightforwardly extended to the case of grayscale data (*i.e.* grayscale functions on the vertices or on the edges), an efficient extension of the proposed algorithms to grayscale case is not straightforward and is an interesting topic for future research.

On the other hand, the use of distance maps in unweighted graphs opens doors towards the investigation of morphological operators on graphs embedded in metric spaces (or more generally on weighted graphs) where the result of an operator depends on the "length" of the edges according to the metric. Such study could lead to significant quality improvements for the results of mathematical morphological operators on graphs.

A last perspective of this thesis, on the computer architecture side, is about the performance optimization based on the use of massive parallelism available in the graphic programming architecture GPU. This later optimization could require new variant around the parallel strategy presented in this thesis.

Bibliography

[1] Virat Agarwal, Fabrizio Petrini, Davide Pasetto, and David A Bader. Scalable graph exploration on multicore processors. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE Computer Society, 2010.

[2] M Usman Akram and Anam Usman. Computer aided system for brain tumor detection and segmentation. In *Computer Networks and Information Technology (ICCNIT), 2011 International Conference on*, pages 299–302. IEEE, 2011.

[3] Felician Alecu. Performance analysis of parallel algorithms. *Journal of Applied Quantitative Methods*, 129, 2007.

[4] Jan Bartovsky. *Architectures matérielles pour filtres morphologiques avec des grandes éléments structurants*. PhD thesis, Université Paris-Est, 2012.

[5] Jan Bartovsky. *Hardware architectures for morphological filters with large structuring elements*. PhD thesis, Paris Est, 2012.

[6] G Birkhoff. Lattice theory. 1995. *American Mathematical Society, Providence*, 1995.

[7] Isabelle Bloch and Alain Bretto. Mathematical morphology on hypergraphs, application to similarity and positive kernel. *Computer vision and image understanding*, 117(4):342–354, 2013.

[8] Isabelle Bloch, Henk Heijmans, and Christian Ronse. Mathematical morphology. In *Handbook of Spatial Logics*, pages 857–944. Springer, 2007.

[9] Gunilla Borgefors. Distance transformations in digital images. *Computer vision, graphics, and image processing*, 34(3):344–371, 1986.

[10] David R Butenhof. *Programming with POSIX threads*. Addison-Wesley Professional, 1997.

[11] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[12] Michel Couprie and Gilles Bertrand. New characterizations of simple points in 2d, 3d, and 4d discrete spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):637–648, 2009.

[13] Jean Cousty, Gilles Bertrand, Michel Couprie, and Laurent Najman. Collapses and watersheds in pseudomanifolds of arbitrary dimension. *Journal of mathematical imaging and vision*, 50(3):261–285, 2014.

[14] Jean Cousty, Laurent Najman, Fabio Dias, and Jean Serra. Morphological filtering on graphs. *CVIU*, 117(4):370–385, 2013.

[15] Jean Cousty, Laurent Najman, and Jean Serra. Some morphological operators in graph spaces. In *Mathematical Morphology and Its Application to Signal and Image Processing*, pages 149–160. Springer, 2009.

[16] Olivier Cuisenaire. Distance transformations: fast algorithms and applications to medical image processing. 1999.

[17] Olaf Delgado-Friedrichs, Vanessa Robins, and Adrian Sheppard. Skeletonization and partitioning of digital images using discrete morse theory. *IEEE transactions on pattern analysis and machine intelligence*, 37(3):654–666, 2015.

[18] Fabio Dias, Jean Cousty, and Laurent Najman. Dimensional operators for mathematical morphology on simplicial complexes. *Pattern Recognition Letters*, 47:111–119, 2014.

[19] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[20] Pedro Felzenszwalb and Daniel Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell University, 2004.

[21] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[22] Lester R Ford Jr. Network flow theory. Technical report, RAND CORP SANTA MONICA CA, 1956.

[23] Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[24] Susan L Graham, Peter B Kessler, and Marshall K Mckusick. Gprof: A call graph execution profiler. In *ACM Sigplan Notices*, volume 17, pages 120–126. ACM, 1982.

[25] Ananth Grama. *Introduction to parallel computing*. Pearson Education, 2003.

[26] Yoshiko Hanada, Hiroyuki Okuno, Mitsuji Muneyasu, and Akira Asano. Application of genetic multi-step search to unsupervised design of morphological filters for noise removal. In *Intelligent Signal Processing and Communication Systems, 2009. ISPACS 2009. International Symposium on*, pages 566–569. IEEE, 2009.

[27] Henk JAM Heijmans. Morphological image operators. *Advances in Electronics and Electron Physics Suppl., Boston: Academic Press,— c1994*, 1, 1994.

[28] Henk JAM Heijmans and Christian Ronse. The algebraic basis of mathematical morphology i. dilations and erosions. *Computer Vision, Graphics, and Image Processing*, 50(3):245–295, 1990.

[29] HJAM Heijmans and Luc Vincent. Graph morphology in image analysis. *OPTICAL ENGINEERING-NEW YORK-MARCEL DEKKER INCORPORATED-*, 34:171–171, 1992.

[30] Muhammad Kaleem, M Sanaullah, M Ayyaz Hussain, M Arfan Jaffar, and Tae-Sun Choi. Segmentation of brain tumor tissue using marker controlled watershed transform method. In *International Multi Topic Conference*, pages 222–227. Springer, 2012.

[31] Henry Kasim, Verdi March, Rita Zhang, and Simon See. Survey on parallel programming model. In *Network and Parallel Computing*, pages 266–275. Springer, 2008.

[32] Ravi B Konuru, Steve W Otto, and Jonathan Walpole. A migratable user-level process package for pvm. *Journal of Parallel and Distributed Computing*, 40(1):81–102, 1997.

[33] Richard E Ladner and Michael J Fischer. Parallel prefix computation. *JACM*, 27(4):831–838, 1980.

[34] Chin Yang Lee. An algorithm for path connections and its applications. *Electronic Computers, IRE Transactions on*, (3):346–365, 1961.

[35] Romain Lerallut, Étienne Decencière, and Fernand Meyer. Image filtering using morphological amoebas. *Image and Vision Computing*, 25(4):395–404, 2007.

[36] N Level Otsu. A threshold selection method from gray-level histogram. *IEEE Transactions on Systems, Man and Cybernetics*, 9(1):62–66, 1979.

[37] Amgad Madkour, Walid G Aref, Faizan Ur Rehman, Mohamed Abdur Rahman, and Saleh Basalamah. A survey of shortest-path algorithms. *arXiv preprint arXiv:1705.02044*, 2017.

[38] Duhu Man, Kenji Uda, Hironobu Ueyama, Yasuaki Ito, and Koji Nakano. Implementations of parallel computation of Euclidean distance map in multicore processors and GPUs. In *ICNC*, pages 120–127, 2010.

[39] Duhu Man, Kenji Uda, Hironobu Ueyama, Yasuaki Ito, and Koji Nakano. Implementations of a parallel algorithm for computing euclidean distance map in multicore processors and gpus. *International journal of networking and computing*, 1(2):260–276, 2011.

[40] Georges Matheron, Georges Matheron, Georges Matheron, and Georges Matheron. Random sets and integral geometry. 1975.

[41] Laurent Mennillo, Jean Cousty, and Laurent Najman. A comparison of some morphological filters for improving ocr performance. In *Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 134–145. Springer International Publishing, 2015.

[42] Fernand Meyer and Jesus Angulo. Micro-viscous morphological operators. *Mathematical Morphology and its Application to Signal and Image Processing (ISMM 2007)*, pages 165–176, 2007.

[43] Fernand Meyer and Serge Beucher. Morphological segmentation. *Journal of visual communication and image representation*, 1(1):21–46, 1990.

[44] Fernand Meyer and Jean Stawiaski. Morphology on graphs and minimum spanning trees. In *ISMM*, pages 161–170. Springer, 2009.

[45] Laurent Najman and Jean Cousty. A graph-based mathematical morphology reader. *Pattern Recognition Letters*, 47:3–17, 2014.

[46] Laurent Najman and Hugues Talbot. *Mathematical morphology*. John Wiley & Sons, 2013.

[47] Tuan Q Pham. Parallel implementation of geodesic distance transform with application in superpixel segmentation. In *DICTA*, pages 1–8. IEEE, 2013.

[48] KR Radhika and SV Sheela. Fundamentals of biometricsâĂŤhand written signature and iris. In *Pattern Recognition, Machine Intelligence and Biometrics*, pages 733–783. Springer, 2011.

[49] Mugdha A Rane. Fast morphological image processing on gpu using cuda. *Pune: Department of Computer Engineering and Information Technology, College of Engineering*, 2013.

[50] Christian Ronse and Jean Serra. Algebraic foundations of morphology. *Mathematical Morphology: from theory to applications*, pages 35–80, 2013.

[51] Azriel Rosenfeld and John L Pfaltz. Distance functions on digital pictures. *Pattern recognition*, 1(1):33–61, 1968.

[52] Toyofumi Saito and Jun-Ichiro Toriwaki. New algorithms for euclidean distance transformation of an n-dimensional digitized picture with applications. *Pattern recognition*, 27(11):1551–1565, 1994.

[53] S Satheesh, RT Santosh Kumar, KVSVR Prasad, and K Jitender Reddy. Skull removal of noisy magnetic resonance brain images using contourlet transform and morphological operations. In *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, volume 4, pages 2627–2631. IEEE, 2011.

[54] Jean Serra. *Image analysis and mathematical morphology, v. 1.* Academic press, 1982.

[55] Shyong Jian Shyu, T Chou, and Tsorng Lin Chia. Distance transformation in parallel. In *Proc. Workshop Combinatorial Math. and Computation Theory*, pages 298–304, 2006.

[56] Olena Tankyevych, Hugues Talbot, Petr Dokládal, and Nicolas Passat. Spatially-variant morpho-hessian filter: Efficient implementation and application. In *Interna-*

tional Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing, pages 137–148. Springer, 2009.

[57] Rafael Verdú-Monedero, Jesús Angulo, and Jean Serra. Spatially-variant anisotropic morphological filters driven by gradient fields. In *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 115–125. Springer, 2009.

[58] Luc Vincent. Graphs and mathematical morphology. *Sig. Proc.*, 16(4):365–388, 1989.

[59] Chao Yin and Hongxia Wang. Developed dijkstra shortest path search algorithm and simulation. In *Computer Design and Applications (ICCDA), 2010 International Conference on*, volume 1, pages V1–116. IEEE, 2010.

Publications

The publications directly related to this thesis are:

- Youkana, I., Cousty, J., Saouli, R., & Akil, M. (2017). Parallelization strategy for elementary morphological operators on graphs: distance-based algorithms and implementation on multicore shared-memory architecture. Journal of Mathematical Imaging and Vision, 1-25.

- Youkana, I., Cousty, J., Saouli, R., & Akil, M. (2016, April). Parallelization strategy for elementary morphological operators on graphs. In International Conference on Discrete Geometry for Computer Imagery (pp. 311-322). Springer International Publishing.

- Youkana, I., Saouli, R., Cousty, J., & Akil, M. (2015, January). Morphological operators on graph based on geodesic distance map. In Computer Vision and Image Analysis Applications (ICCVIA), 2015 International Conference on (pp. 1-6). IEEE.