Pontifícia Universidade Católica
do Rio de Janeiro

Ian Hodara Herszterg

2D Phase Unwrapping via Minimum Spanning Forest with Balance Constraints

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–Graduação em Informática of the Departamento de Informática, PUC–Rio as partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor : Prof. Thibaut Victor Gaston Vidal
Co–advisor: Prof. Marcus Vinícius Soledade Poggi de Aragão

Rio de Janeiro
August 2015

Pontifícia Universidade Católica
do Rio de Janeiro

Ian Hodara Herszterg

2D Phase Unwrapping via Minimum Spanning Forest with Balance Constraints

Dissertation presented to the Programa de Pós–Graduação em Informática, of the Departamento de Informática do Centro Técnico Científico da PUC–Rio, as partial fulfillment of the requirements for the degree of Mestre.

Prof. Thibaut Victor Gaston Vidal
Advisor
Departamento de Informática — PUC–Rio

Prof. Marcus Vinícius Soledade Poggi de Aragão
Co–advisor
Departamento de Informática — PUC–Rio

Prof. Yuri Abitbol De Menezes Frota
UFF

Prof. Rafael Martinelli Pinto
Trieda

Prof. José Eugenio Leal
Coordinator of the Centro Técnico Científico — PUC–Rio

Rio de Janeiro, August 31st, 2015

Ian Hodara Herszterg

Obtained a bachelor's degree in Computer Engineering from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio, Rio de Janeiro, Brazil) in December 2012.

Bibliographic data

Herszterg, Ian Hodara

2D Phase Unwrapping via Minimum Spanning Forest with Balance Constraints / Ian Hodara Herszterg; advisor: Thibaut Victor Gaston Vidal; co–advisor: Marcus Vinícius Soledade Poggi de Aragão. — 2015.

98 f. : il. (color.); 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Departamento de Informática, 2015.

Inclui bibliografia.

1. Informática – Teses. 2. phase unwrapping. 3. processamento de sinais. 4. heurísticas. 5. otimização combinatória. 6. branch-and-cut. I. Vidal, Thibaut. II. Poggi de Aragão, Marcus Vinícius. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

Acknowledgement

First, I would like to thank my advisors Thibaut Vidal and Marcus Poggi for their incredible enthusiasm and constant support. During my dissertation, they have always been a source of knowledge and inspiration. I could not ask for greater advisors.

I would also like to thank CAPES, for supporting me through the entire duration of this degree via a scholarship.

Last but not least, I would like to thank my family for their unconditional love and support.

Abstract

Herszterg, Ian Hodara; Vidal, Thibaut (Advisor); Poggi de Aragão, Marcus Vinícius (Co-Advisor). **2D Phase Unwrapping via Minimum Spanning Forest with Balance Constraints**. Rio de Janeiro, 2015. 98p. MSc. Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The development and application of techniques in coherent signal processing have greatly increased over the past several years. Synthetic aperture radar, acoustic imaging, magnetic resonance, X-Ray crystallography and seismic processing are just a few examples in which coherent processing is required, resulting in the need for accurate and efficient phase unwrapping methods. The phase unwrapping problem consists in recovering a continuous phase signal from an originally wrapped phase data between the $]-\pi,\pi]$ interval. This dissertation proposes a new model for the L^0-norm 2D phase unwrapping problem, in which the singularities of the wrapped phase image are associated to a graph where the vertices have different polarities (+1 and -1). The objective is to find a minimum cost balanced spanning forest where the sum of polarities is equal to zero in each tree. A set of primal and dual heuristics, a branch-and-cut algorithm and a hybrid metaheuristic method are proposed to address the problem, leading to an efficient approach for L^0-norm 2DPU, previously viewed as highly desirable but intractable. A set of experimental results illustrates the effectiveness of the proposed algorithm, and its competitiveness with state-of-the-art algorithms.

Keywords

phase unwrapping; signal processing; heuristics; combinatorial optimization; branch-and-cut; mathematical programming; algorithms; graphs.

Resumo

Herszterg, Ian Hodara; Vidal, Thibaut; Poggi de Aragão, Marcus Vinícius. **Phase Unwrapping 2D via Floresta Geradora Mínima com Restrições de Balanceamento**. Rio de Janeiro, 2015. 98p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O desenvolvimento e a aplicação de técnicas em processamento de sinais coerentes obtiveram um avanço significativo nas últimas décadas. A decodificação de imagens de radar de abertura sintética, imagens acústicas, de ressonância magnética, e de processamento sísmico são apenas alguns exemplos em que o processamento de sinais coerentes é utilizado, levando à necessidade do uso de métodos precisos e eficientes para a técnica de phase unwrapping. O problema de *phase unwrapping* consiste em recuperar um sinal de fase contínuo a partir de um conjunto de dados de fase originalmente restritos ao intervalo de $]-\pi,\pi]$. Este trabalho propõe um novo modelo para a norma L^0 do problema de *Phase Unwrapping* 2D (2DPU), onde as inconsistências da imagem de fase *wrapped* são associadas a um grafo no qual vértices possuem diferentes polaridades (+1 e -1). Procura-se obter uma floresta geradora de custo mínimo, onde o somatório das polaridades dos vértices é igual a zero para cada árvore. Um conjunto de heurísticas primais e duais, um método exato de *branch-and-cut* e uma meta-heurística híbrida são propostos para tratar o problema, resultando em uma abordagem eficiente para a norma L^0 do problema 2DPU anteriormente vista como altamente desejável, porém intratável.

Palavras–chave

phase unwrapping; processamento de sinais; heurísticas; otimização combinatória; branch-and-cut; programação matemática; algoritmos; grafos.

Contents

List of tables

1
Introduction

Two-dimensional phase unwrapping (2DPU) is the process of reconstructing a set of continuous phase signals from a 2D array of phase values wrapped around a 2π interval. This procedure is a key data-processing step for many coherent signal processing applications. Synthetic aperture radar, acoustic imaging, magnetic resonance, X-Ray crystallography and seismic processing are just a few examples in which coherent processing is required and the phase values extracted from the acquired signals are of great interest.

In complex signals, the phase information defines the position of a sample on a waveform cycle, measured as an angle in degrees or radians. Moreover, phase values can also be expressed as the relative displacement between waves having the same frequency. In the context of radar interferometry [1], multiple coherent radar images of a surface are combined to form interferograms maps. The reflected radiations from the image area are used to generate maps of the surface's topology, where different elevations are described by the relative displacements between the reflected waves (i.e. the phase values). However, since the phase values are extracted by trigonometric operators, the acquired phase signal is wrapped around a 2π domain. The wrapped signal must be unwrapped into a full range of real numbers, thus reconstructing the continuous signal and retrieving the original information. Although there is an extensive number of methods already published in the literature, phase unwrapping is still considered to be an active research topic [2]. How to deal with inconsistencies caused by noise, under-sampled signals and other natural discontinuities still poses a challenge for many state-of-the-art algorithms. Phase unwrapping for practical real-world applications is considered to be one of the most challenging tasks in digital signal processing.

This dissertation proposes a new approach for the phase unwrapping problem via an optimization model, presenting a new set of mathematical formulations, heuristic methods and an exact branch-and-cut algorithm. We express the desired goal as an optimization objective and develop efficient methods known from the field of optimization and operational research to produce satisfactory solutions. The contributions of this research includes:

– A new approach for the minimization of the L^0-norm of the 2DPU problem, considered as highly desirable but intractable. We seek a

minimum cost spanning forest where every tree spans a balanced number of inconsistencies in the wrapped phase data.

- Efficient exact and heuristic methods to address the problem. In particular, we introduce a dual ascent heuristic as well as a branch-and-cut algorithm which can solve problems to optimality with up to 128 residues in reasonable computational time.
- A hybrid metaheuristic designed to address larger instances.
- Computational experiments demonstrating that the new formulation provides a very satisfactory answer to phrase unwrapping problem, for applications in radar intereferometry and magnetic resonance imaging.

This thesis is structured as follows: Chapter 2 introduces the theoretical aspects of the phase unwrapping problem. Chapter 3 presents the main approaches and state-of-the-art methods proposed in the literature. Chapter 4 presents the proposed MSFBC approach, heuristic and exact methods, with emphasis on its mathematical formulations and a proof of its complexity. Chapter 5 presents the experimental results and a set of benchmarks against state-of-the-art algorithms. Finally, Chapter 6 concludes the dissertation and sets directions for future work.

2
Introduction to Phase Unwrapping

2.1
The Importance of Phase

The major importance of the phase information in signal processing can be highlighted by a simple experiment, described by Oppenheim and Lim in [3]. In this experiment, the 2D discrete Fourier transform (DFT) is used to decompose two images, into their sine and cosine components, separating the resulting complex spectrum into a magnitude and a phase value for each pixel. The output of the transformation represents the image in the frequency domain, while the original image is the spatial domain equivalent. The experiment consisted in obtaining the DFT of two different images, A and B, and recombine their magnitude and phase spectra ($magnitude_A$ with $phase_B$ and $magnitude_B$ with $phase_A$) to synthesize new images and better understand the importance of each signal property extracted by the Fourier transform.

Figures 2.1(a) and 2.1(b) shows two digital grayscale photographs of the Eiffel Tower and the Statue of Liberty, respectively. It is possible to obtain the magnitude and phase spectra from the DFT $H(\mathrm{x})$ by applying the following operations on the real and imaginary parts ($\Re_H$ and $\Im_H$):

$$magnitude_H = |\Re_H| \tag{2.1.1}$$

$$phase_H = \arctan(\Im_H, \Re_H), \tag{2.1.2}$$

where arctan is the four-quadrant arc tangent operator that wraps the phase values to the interval of $]-\pi, \pi]$. Observing the resulting magnitude spectra in Figures 2.2(a) and 2.2(b), we can easily identify that each of them shows the distribution of magnitudes in the spatial frequency of its corresponding image. Figure 2.1(a) has a much wider bright area in the center of the image than Figure 2.1(b), which is translated by the larger concentration of higher valued (brighter) pixels in the center of its magnitude spectrum. On the other hand, both phase spectra showed in Figures 2.3(a) and 2.3(b) do not have any direct visual correlation with the information contained in the original images.

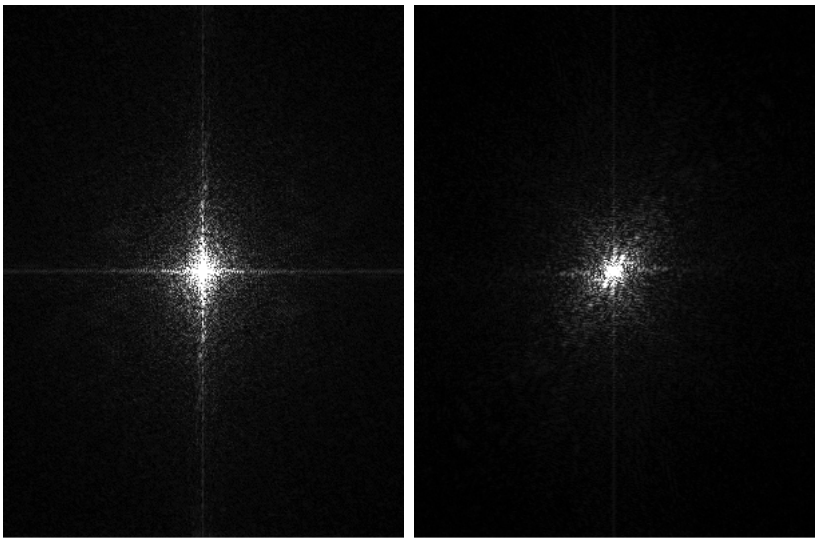
Phase values by themselves do not have any sort of visual correlation, as they only indicate the relative position of each two-dimensional complex

2.1(a): 2.1(b):

Figure 2.1: *300x400 pixel digital gray scale images. (a) The Eiffel Tower (b) The Statue of Liberty. Both images were courtesy of FreeDigitalPhotos.net*

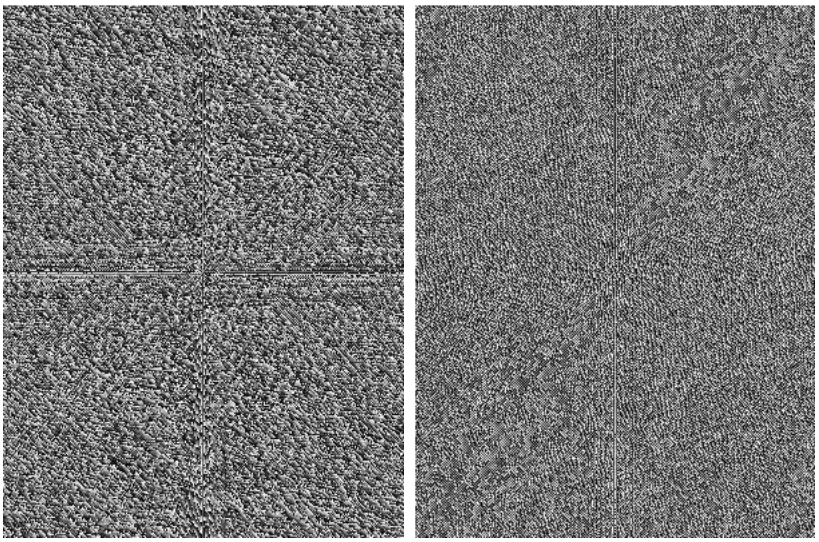
exponential in the 2D Fourrier decomposition. The values stored in each pixel of the 2D phase spectrum contains essential information about the pixel topology of its spatial domain image. This concept can be better understood by the second part of the experiment, which consists in using the DFT inversion technique to reconstruct and synthesise new images from the two new complex signals created by swapping the magnitude and phase spectra from figures 2.1(a) and 2.1(b).

Figures 2.4(a) and 2.4(b) display the output of this experiment, showing that while swapped magnitude spectra simply brightens, darkens or creates cloudy areas in the recombined image, the information contained in the phase spectra retains most of the essential characterization of its original photograph. The overall interpretability of each image is still maintained even when its magnitude spectrum is disturbed. This simple experiment highlights the importance of phase in coherent signal processing and the extreme care that the phase information must be handled in order to develop high quality methods.



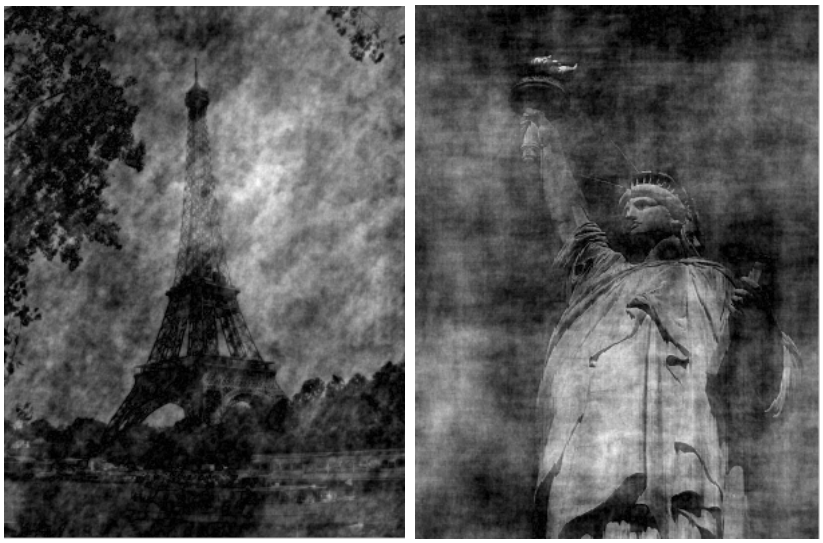
2.2(a): 2.2(b):

Figure 2.2: *Log-scaled 2D DFT magnitude spectra. (a) magnitude spectrum of the Eiffel Tower image (b) magnitude spectrum of Statue of Liberty image.*



2.3(a): 2.3(b):

Figure 2.3: *2D DFT phase spectra. (a) phase spectrum of the Eiffel Tower image (b) phase spectrum of Statue of Liberty image.*



2.4(a): 2.4(b):

Figure 2.4: *New images reconstructed from the inverse DFT transfrom with swapped spectra. (a) The resulting image of the Statue of Liberty magnitude with the Eiffel Tower phase spectra. (b) The resulting image of the Eiffel Tower magnitude with the Statue of Liberty phase spectra.*

2.2 Why Phase Unwrapping?

While the amplitude of phase information can take any real value, it is wrapped to a 2π interval with a $]-\pi,\pi]$ domain by the four-quadrant arctangent operator. Since none of the trigonometric operators are injective functions (one-to-one), their domain must be restricted so they can have inverse functions. Inverse functions will only evaluate to a single value (i.e. the principal value).

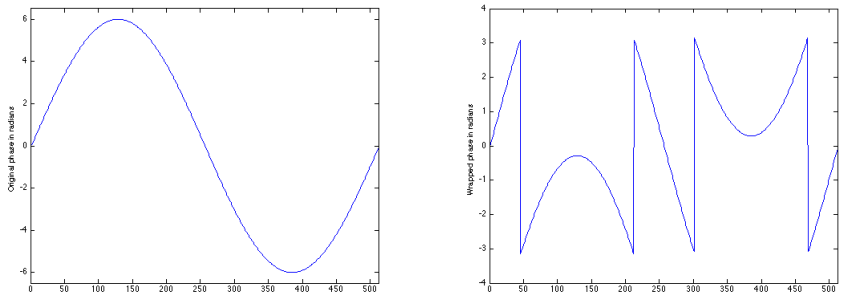
In mathematical terms, the principal value of phase is also known as its *modulo*-2π, which is the result of wrapping the phase around its principal interval. The wrapping of an unknown phase function ϕ(t) to the $]-\pi,\pi]$ interval can be expressed as:

$$W(\phi(t)) = \phi(t) + 2\pi k(t), \tag{2.2.1}$$

or, more generally as:

$$\psi(t) = W(\phi(t)), \tag{2.2.2}$$

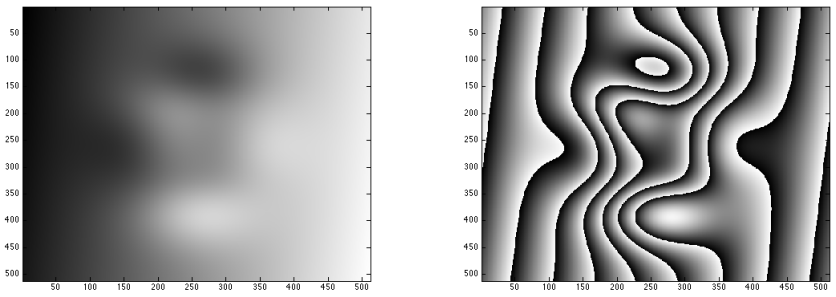
where k(t) is an integer function that wraps all phase values ϕ(t) around $]-\pi,\pi]$, $W()$ is the wrapping operator and ψ(t) is the wrapped output. Figures 2.5 and 2.6 shows the wrapping effect on a continuous 1D signal and a 2D phase image, respectively.



2.5(a): 2.5(b):

Figure 2.5: *Wrapping effect on a 1D continuous phase signal. (a) Continuous signal (b) Wrapped signal.*

The artificial 2π jumps presented in the wrapped phase signal ψ(t) must be removed in order to reconstruct the original continuous signal, ϕ(t). This process is called *phase unwrapping*, whose goal is to remove the 2π-multiple ambiguity.



2.6(a): 2.6(b):

Figure 2.6: *Wrapping effect on a 2D phase image. (a) Absolute phase image (b) Wrapped phase image.*

Itoh et al. showed in [4] that for an efficient phase unwrapping, the absolute phase difference between any two adjacent samples in the continuous phase signal cannot exceed the value of π (known as the Itoh condition). If we consider a sequence ϕ_n of absolute phase values stored in a discretized array, then the linear difference Δ between adjacent samples can be defined as:

$$\Delta_{\phi_n} = \phi_n - \phi_{n-1}, \tag{2.2.3}$$

leading us to:

$$\begin{aligned}\sum_{n=1}^{m} \Delta\phi_n &= (\phi_m - \phi_{m-1}) + (\phi_{m-1} - \phi_{m-2}) + \cdots + (\phi_2 - \phi_1) \\ &= (\phi_m - \phi_1) + (\phi_{m-1} - \phi_{m-1}) + \cdots + (\phi_2 - \phi_2) \\ &= (\phi_m - \phi_1).\end{aligned} \tag{2.2.4}$$

Let $\Delta W(\phi_n)$ be the linear difference between adjacent wrapped samples. From (2.2.1) and (2.2.3), we get that:

$$\begin{aligned}\Delta W(\phi_n) &= W(\phi_n) - W(\phi_{n-1}) \\ &= (\phi_n + 2\pi k_n) - (\phi_{n-1} + 2\pi k_{n-1}) \\ &= \phi_n - \phi_{n-1} - 2\pi(k_n - k_{n-1}).\end{aligned} \tag{2.2.5}$$

Now, if we apply wrapping operator $W()$ over $\Delta W(\phi_n)$, we obtain the wrapped difference between wrapped phase samples, described as:

$$\begin{aligned}W(\Delta W(\phi_n)) &= \Delta_{\phi_n} - 2\pi(k_n - k_{n-1}) - 2\pi k' \\ W(\Delta W(\phi_n)) &= \Delta_{\phi_n} - 2\pi(k_n - k_{n-1} - k'),\end{aligned} \tag{2.2.6}$$

where k' is a proper 2π multiple that brings the right-hand size of the equation to its principal interval and assures that it does not violate the Itoh condition. Since Δ_{ϕ_n} is at most equal to π in its absolute value, we can immediately reach out to the conclusion that k' must be equal to $(k_n - k_{n-1})$ in order to keep both sides of the equation in the appropriate domain. Hence, we have:

$$W(\Delta W(\phi_n)) = \Delta_{\phi_n} \tag{2.2.7}$$

Substituting (2.2.7) in (2.2.4) and isolating the term ϕ_m, we finally obtain:

$$\phi_m = \sum_{n=i}^{m} W(\Delta W(\phi_n)) + \phi_i, \tag{2.2.8}$$

which gives us a mathematical formulation for a procedure to obtain the absolute phase ϕ_m from the wrapped phase values alongside any path connecting sample i to another sample m, where the absolute phase value ϕ_i is known. The term *integration path* is used to define any valid unwrapping path described by Equation (2.2.8). This computation is illustrated in the next section.

2.3 The Itoh Algorithm for One Dimensional Phase Unwrapping

A simple one dimensional phase unwrapping procedure based on Itoh's analysis is presented in Algorithm 1. The procedure starts by iterating from a sample in which the absolute phase value is assumed to be known or arbitrarily fixed ($\phi(1) = \psi(1)$), then verifies if the wrapped phase difference of its lefthand immediate adjacent sample does violate the $]-\pi, \pi]$ interval. If so, a 2π increment is added. These steps are iteratively executed until all samples are evaluated. Figure 2.7 shows the step-by-step algorithm on a wrapped signal and how it reconstructs the original continuous signal.

2.4 The Effects of Noise and Under Sampling

Although this simple algorithm succeeds in unwrapping an ideal input, the effects of noise and under sampling can have a severe impact on the solution's quality. In order to be processed in a digital environment, the continuous analog signal must be discretized and converted into a sequence of samples. The Nyquist-Shannon sampling theorem, as described in [5], states that a continuous signal can be completely reconstructed from a sampling made at a frequency greater or equal than the twice of its higher frequency component w. This means that the minimum sampling frequency needed for

a faithful discretization must be equal to $2w$, for which the difference between every adjacent samples has a value of π, at most. This concept can be directly related with the Itoh condition, which guarantees the correctness of any phase unwrapping method in this ideal scenario.

Algorithm 1: Itoh's method for 1D Phase Unwrapping

input : Wrapped phase values, $\psi(n)$
output: Unwrapped phase values $\phi(n)$

Initialisation: $\phi(1) = \psi(1)$;
for $i \leftarrow 2$ **to** N **do**
  $\Delta_\psi \leftarrow \psi(i) - \psi(i-1)$;
  if $\Delta_\psi \leq -\pi$ **then**
    $\Delta_\psi \leftarrow \Delta_\psi + 2\pi$;
  else if $\Delta_\psi > \pi$ **then**
    $\Delta_\psi \leftarrow \Delta_\psi - 2\pi$;
  $\phi(i) \leftarrow \phi(i-1) + \Delta_\psi$;
end

Since the difference between adjacent samples of the wrapped phase signal must be iteratively computed in order to perform the unwrapping process, any *fake wrap* caused by a slight perturbation in the data (i.e. introduced by noise) or by a poor discretization (under sampling) can generate an unnecessary 2π increment, which will be propagated through the subsequent values. It is unrealistic to assume that the Itoh condition will be naturally preserved between all adjacent samples, which can mislead even the most sophisticated phase unwrapping methods. Figures 2.8 and 2.9 show the unwrapping process over noisy and under-sampled data, respectively, demonstrating how the Itoh's algorithm fails to reconstruct the original continuous signal.

Phase unwrapping problems often comes from complex applications dealing with rich geometries and signal acquisition methods that are highly susceptible to noise. The abrupt changes in the phase values caused by these events cannot be distinguished from the natural discontinuities, and thus the unwrapping is most likely to fail, even in the one-dimensional scenario.

As seen in Section 2.3, Itoh's unwrapping procedure can be executed for any continuous integration path, as long as no artificial discontinuities are presented along its way. Assuming that the wrapped phase gradients are known along with the absolute phase value at some initial point, every integration path P can constitute a discrete unwrapping path. In this scenario, adjacent samples are no longer limited to a single dimension. We can express this concept mathematically by:

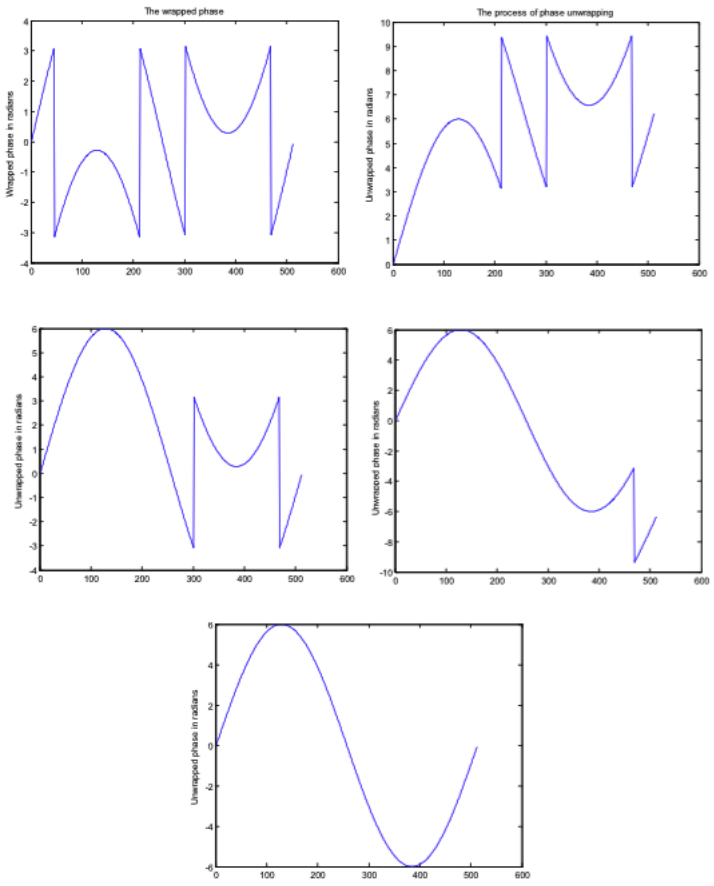


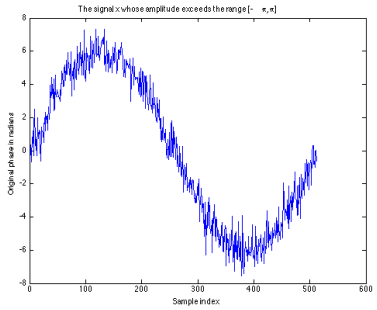
Figure 2.7: *Unwrapping process by Itoh's method for 1D Phase Unwrapping, recovering the original continuous signal from the wrapped phase samples.*

$$\phi_n = \sum_{i \in \nabla(P)} W(\Delta W(\phi_{P(i)})) + \phi_0, \tag{2.4.1}$$

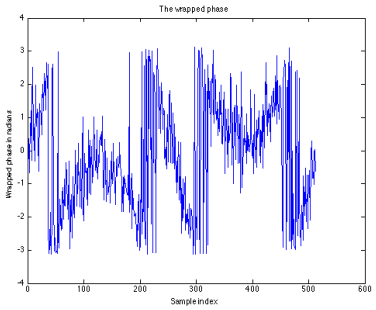
where $\nabla(P)$ represents consecutive samples within path P connecting samples ϕ_0 to ϕ_n.

In two or more dimensions, the effects of noise and under-sampling introduces an even more complex scenario for the unwrapping procedure. While there is no way to overcome these singularities in one dimension, the N-dimensional environment offers the possibility of the appropriate selection of integration paths. These paths can be tailored to succeed the unwrapping process by avoiding damaged regions, as shown in the following sections.

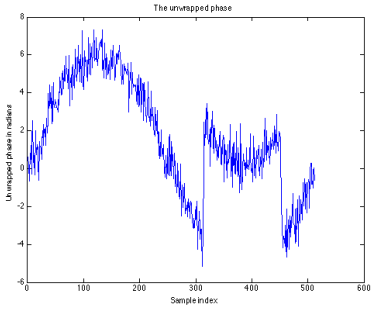
Sections 2.5 and 2.6 introduces the path dependency concept and residue theory, which are indispensable for the 2D path-following phase unwrapping methods discussed in Section 3.1. Section 2.7 presents a simple 2D unwrapping procedure and its results on artificial examples, showing the outcome in the presence of singularities in the data.



2.8(a):

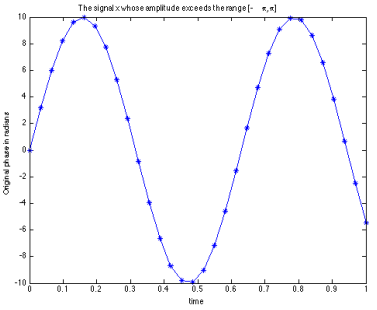


2.8(b):

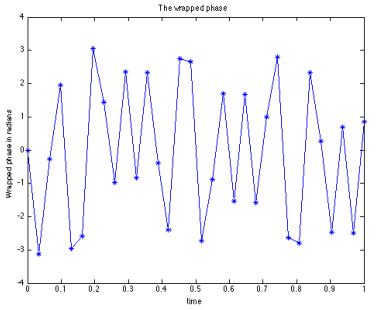


2.8(c):

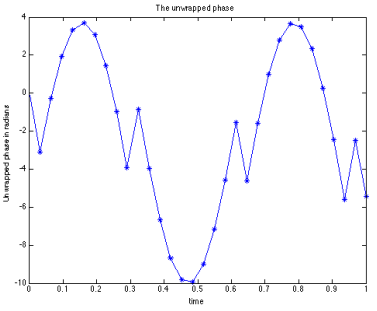
Figure 2.8: *Unwrapping process over noisy data. (a) The continuous phase signal with noise. (b) The wrapped noisy phase signal. (c) The unwrapping obtained by Itoh's algorithm.*



2.9(a):



2.9(b):



2.9(c):

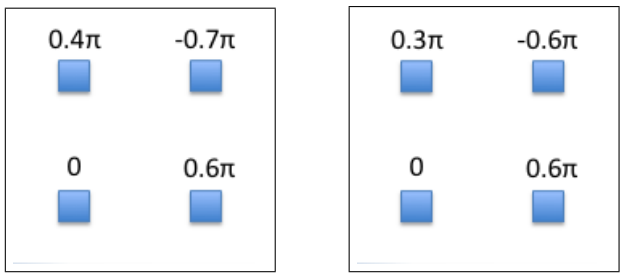
Figure 2.9: *Unwrapping process over under-sampled data. (a) The continuous phase signal under-sampled. (b) The wrapped under-sampled phase signal. (c) The unwrapping obtained by Itoh's algorithm.*

2.5 Path Dependency

Itoh's method can be used to compute the absolute phase values on all samples in a multidimensional input, as shown by Equation 2.4.1. When no singularities are found, the choice of a discrete integration path is arbitrary and therefore, any potential path linking two distinct samples should produce the same unwrapping output.

Figure 2.10(a) presents a 2D wrapped phase data with no sign of singularities. The unwrapping procedure is done by fixing an initial sample and iterating over a clockwise or a counterclockwise path, as shown by Figure 2.11. There is no distinction between the unwrapped solution values computed by both integration paths, validating Equation 3.0.1 and its assumption that the selection of integration paths is arbitrary when no singularities are present.

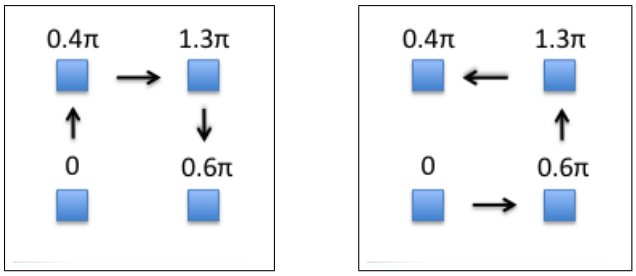
Figure 2.10(b), on the other hand, is an extract of a 2D under-sampled signal. The unwrapping procedure over different integration paths produces vastly different solutions, as shown by Figure 2.12. The singularity causes the unwrapping procedure to be *path dependent*, as they produce different outputs while iterating over distinct paths. This event can be easily identified by evaluating the different results produced by a path and its inverted direction counterpart. The term *parallel paths* is often used in the literature to define two or more paths that share the same start and end samples, but produces different unwrapping results.



2.10(a): 2.10(b):

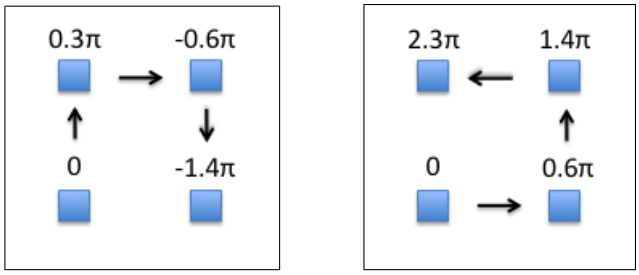
Figure 2.10: *2D Wrapped phase data examples. (a) The example A does not contain singularities. (b) The example B is under-sampled.*

Ghiglia et al. first introduced the path dependency phenomenon in [6]. While most of the singularities produced by noise and/or under-sampling can entirely corrupt the solution, the authors observed that these inconsistencies were only restricted to certain regions in the 2D phase image. Section 2.6 introduces the residue theorem applied for the phase unwrapping problem and how to correctly detect any singularity or discontinuity present in the data.



2.11(a): 2.11(b):

Figure 2.11: *Unwrapped solutions from Example A in Figure 2.10(a) showing no path dependency. (a) Unwrapped solution obtained by the clockwise integration path. (b) Unwrapped solution obtained by the counterclockwise integration path.*



2.12(a): 2.12(b):

Figure 2.12: *Unwrapped values from Example B in Figure 2.10(b), showing the occurrence of path dependency. (a) Unwrapped values obtained by the clockwise integration path. (b) Unwrapped values obtained by the counterclockwise integration path.*

2.6 Residue Theory

The term "residue" was first introduced by *Goldstein et. al.* [1] as an analogy between residues found in complex signals and the inconsistencies presented in the phase unwrapping problem.

From the theory of complex signals, a residue is a complex value proportional to the number of singularities encircled by a closed integral path over the complex function. The residue theorem for complex signals (a.k.a. Cauchy's residue theorem, defined in [7]) states that any closed integral can be written as:

$$\oint f(z)dz = 2\pi j \times \sum r_i, \tag{2.6.1}$$

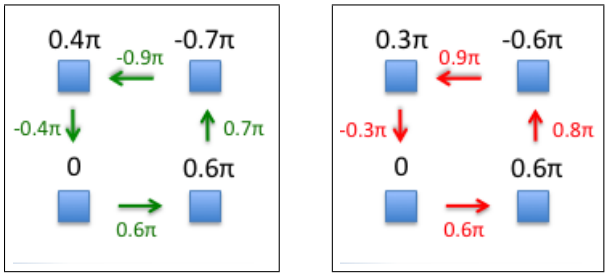
where $\sum r_i$ represents the sum of all residues r_i found in the closed path and *f(z)* is the complex function over the complex variable *z*.

In order to consider the residue theorem in the two-dimensional phase unwrapping problem, Equation 2.6.1 can be re-written as:

$$\sum_{i\in\nabla(P')} W[\Delta W(\phi_{P'(i)})] = 2\pi \times \sum R_i, \tag{2.6.2}$$

where P' is a closed path defined over the 2D wrapped phase samples. *Ghiglia et. al.* showed in [2] that each singularity can only cause a $\pm 2\pi$ unwrapping error to its subsequent samples around an integration path, which then evaluates to an error multiple of 2π in the path's last sample. Hence, in the phase unwrapping problem, each singularity R_i represents a ± 1 charged residue.

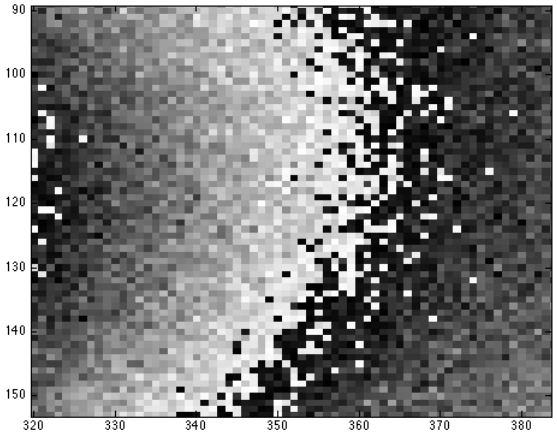
Based on this concept, Ghiglia *et al.* also proposed the following strategy to identify the location of all residues in the 2D wrapped phase image. By applying the procedure from Equation 2.6.2 over the wrapped phase values around every elementary 2x2 loop, any residue's location and charge can be precisely obtained whenever the sum of the wrapped phase gradients is not equal to zero. This simple strategy succeeds to pinpoint every potential residue and inspire techniques to solve the path dependency problem. Figure 2.13 shows the elementary loop on the previous path dependency examples, enlightening the +1 charged residue located in example B from Figure 2.10(b). Figure 2.13 shows a small extract from the residues found in a noisy wrapped phase data.



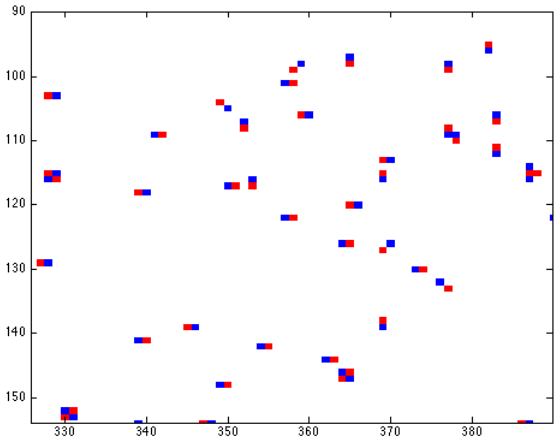
2.13(a): 2.13(b):

Figure 2.13: *Closed loop paths and the wrapped phase gradients on examples A and B from Figures 2.10(a) and 2.10(b). (a) The sum of the wrapped phase gradients sums to zero, indicating that no residue is present. (b) The sum of the wrapped phase gradients sums up to 2π, identifying the +1 charged residue.*

While the residues of a 2D wrapped phase image indicates the existence of inconsistencies, not every residue represents a singularity produced by noise and/or under-sampling. Phase discontinuities may be naturally present in many phase unwrapping applications, where the absolute phase values can abruptly increase or decrease over adjacent samples. The different elevations and deformities presented in a terrain's surface is one example in which the natural phase discontinuities will most likely be interpreted as residues. As



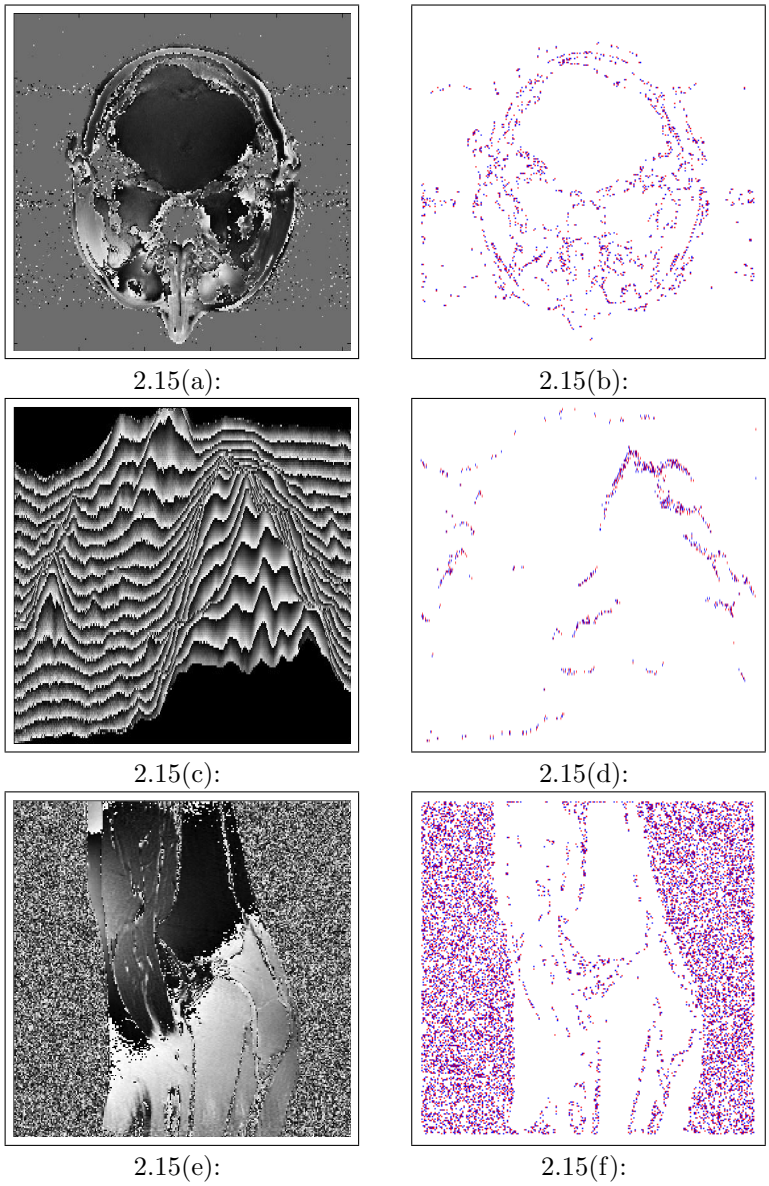
2.14(a):



2.14(b):

Figure 2.14: *Residues detected over a wrapped phase image corrupted by noise. (a) The wrapped phase image with noise. (b) A small extract of the image showing the +1 (blue) and -1 (red) charged residues.*

shown in Figure 2.15, the topology of residues reflects many of the natural structural delimitations present in the subjects of study. Other sources of natural discontinuities are discussed in [8].



2.15(a): 2.15(b): 2.15(c): 2.15(d): 2.15(e): 2.15(f):

Figure 2.15: *The residues detected over several wrapped phase examples mentioned in [2]. Clearly, the topology of residues are suggesting contours for different well-defined regions in each subject of study, apart from additional noise and under-sampling. (a) The wrapped phase image of an magnetic resonance image head experiment (b) The residues detected over Figure 2.15(a). (c) The wrapped phase image of an bird's eye surface view of a terrain's elevation. (d) The residues detected over Figure 2.15(c). The wrapped phase image of an magnetic resonance image knee experiment. (f) The residues detected over Figure 2.15(e).*

2.7 2D Phase Unwrapping

The choice of the appropriate integration paths in a two-dimensional phase unwrapping problem with no residues can be done via a straightforward modification of the one-dimensional algorithm. Since we need to explore all samples in a continuous way, one simple method of integration is to iterate over all rows and columns of the image, as shown by Algorithm 2.

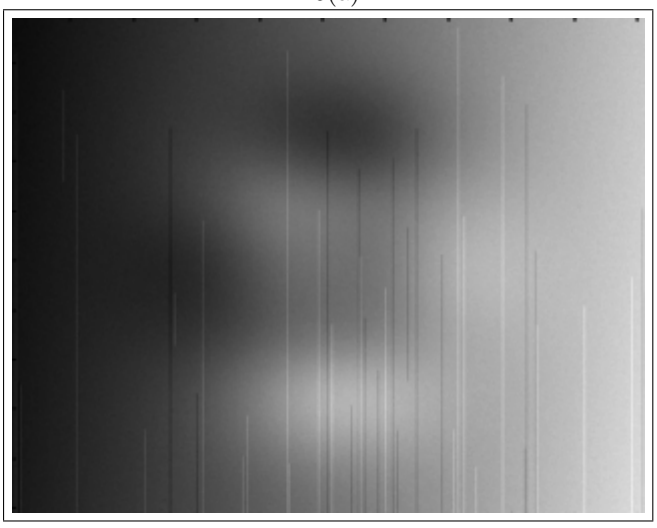
When residues are present, though, the algorithm fails. The unwrapping is possible if, and only if, every integration path encircles none or a balanced number of residue charges. The line integrals around each path must be evaluated to zero so that no error is propagated through the unwrapping process. The erroneous horizontal and vertical lines displayed in Figure 2.16 shows the outcome of using Itoh's 2D algorithm over noisy data, signalizing the occurrence of residues. Chapter 3 will introduce the main 2D Phase Unwrapping approaches and algorithms tailored to overcome the presence of residues and minimize their effect on the solution's quality.

Algorithm 2: Itoh's method for 2D Phase Unwrapping

input : Wrapped phase values, $\psi(n, n)$
output: Unwrapped phase values $\phi(n, n)$

$\phi(1,1) = \psi(1,1)$;
\\ *Unwraps the first row*
for $j \leftarrow 2$ **to** N **do**
  $\Delta_\psi \leftarrow \psi(1,j) - \psi(1,j-1)$;
  if $\Delta_\psi \leq -\pi$ **then**
    $\Delta_\psi \leftarrow \Delta_\psi + 2\pi$;
  else if $\Delta_\psi > \pi$ **then**
    $\Delta_\psi \leftarrow \Delta_\psi - 2\pi$;
  $\phi(1,j) \leftarrow \phi(1,j-1) + \Delta_\psi$;
end
\\ *Unwraps every column*
for $j \leftarrow 1$ **to** N **do**
  for $i \leftarrow 2$ **to** N **do**
    $\Delta_\psi \leftarrow \psi(i,j) - \psi(i,j-1)$;
    if $\Delta_\psi \leq -\pi$ **then**
      $\Delta_\psi \leftarrow \Delta_\psi + 2\pi$;
    else if $\Delta_\psi > \pi$ **then**
      $\Delta_\psi \leftarrow \Delta_\psi - 2\pi$;
    $\phi(i,j) \leftarrow \phi(i,j-1) + \Delta_\psi$;
  end
end

2.16(a):



2.16(b):

Figure 2.16: *2D Unwrapping process over wrapped phase data of Figure 2.6(b) corrupted with noise. (a) Itoh's 2D algorithm iterating over each row. (b) Itoh's 2D algorithm iterating over each column.*

3
Main 2D Phase Unwrapping approaches

In this chapter, we will present the two main approaches and state-of-the-art algorithms for the 2DPU problem: *path-following* methods and minimum L^p-norm methods.

Path-following algorithms directly applies the line integration schemes seen in the previous sections, relying on the concept that the Itoh condition must hold along every integration path. Whenever this condition is not met, different integration paths may lead to different unwrapped solutions due to the path dependency problem. The techniques employed to handle these inconsistencies concentrates on creating efficient artificial barriers to impede such misguided paths, which leads to a more local optimization problem.

Minimum L^p-norm methods, on the other hand, are based on the idea that the absolute and wrapped phase gradients among adjacent samples must be equal. The objective is to find a phase solution for which the L^p norm of the differences between absolute and wrapped phase gradients is minimized, leading to a global optimization problem.

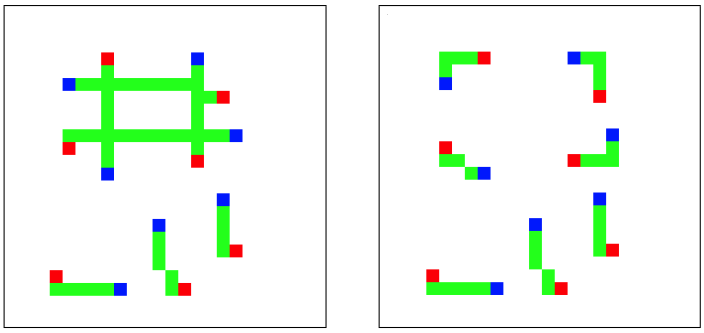
The two approaches are based on different concepts. Although none of them has a clear leverage over the other, each approach has its own advantages and deficiencies. Sections 3.1 and 3.2 gives a review at these approaches and how they apply each of these concepts in order to succeed the phase unwrapping problem. The two state-of-the-art path-following methods discussed in Section 3.1 will be used as benchmarks in Chapter 5.

3.1
Path-Following Methods

The term *branch-cuts* was first introduced by *Goldstein et. al.* to define elementary barriers connecting pairs of residues. The premise from path-following methods is based on the assumption that, if the branch-cuts are positioned in such way that no integration path could ever encircle an unbalanced number of residues, then the path-dependency problem would be eliminated.

Figure 3.1(a) shows an example with eight residues (4 positives and 4 negatives). Although the branch-cuts represented by the green lines succeeds to eliminate the path dependency problem, it is clear that they were not placed in the optimal way. The superposition of branch-cuts manages to create an isolated region that will never be reached by any integration path, and

will therefore be excluded from the unwrapping procedure. Samples inside an isolated region requires a separate unwrapping using an arbitrary initial phase value on a starting sample. Choosing the best placement for the branch-cuts is a critical task, as shown by Figure 3.1(b). The second configuration is most likely to produce visually better solutions, thus suggesting the minimization of the branch-cut lengths as an optimization objective (as highlighted later in this section, the optimal branch-cut configuration is also the optimal solution in terms of the minimization of the L^0-norm of the 2DPU problem).

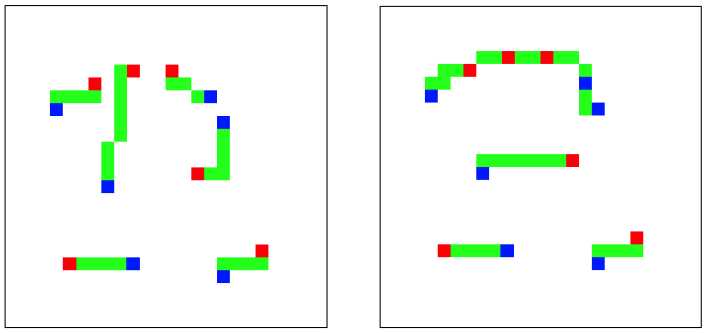


3.1(a): 3.1(b):

Figure 3.1: *Example of residues and two possible branch-cut configurations. (a) First branch-cut configuration, creating an isolated region. (b) Minimum length branch-cut configuration.*

As long as no integration path can ever encircle an unbalanced number of residues, any branch-cut configuration will eliminate the path-dependency problem. However, once a branch-cut pixel is found during the unwrapping process, the unwrapped samples from opposite sides of the branch-cuts may carry a phase discontinuity of $\pm 2\pi$, as explained by [1]. Finding the optimal placement of the branch-cuts is not an easy task, and thus some criteria must be established in order to guide the process. Figures 3.2 shows two different possible heuristics, both eliminating path-dependencies in the unwrapping procedure. One is focused on connecting closest dipoles (close pairs of opposite charged residues, Figure 3.2(a)), where the other focus on creating balanced components of closest residues (Figure 3.2(b)). Each of them may introduce critical discontinuities and are indicated to different topologies of residues.

Minimizing the sum of the branch-cuts lengths is, by itself, considered as a difficult problem, whatever heuristic is used. Sections 3.1.1 and 3.1.2 presents two state-of-the-art methods and discusses how they behave on different types of instances.



3.2(a): 3.2(b):

Figure 3.2: *Another example of residues and two possible branch-cut configurations. (a) Minimum length connections between pairs of positive and negative residues. (b) Minimum length connections forming balanced components.*

3.1.1 Goldstein's Algorithm

Goldstein et al. proposed in [1] a path-following algorithm which is effective at generating short length branch-cuts in short CPU time. The method connects nearby residues with branch-cuts until every connected component becomes balanced. The cuts are generated in an attempt to minimize the sum of the cut lengths, as follows.

The algorithm starts by scanning the wrapped phase image, sample by sample, until a first residue is detected. A 3x3-sample box is then centred around the discovered residue and the algorithm searches for other residues inside the box. If found, a branch-cut is placed connecting both residues, no matter their polarities, forming a connected component. If the component has an equal number of negative and positive residues when the search is over (i.e. the component is balanced), then the algorithm resumes the search for a next unvisited residue in the wrapped phase image. If the component is not balanced, though, the mask is placed around every one of its residues and the previous steps are repeated, until the component becomes balanced or a border point is found within the limits of the box. If no balance is reached, then the size of the box is doubled and the algorithm restarts from the current active component. The method maintains a list of visited and active residues as boolean values, indicating whether a given residue was already visited by a mask box and if it is part of a current unbalanced component. All residues are initialized as inactive and as not visited when the algorithm starts.

Although the method can be efficient for instances with a low density of residues, the opposite scenario can be problematic. The branch-cuts are placed inadvertently whenever a residue is found within the limits of the search box. Isolated regions are most likely to occur in these circumstances,

mostly on instances with a high density of residues, as shown by Figure 3.3. Several improvements over Goldstein's algorithm have been proposed in the literature ([9], [10], [2]), often suggesting the use of quality maps (discussed in Section 4.1.4) to guide the placement of the cuts and the removal of dipoles in order to diminish the occurrence of isolated regions. For its robustness and relevance to the phase unwrapping problem, Goldstein's algorithm is often used in benchmarks as a comparison parameter. The pseudo-code for the original method is shown in Algorithm 3.

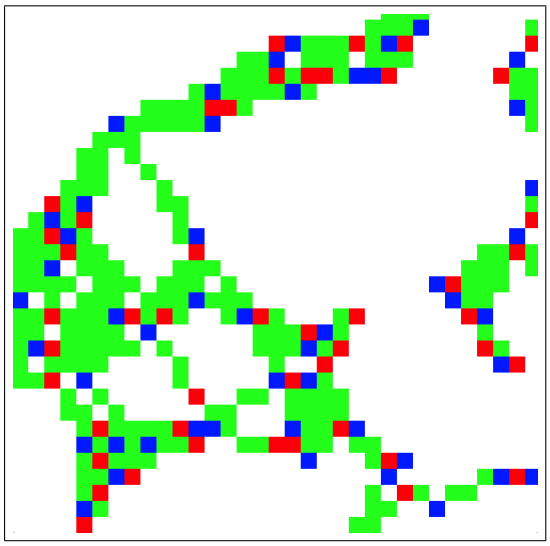


Figure 3.3: *Outcome of the Goldstein's algorithm on an instance with a high density of residues. One can observe the large number of isolates regions formed by the branch-cuts.*

3.1.2 Minimum-Cost Matching Algorithm (MCM)

Buckland et al. proposed in [11] a minimum-cost matching algorithm that solves the path-dependency problem by creating branch-cuts between close pairs of positive and negative residues, treating the minimization of the branch-cuts lengths as a global optimization problem. The method aims to find an optimal matching (i.e. a minimum-distance pairing between residues of opposite sign) using of the Hungarian algorithm. The problem is reduced to find the minimum cost matching between vertices in a bipartite graph.

Let $G = (V,E)$ be a bipartite graph where the vertex set $V = (V^+ \cup V^-)$ represents the union between the groups of positive and negative residues (V^+ and V^-, respectively) with $|V| = N$ and let the edge set E be composed by edges $e = \{(i,j),\ i \in V^+,\ j \in V^-\}$ connecting opposite signed residues in the 2D space. The objective is to find a minimum cost matching between vertices from V^+ and V^-, as described by the following integer program:

Algorithm 3: Goldstein's algorithm for 2D Phase Unwrapping

```
1  foreach inactive and not visited residue do
2      Mark the current residue as active;
3      Let balance = 1 if the residue is positive, -1 otherwise;
4      for n = 3 to MaxBoxSize do
5          foreach active pixel do
6              Centralize the n × n box at the active pixel;
7              foreach box pixel do
8                  if the box pixel is an inactive residue then
9                      if it is unvisited then
10                         Add its polarity to balance;
11                         Mark the residue as visited;
12                     end
13                     Mark the residue as active;
14                     Place a branch-cut between the active residue and
                       the box residue;
15                 end
16                 else if the box pixel is a border point then
17                     balance = 0;
18                     Place a branch-cut between the active residue and
                       the border point;
19                 end
20             end
21             if balance = 0 then jump to * ;
22         end
23     end
24     if balance != 0 then Place a branch-cut to the nearest border ;
25     *Mark all the active pixels visited and inactive.
26 end
```

$$\min \sum_i \sum_j c_{ij} x_{ij} \tag{3.1.1}$$

$$\textbf{\textit{s.t.}} \quad \sum_j x_{ij} = 1, \ \ \forall i \in V^+ \qquad \sum_i x_{ij} = 1, \ \ \forall j \in V^- \tag{3.1.2}$$

$$x_{ij} \in \{0,1\}, \forall (i \in V^+, j \in V^-), \tag{3.1.3}$$

where x_{ij} is the binary integer decision variable indicating whether the edge e_{ij} is part of the solution and c_{ij} is the edge cost. The constraints shown by Equation 3.1.2 assures that every vertex from group V^+ will be matched with exactly one vertex from group V^-, and vice-versa. In order to balance the number of residues and reduce the method's overall memory requirement, only

a small number of nearest neighbours is considered when creating the edge set. If border points are encountered during the search for nearby residues, a border source of opposite sign of its nearest residue is created and added to the list of residues. These border points are called *fictitious sources*, since they are not part of the original instance and there is frequently an unbalanced number of positive and negative residues in the wrapped 2D images. Figure 3.4 illustrates the minimum cost matching problem for the 2D phase unwrapping domain.

The Hungarian algorithm [12] can be used to solve the matching problem in polynomial time. The original method had a time complexity of $\mathcal{O}(N^4)$, although Ford and Fulkerson were able to reduce it to $\mathcal{O}(N^3)$ in [13].

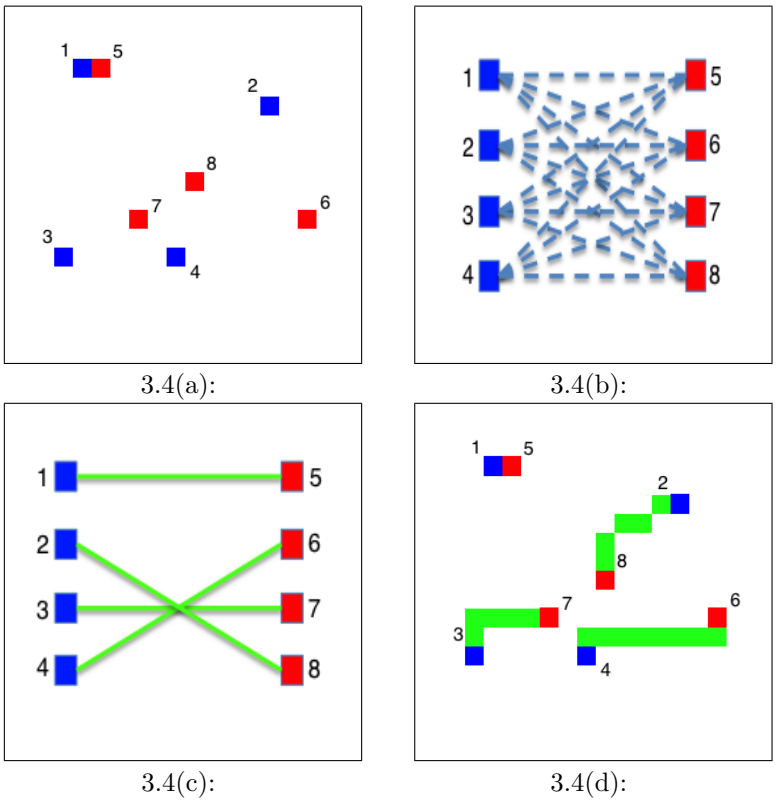
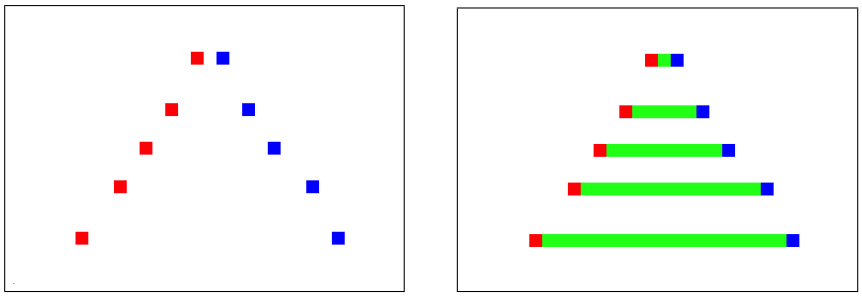


Figure 3.4: *The minimum cost matching algorithm for the 2D Phase Unwrapping problem. (a) The original instance (b) The original instance modelled as a bipartite graph. (c) The minimum cost matching found by the Hungarian method. (d) The equivalent minimum cost solution.*

Minimum cost matching algorithms tend to produce visually good solutions for instances with a high density of residues. However, they are most likely to fail in preserving the structural delimitations arising from natural discontinuities in the subject of study, as seen in Figure 3.5. *Gdeisat et. al.* later proposed in [14] a three dimensional phase unwrapping algorithm using the Hungarian method, expanding the ideas introduced in the original paper.



3.5(a): 3.5(b):

Figure 3.5: *Example where the minimum cost matching algorithm fails to preserve the topology of residues. (a) The topology of residues suggesting a triangular structural delimitation. (b) The solution obtained by the minimum cost matching algorithm.*

3.1.3 Quality Guided Algorithms

Quality guided algorithms uses the quality information extracted from phase data to guide the integration paths along the wrapped phase image, determining the order in which the phase samples are unwrapped [15]. One of the main goals of this class of methods is to overcome the potential misplacement of branch-cuts which, as previously referred, is considered a difficult problem. Quality maps are typically obtained through correlation, pseudocorrelation and phase derivative variance maps [2].

A further development in quality guided algorithms is the conception of using quality maps to delineate branch-cuts, as discussed in [10]. Quality maps constitutes an important tool to guide and support several path-following and L^p-norm methods [16], [17], [2].

3.2 Minimum L^p-norm Methods

Minimum norm methods take a completely different approach to the phase unwrapping problem. The problem is written as an integer global optimization program, whose objective is to find an absolute phase image solution for which the L^p-norm of the difference between absolute and wrapped phase gradients is minimized. Let $\phi_{m,n}$ and $\psi_{m,n}$ be the absolute and wrapped phase values of a sample located in the (m, n) position of an M x N-pixel image. The minimization of the L^p-norm for the 2DPU problem can be written as the following integer program:

$$\arg\min_{\Phi} E(\Phi) \tag{3.2.1}$$

$$\boldsymbol{s.t.}\ \ \phi_{m,n} = \psi_{m,n} + 2\mathbf{k}_{m,n}\pi\ \ \forall(m,n) \tag{3.2.2}$$

$$\mathbf{k} \in \mathbb{Z} \ \forall(m,n) \tag{3.2.3}$$

where $\mathbf{k}_{m,n}$ is the integer decision variable representing the 2π difference between the unwrapped and wrapped phase values for every sample m, n. The objective function $E(\Phi)$ represents the L^p-norm given by:

$$E(\Phi) = \sum_{m=1}^{M}\sum_{n=1}^{N-1}\left|\Delta^h\phi_{m,n} - \Delta^h\psi_{m,n}\right|^p + \sum_{m=1}^{M}\sum_{n=1}^{N-1}\left|\Delta^v\phi_{m,n} - \Delta^v\psi_{m,n}\right|^p, \tag{3.2.4}$$

where $\Delta^h\phi_{m,n}$, $\Delta^v\phi_{m,n}$, $\Delta^h\psi_{m,n}$ and $\Delta^v\psi_{m,n}$ represents the horizontal and vertical absolute and wrapped phase gradients, given by:

$$\begin{aligned}
\Delta^h\phi_{m,n} &= \phi_{m,n} - \phi_{m,n-1} \\
\Delta^v\phi_{m,n} &= \phi_{m,n} - \phi_{m-1,n} \\
\Delta^h\psi_{m,n} &= W(\psi_{m,n} - \psi_{m,n-1}) \\
\Delta^v\psi_{m,n} &= W(\psi_{m,n} - \psi_{m-1,n}),
\end{aligned} \tag{3.2.5}$$

where W is the wrapping operator defined in Equation 2.2.1. When a quality map is available, the energy function can be re-written as a weighted L^p-norm. The quality measure value $q_{m,n}$ computed over corrupted phase values (i.e. noise) plays an important role of reducing the impact of bad quality pixels over the global unwrapped solution, as described by [2]. The weighted L^p-norm is written as

$$E(\mathbf{\Phi}) = \sum_{m=1}^{M}\sum_{n=1}^{N-1} q^h_{m,n}\left|\Delta^h\phi_{m,n} - \Delta^h\psi_{m,n}\right|^p + \sum_{m=1}^{M}\sum_{n=1}^{N-1} q^v_{m,n}\left|\Delta^v\phi_{m,n} - \Delta^v\psi_{m,n}\right|^p, \tag{3.2.6}$$

where the solution's quality is directly related to the L^p-norm chosen as the objective function. Higher values of p ($p > 2$) tend to produce unreliable solutions, while p = 0,1,2 are the most representative norms in the state-of-the-art methods [2]. Since the minimization of an L^p-norm is a discrete minimization problem, many of the proposed algorithms in the literature only find approximate solutions [2].

3.2.1 L^2-norm

The L^2-norm methods consists in obtaining an absolute phase image solution where the least squares difference between the absolute and wrapped phase gradients is minimized. *Fried and Hudgin* proposed the first least squares approximate algorithms in [18] and [19], enlightening the fact that this approach tends to smooth areas of discontinuities in the unwrapped image unless the energy function is provided with binary weights.

In order to overcome the complexity introduced by the discrete optimization problem, many of the proposed algorithms relax the discrete domain $\mathbb{Z}^{MN}$ to the real domain $\mathbb{R}^{MN}$. Given this relaxation, *Ghiglia et. al.* showed in [20] that the minimization of the L^2-norm is equivalent to solving a Poisson partial different equation using the Fast Fourier transform (FFT). Despite the reasonable mathematical efficiency, *Chen et. al.* stated in [21] that most least squares algorithms give disappointing results in practice because they underestimate the true unwrapped phase gradients and produce biased results.

3.2.2 L^1-norm

When compared to the L^2-norm, L^1-norm algorithms performs better in preserving good quality samples near areas of discontinuities. Although this approach is far less cited in the literature than L^2-norm and L^0-norm methods, Flynn and Constantini proposed in [22] and [23] two algorithms that solved the L^1-norm problem to optimality.

3.2.3 L^0-norm

When $p = 0$, the objective is to minimize the total number of samples where the absolute and wrapped phase gradients are not equivalent. More generally, the solution obtained tends to preserve the wrapped phase gradients *in as many places as possible* [2].

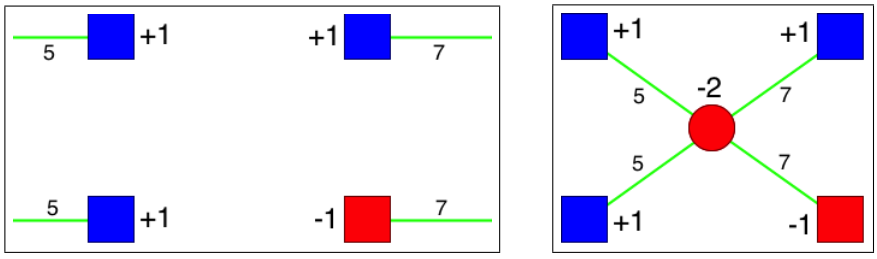
As seen in Equation 2.2.7, the integration paths unwraps the data by adding the related wrapped phase gradient $\Delta W(\phi_n)$ with a 2π increment to each sample along its way. Thus, when the Itoh condition holds along an integration path, then the absolute and wrapped phase gradients are identical. Path-following methods directly addresses the minimization of the L^0-norm, although solving it to optimality was proven to be a NP-*hard* problem [21]. Minimizing the L^0-norm is generally accepted as the most desirable phase unwrapping approach.

4
The Minimum Spanning Forest with Balance Constraints (MSFBC) approach

As seen in the previous sections, current branch-cut methods do not provide the optimal solution for the L^0-norm, but rather concentrate on restrictions of this optimization problem that have the benefit of being polynomial. Although Goldstein's algorithm finds an approximate minimal set of branch-cuts, the occurrence of isolated regions and unnecessary connections can create critical discontinuities to the unwrapped solution. The minimum cost matching algorithm, for instance, finds a minimal set of branch-cuts connecting closest dipoles, but restricts no more than two residues in a same branch.

The true minimization of the L^0-norm is a variant of the Euclidean Steiner-tree problem, known to be NP-*hard* [**?**]. We propose an alternative formulation, where we use minimum spanning trees rather than Steiner trees to cluster close groups of residues. The motivation behind this approach is to find a branch-cut configuration that is able to better respect the structural topology of residues, while still addressing the minimization of the L^0-norm. The problem is reduced to the search of a minimum spanning forest with balance constraints (MSFBC), where each tree spans a balanced number of positive and negative weighted vertices.

The 2DPU is a real-world application of the MSFBC problem (MSFBCP), where residues and border points can be directly modelled through a set of weighted vertices on a graph. Residues are mapped to ± 1 weighted vertices in the 2D Euclidean space, where the cost of each edge connecting a pair of vertices is determined by the Euclidean distance between residues in the wrapped phase image. The concept of border points can be interpreted as a single special vertex in the graph whose weight balances the difference between the total number of positive and negative residues (i.e, the positive or negative excess) of the instance, as shown by Figure 4.1. The cost of the edge connecting a vertex to the special vertex is equal to the distance between the related residue and its nearest border point. Section 4.1 presents a formal definition for the more generalized MSFBC problem on graphs through a set of mathematical formulations and a proof of its complexity. Section 4.2 devises a solving procedure for the relaxed problem, while Section 4.3 proposes a series of dual heuristic methods. Finally, an exact branch-and-cut method and a metaheuristic approach are proposed in Sections 4.4, 4.5 and 4.6.



4.1(a): 4.1(b):

Figure 4.1: *The 2DPU-MSFBCP particularities. (a) The 2DPU-MSFBC instance with border points. (b) The equivalent MSFBC instance with a special vertex replacing the concept of border points. The weight of the special vertex balances the positive excess of the instance.*

4.1 Problem Formulation and Complexity

Let $G = (V,E)$ be a graph with positive edge costs, where every vertex $v \in V$ has a weight $w_v \in \{-1, 1\}$. Let d_e be the cost (distance) of edge $e \in E$ and x_e be the decision variable indicating whether edge e should be part of the solution. The undirected formulation for the generalized minimum spanning forest with balance constraints problem can be written as the following integer program:

$$\min \sum_{e \in E} d_e x_e \tag{4.1.1}$$

$$\textbf{\textit{s.t.}} \sum_{e \in \delta(S)} x_e \geq 1, \ \forall S \subset V, \ s.t. \ \sum_{v \in S} w_v \neq 0 \tag{4.1.2}$$

$$x_e \in \{0, 1\}, \forall e \in E, \tag{4.1.3}$$

where the set of constraints indicated by Equation 4.1.2 implies that for any unbalanced cut $S = (S, V\text{-}S) \in$ V, at least one edge $e \in \delta(S)$ is included in the solution set, where $e = \{(i,j),\ i \in S$ and $j \in (V\text{-}S)\}$. More generally, it assures that every unbalanced cut S must be connected to a least one vertex from $(V\text{-}S)$ in order to balance the number of positive and negative weighted vertices of that particular cut.

Since each constraint is satisfied when the optimal solution is reached, then every connected component described by the solution set $\overline{x}$ must be balanced. If we consider an unbalanced cut $U \subset V$, then at least one minimum cost edge $e = \{(i,j),\ i \in U$ and $j \in (V\text{-}U)\}$ must be added to the solution set, leading to the following scenarios: in the case where the selected edge e

connects U to a set of vertices balancing the cut, then the constraint related to U is enough to produce a balanced tree. If the edge e does not balances the cut U, then the unbalanced cut U' will be formed. Since the set of constraints considers all unbalanced cuts, U' will be naturally present as a constraint and the same scenarios will be applied for it.

We also introduce a directed formulation, where unbalanced cuts with a positive or negative excess are considered in different sets of constraints. The program is re-written as:

$$\min \sum_{e \in E} d_e x_e \tag{4.1.4}$$

$$\textbf{\textit{s.t.}} \quad \sum_{a \in \delta^+(S)} x_a \geq 1, \;\; \forall S \subset V, \;\; s.t. \;\; \sum_{v \in S} w_v > 0 \tag{4.1.5}$$

$$\sum_{a \in \delta^-(S)} x_a \geq 1, \;\; \forall S \subset V, \;\; s.t. \;\; \sum_{v \in S} w_v < 0 \tag{4.1.6}$$

$$x_e + x_{e'} \leq 1, \;\; \forall e = (i,j), e' = (j,i) \in E \tag{4.1.7}$$

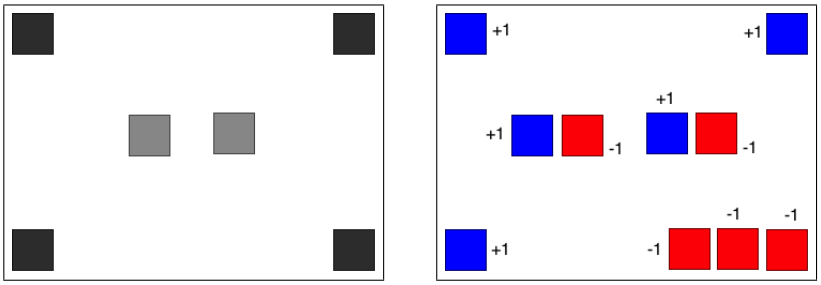
$$x_e \in \{0,1\}, \forall e \in E \tag{4.1.8}$$

The first set of constraints (Equation 4.1.5) indicates that for every unbalanced cut with a positive excess, one or more edges $a = \{(i,j),\ i \in S$ and $j \in (V\text{-}S)\}$ leaving S must be included in the solution. The second set of constraints (Equation 4.1.6) implies that for every unbalanced cut with a negative excess, one or more edges $a = \{(i,j),\ i \in (V\text{-}S)$ and $j \in S\}$ entering S must be included in the solution. Finally, the third set of constraints (Equation 4.1.7) prohibits the selection of a pair of edges with opposite direction. Collectively, the set of constraints (Equations 4.1.5, 4.1.6 and 4.1.7) enforces that no unbalanced spanning tree would be ever present in the optimal solution.

A feasible solution for the MSFBCP can be characterized as a partition $P = (P_1,...,P_k)$ of the set of vertices, where each set P_i for i $\in \{1,\ldots,k\}$ is described by a group of vertices such that $\sum_{v \in P_i} w_v = 0$. In order to prove that the MSFBC optimization problem is NP-*hard*, consider the following reduction from the Steiner-tree problem on graphs [24].

Theorem 1 *The MSFBC problem is NP-hard.*

Proof: Let $G = (V,E)$ be a graph with positive edge costs and a set of terminal vertices $T \subseteq V$. The Steiner-tree optimization problem seeks to find a minimum cost connected subgraph $G' = (V',E')$ with $T \subseteq V'$. Given an instance for the Steiner problem, the reduction is achieved by assigning a positive weight of $w = 1$ to all terminal vertices, except for one, which is replaced by $|T|$-1 vertices with a negative weight of w = -1, connected to each other with cost zero. In addition, a pair of connected vertices with opposite signed weights is located for each (V-T) remaining vertices (called Steiner-points). The cost of connecting such pair of vertices is also set to zero. Figure 4.2 demonstrates the reduction for a classical Steiner-tree problem instance with four terminal nodes and two Steiner-points, while Figure 4.3(a) illustrates the optimal solution for the Steiner-tree problem. After computing a solution for the MSFBCP on the reduced instance, one of the following cases can arise: (1) There is a single balanced tree which spans all terminal vertices (Figure 4.3(b)); (2) There are at least two disjoint balanced trees containing terminal vertices (Figure 4.3(c)). In the first case, the related minimum spanning tree already constitutes an



4.2(a): 4.2(b):

Figure 4.2: *The instance reduction from the Steiner-tree problem to the MSFBC. (a) The Steiner-tree instance, with four terminal vertices (black) and two Steiner-points (gray). (b) The equivalent reduced instance for the MSFBC problem with three positive weighted vertices (blue) with a $w = 1$ weight replacing the $|T-1|$ terminal vertices, $|T-1|$ negative weighted vertices (red) with a w = -1 weight replacing the remaining terminal vertex and the pairs of positive and negative weighted vertices replacing the Steiner-points with $w = 1$ and w = -1, respectively.*

optimal Steiner-tree. All remaining trees of the MSFBC solution are related to unused Steiner-points (2-node balanced trees with cost zero), which do not impact on the solution cost ($c(P) = \mathrm{w}(G') + 0 = \mathrm{w}(G')$). The MSFBC solution constitutes a feasible and optimal solution for the Steiner-tree problem.

In the second case, since there are disjoint trees containing terminal vertices, we can assume that there is at least one unused edge with zero cost connecting a pair of vertices which replaces a Steiner point. This scenario can

be proven by contradiction: if all zero cost edges are used in the solution, then there must be at least one unbalanced tree in the disjoint set of trees. The connection of such pair of vertices would sum an excess of plus or minus one to that tree, turning it to be unbalanced as illustrated in Figure 4.3(d). Given that the trees must be balanced, this solution would be infeasible for the MSFBCP. We can now show that there is another solution, with the same cost, where disjoint trees can be merged together and at least one unused zero cost edges is included.

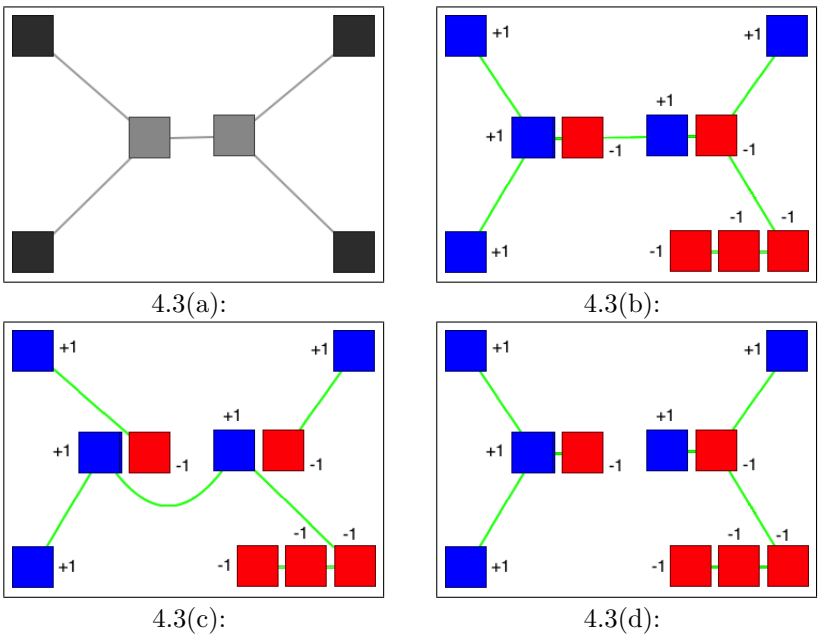


Figure 4.3: *The optimal solution for the Steiner-tree problem and two feasible solutions for the MSFBC. (a) The optimal solution for the Steiner-tree problem. (b) An optimal solution for the MSFBC, also feasible and optimal for the Steiner-tree problem. (c) Another solution for the MSFBC with the same cost, but infeasible for the Steiner-tree problem. (d) An infeasible solution for the MSFBC.*

Let $T_1 = (V_1,\ E_1)$ and $T_2 = (V_2,\ E_2)$ be two disjoint balanced trees containing terminal vertices and let $E_{1,2}$ be the set of edges connecting vertices from T_1 to T_2. If we compute the minimum spanning tree over the merged component $T_m = (V_m,\ E_m)$, where $V_m = \{V_1 \cup V_2\}$ and $E_m = \{E_1 \cup E_2 \cup E_{1,2}\}$, naturally every edge in T_1 and T_2 will figure in T_m. But since a minimum spanning tree algorithm lists the edge set in increasing order, every zero cost edge including those in $E_{1,2}$ will be treated first. The addition of such edges will connect the disjoint components and the cost of the new component will be given by:

$$c(T_m) = c(T_1) + c(T_2) + 0 = c(T_1) + c(T_2). \tag{4.1.9}$$

This procedure can be executed iteratively until a single tree spans all terminal vertices, leading to the first case where $c(P) = \mathrm{w}(G')$. Since we are able to build a valid and optimal Steiner-tree by solving the MSFBCP on the reduced instance, we prove by extension that the MSFBC is also NP-*hard*.

4.2 Linear Program (LP)

The linear relaxation for the directed MSFBC formulation can be written as:

$$\min \sum_{e \in E} d_e x_e \tag{4.2.1}$$

$$\textbf{\textit{s.t.}} \quad \sum_{a \in \delta^+(S)} x_a \geq 1, \ \forall S \subset V, \ s.t. \ \sum_{v \in S} w_v > 0 \tag{4.2.2}$$

$$\sum_{a \in \delta^-(S)} x_a \geq 1, \ \forall S \subset V, \ s.t. \ \sum_{v \in S} w_v < 0 \tag{4.2.3}$$

$$x_e + x_{e'} \leq 1, \ \forall e = (i,j), e' = (j,i) \in E \tag{4.2.4}$$

$$0 \leq x_e \leq 1, \ \forall e \in E, \tag{4.2.5}$$

$$x_e \in \Re, \forall e \in E, \tag{4.2.6}$$

where the decision variables x_e can now assume any real value from the [0,1] interval. This relaxation technique transforms the NP-hard integer optimization problem into a related problem which can be solvable in polynomial time and can provide information for devising an exact branch-and-cut method. Linear programming has an important property which states that if a problem has a separation routine which runs in polynomial time, then the linear program can also be solved in polynomial time even with an exponential number of constraints [25]. Hence, solving the integer MSFBCP is NP-*hard*, but the relaxed linear program can be polynomially solved.

4.2.1 Solving the Linear Program (LP)

Because of the exponential number of constraints, it is unrealistic to assume that even the relaxed MSFBC problem could be solvable in a reasonable time without a good set of decomposition techniques.

Since the constraints are all related to unbalanced cuts, we can start by solving an initial LP which considers as constraints only the unbalanced cuts containing a single vertex. The optimal solution for this initial subproblem

describes a minimum spanning forest where all the vertices are connected to at least one other vertex, for which the balance constraint may or may not be respected for each tree. The LP is modified by adding a new constraint for every unbalanced connected component, and the resulting program is re-optimized. This process is repeated, iteratively, until the optimal linear solution describes a set of balanced trees. At each iteration, the new constraints assures that all unbalanced cuts formed by the previous solutions will be connected to at least one additional vertex. In the worst case scenario, all the possible unbalanced cuts $S \subset R$ will be individually added as constraints to the formulation before the optimal solution is reached.

Let $\overline{x}$ be the current linear optimal solution and $\overline{G_1} = (\overline{V}, \overline{E_1})$ and $\overline{G_2} = (\overline{V}, \overline{E_2})$ be auxiliary directed and undirected graphs, respectively, where $\overline{E} = \{e \mid \overline{x}_e > 0\}$. The separation of cuts is done by solving the maximum flow/minimum cut problem, computed efficiently in O(nm $\log(n^2/m)$) by Goldberg and Tarjan's algorithm [26], for: (1) all pairs of vertices in a same connected component; (2) pairs between vertices from a given connected component and an external vertex. Each edge capacity is defined by $\overline{x}_e$ as shown by Algorithm 4. The set of cuts with an unbalanced group of vertices must be individually added as a constraint. Algorithm 5 shows the pseudo-code for the solving procedure.

Although the linear relaxation is able to produce feasible solutions with integer decision variables, its solution set is typically non integral and infeasible for the original problem. The integer optimal solution constitutes a feasible but not necessarily optimal solution for the relaxed program. The relaxation of a constraint can only lower the optimal solution value for a given problem, implying that the optimal solution set for the LP works as a lower bound for the integer program.

While rounding the relaxed decision variables does not necessarily produce the optimal solution for the original problem, it can constitute a rather good approximation if the solution maintain its feasibility. The gap between the optimal solutions for the LP and for the integer program is referred as the *integrality gap* of the linear program.

Several techniques have been proposed in the literature to derive an integer solution from the linear optimum. These techniques, called primal heuristics, involves successive variable fixing and rounding (according to the problem's particularities), followed by the re-optimizations of the modified LP [27]. While the heuristic approach does not guarantee that the optimal integer solution will be ever achieved, they constitute a powerful tool to generate reasonably good solutions. Heuristic solutions establishes upper bounds for the

Algorithm 4: LP cuts separation routine

input : The undirected graph $\overline{G_1} = (\overline{V}, \overline{E_1})$ and the directed graph $\overline{G_2} = (\overline{V}, \overline{E_2})$, both described by the solution set **x**
output: The set of cuts **S**

Initialization: *For each node v in* $\overline{V}$*, assign* visited*[v]* $\leftarrow$ *false. Let* C *be the list of connected components and* S *the list of cuts.*
foreach *node v in* $\overline{V}$ **do**
  if *vertex v was not visited* **then**
    Run a depth-first-search on graph $\overline{G_1}$, starting from node v.
    Add the discovered connected component to C.
  end
end
foreach *connected component* $c_k \in$ C **do**
  foreach *pair of nodes (i,j)* $\in c_k$ **do**
    {s', MaxFlow} $\leftarrow$ minCutMaxFlow(i,j,$\overline{G_2}$);
    if s' *is unbalanced, MaxFlow* $<$ *1 and* s' $\notin$ S **then**
      $S \leftarrow S + s$'
    end
  end
  foreach *node* i $\in c_k$ *and a single node* j $\notin c_k$ **do**
    {s', MaxFlow} $\leftarrow$ minCutMaxFlow(i,j,$\overline{G_2}$);
    if s' *is unbalanced, MaxFlow* $<$ *1 and* s' $\notin$ S **then**
      $S \leftarrow S + s$'
    end
  end
  if s(c_k) *is unbalanced and* s(c_k) $\notin$ S **then** $S \leftarrow S + s(c_k)$;
end
return S;

original integer program. Several primal heuristics have been proposed in the literature, for a wide variety of problems and applications. An detailed survey on primal heuristics for mixed-integer and linear programming can be seen in [28].

4.2.2 Reverse Delete Step

Applied as a primal heuristic, the reverse delete step method takes a straightforward approach to construct a feasible solution for the integer program. The method uses a strict greedy approach, where elements are considered for removal by their related costs.

Since the relaxed decision variables indicates whether a given edge should be part of the solution in the MSFBC relaxed program, then a fractional solution set is most likely to lead to a group of edges that produces cycles or

Algorithm 5: LP solving procedure

input : The instance of the problem
output: The optimal solution set

```
1  Initialization: Solve the initial LP considering only the cuts with
   single vertices as constraints. Let x̄ be the solution set, lp the current
   linear program and balanced a boolean variable that indicates if all
   trees are balanced.
2  x̄ ← solve(lp)
3  balanced ← false;
4  while not balanced do
5  |  Build the directed graph G1 = (V, E1) and the undirected graph
   |  G2 = (V, E2) from the solution set x̄;
6  |  balanced ← true;
7  |  S ← SeparationRoutine( G1, G2 );
8  |  foreach cut s ∈ S do
9  |  |  if s ∉ lp then
10 |  |  |  lp ← lp + {s};
11 |  |  |  balanced ← false;
12 |  |  end
13 |  end
14 |  if not balanced then
15 |  |  x ← solve(lp);
16 |  end
17 end
18 return x̄;
```

unnecessary connections in the modelled graph. In order to produce a feasible solution, the following method of removing unnecessary edges and cycles is proposed.

First, the set of edges is sorted in decreasing order by their related costs. The algorithm then proceeds by removing one edge at time and checking if the solution maintains its feasibility. If the removal of an edge unbalances the trees, then the edge is re-inserted and marked as essential. If not, its related decision variable is set to zero. At the end of the algorithm, all remaining fractional decision variables are rounded to one and a feasible integer solution is obtained. The procedure is illustrated in Algorithm 6.

The reverse delete step method is executed over every edge $e \in \overline{x}$, where $x_e > 0$ and $|\overline{x}| = \mathrm{m}$. The detection of unbalanced cuts is done through a depth-first-search algorithm, in $\mathcal{O}(n)$, where n is the number of vertices in the graph. Hence, the running time is $\mathcal{O}(mn)$.

Algorithm 6: Reverse Delete Step - Primal heuristic

input : The linear optimal solution from the LP
output: A feasible integer solution

```
1  Initialization: Sort the solution set x̄ by their related edge cost in
   decreasing order.
2  Build the graph G from the solution set x;
3  x̄* ← x̄;
4  foreach x_e in x̄* do  x_e = 1 ;
5  foreach x_e in x̄ do
6  |  Remove edge e from G;
7  |  if G describes a set of balanced trees then
8  |  |  Fix x_e = 0 in x̄*
9  |  end
10 |  else
11 |  |  Re-insert e in G and mark as essential
12 |  end
13 end
14 return x̄*;
```


4.3 Dual Approach and Heuristics

In linear programming, every program can be formulated from two different perspectives: the primal and its corresponding dual problems. The concept of *duality* [29] is extensively used in combinatorial optimization to devise techniques and methods to bound the optimal solution of a problem and build exact methods. Although the solution values for the primal and dual problems may not be equivalent, the dual approach is able to provide a good lower bound on the optimal solution for the primal minimization problem. The gap between the primal and dual solutions is referred to as the *duality gap*.

In the primal form of a minimization problem, the objective function describes a linear combination of n variables bounded by m constraints. The dual form of the same program is written as a maximization problem, where a *dual variable* π_i is assigned for each primal constraint. The objective function is re-written as a linear combination of the m dual variables, subjected to n constraints, one for each primal variable.

In both the undirected and directed primal formulations for the MSF-BCP, we consider a set of constraints related to every unbalanced cut $S \subset R$. Assigning a *dual variable* π_S to each of these unbalanced cuts, the undirected dual program can be written as:

$$\max \sum_{S \subset V: \sum_{v \in S} p_v \neq 0} \pi_S \tag{4.3.1}$$

$$\textbf{\textit{s.t.}} \sum_{S \subset V, \sum_{v \in S} w_v \neq 0: e \in \delta(S)} \pi_S \leq d_e, \ \ \forall e \in E \tag{4.3.2}$$

$$\pi_S \geq 0, \forall S \subset V \tag{4.3.3}$$

The problem now consists in maximizing the sum of the dual variables related to violated cuts. The set of constraints described by Equation 4.3.2 states that the sum of the dual variables associated to the unbalanced cuts containing a given edge $e \in E$ is, at most, equal to the cost of that edge. These constraints can be described in a more compact form by considering the concept of *reduced costs*.

The reduced cost of a variable indicates the amount by which the primal objective function would have to increase (in a minimization problem) before the associated primal decision variable could assume a strictly positive value. More generally, the reduced cost of a variable gives a measure of its impact on the cost of the objective function. For each edge $e \in E$, the related reduced cost $\bar{c}_\pi(e)$ is defined by:

$$\bar{c}_\pi(e) = d_e - \sum_{S \subset V: e \in \delta(S)} \pi_S. \tag{4.3.4}$$

An edge is said to be *saturated* when its related reduced cost is equal to zero, indicating that the associated primal variable has a strictly positive value. For the directed formulation, we consider the additional set of dual variables v_e related to the set of constraints represented by Equation 4.2.4, for every $e \in E$. The directed dual formulation is written as:

$$\max \sum_{S \subset R: \sum_{v \in S} w_v > 0} \pi_S + \sum_{S' \subset R: \sum_{v \in S'} w_v < 0} \pi'_{S'} - \sum_{e \in E} v_e \tag{4.3.5}$$

$$\textbf{\textit{s.t.}} \ \Big(\sum_{S \subset V, \sum_{v \in S} w_v > 0: e \in \delta^+(S)} \pi_S \Big) + \Big(\sum_{S' \subset V, \sum_{v \in S'} w_v < 0: e \in \delta^-(S')} \pi'_{S'} \Big) - v_e \leq d_e, \ \ \forall e \in E \tag{4.3.6}$$

$$\pi_S, \pi'_{S'} \geq 0, \forall S \subset V, \ \ v_e \geq 0, \forall e \in E, \tag{4.3.7}$$

where the reduced costs are expressed by:

$$\overline{c}_\pi(e) = d_e - \{(\sum_{S \subset V, \sum_{v \in S} w_v > 0 : e \in \delta^+(S)} \pi_S) + (\sum_{S' \subset V, \sum_{v \in S'} w_v < 0 : e \in \delta^-(S')} \pi'_{S'}) - v_e)\}. \quad (4.3.8)$$

The duality theorem implies a relationship between the primal and dual problems known as *complementary slackness* [30], which describes a link between variables in one problem and the associated constraints in the other. Specifically, it states that if a primal variable assumes a positive value when the optimal solution is reached, then its associated dual inequality constraint holds with equality at the dual optimal solution. In other words, the reduced cost of the primal variable is set to zero (i.e, the primal variable becomes saturated). If a dual inequality constraint holds as a strict inequality when the dual optimum is reached, then the associated primal variable holds a value of zero in the primal optimal solution. The complementary slackness theorem asserts that no slack can be present in both a constraint and its associated dual variable. In the MSFBC directed problem, the theorem can be interpreted as the following set of properties.

Property 1: for every edge $e \in$ E, one of the following two conditions is met when the optimal solution is reached.

$$(\sum_{S \subset V, \sum_{v \in S} w_v > 0 : e \in \delta^+(S)} \pi_S) + (\sum_{S' \subset V, \sum_{v \in S'} w_v < 0 : e \in \delta^-(S')} \pi'_{S'}) - v_e = d_e; \quad or \quad (4.3.9)$$

$$x_e = 0; \quad (4.3.10)$$

Property 2: for every dual variable π_S, one of the following three conditions is met when the optimal solution is reached.

$$\sum_{a \in \delta^+(S)} x_a = 1, \quad s.t. \sum_{v \in S} w_v > 0; \quad or \quad (4.3.11)$$

$$\sum_{a \in \delta^-(S)} x_a = 1, \quad s.t. \sum_{v \in S} w_v < 0; \quad or \quad (4.3.12)$$

$$\pi_S = 0; \quad (4.3.13)$$

The concepts of duality and complementary slackness will play an essential role for the pre-processing stage of the branch-and-cut method in Section 4.4. The lower and upper bounds of a problem can be used to reduce its instance through the reduced costs obtained by the dual solution. This technique is called *reduced cost fixing* [31], and goes as follows. Consider the following theorem:

Teorema 4.1 *Let* I *be an instance of the problem,* $\overline{\pi}$ *a dual feasible solution with value* $\mathrm{v}(\overline{\pi})$ *and* P* *the best primal solution known for* I. *A primal decision variable* x_e *can be fixed to zero (i.e, removed from* I*) if the following condition is met:*

$$\mathrm{v}(\pi) + \overline{c}_\pi(\mathrm{e}) > \mathrm{P}^*. \tag{4.3.14}$$

Proof: If we suppose that the edge e is in the optimal primal solution set, then $x_e > 0$. By extension, the related dual constraint can be augmented up to $\overline{c}_\pi(e)$ units without affecting the feasibility of the dual solution, since a higher value will produce negative reduced costs. If this is possible, then the dual solution cost becomes $v'(\pi) = v(\pi) + \overline{c}_\pi(e)$, which by definition denotes a lower bound to the problem. If $v'(\pi) > P^*$, the lower bound becomes worse than the best known solution for the primal problem, which is absurd. More generally, if the cost of the dual solution plus the reduced cost of a given variable x_e exceeds the value of the best known primal solution, then we know for sure that x_e will never figure in the solution set.

After computing a primal and a dual solutions, the instance can be easily reduced by eliminating all variables whose reduced cost exceeds the gap between the lower and the upper bounds for the problem. Section 4.3.1 and 4.3.2 proposes a set dual heuristic procedures to obtain and improve dual solutions, while Section 4.4 shows how the instance reduction can be used as a pre-processing stage for the exact method. The results produced by the dual heuristics will be discussed in Chapter 5.

4.3.1 Dual Ascent

Wong proposed in [32] a constructive dual heuristic approach to obtain good feasible solutions for the dual program of the Steiner-tree problem on graphs, based on the rooted multi-flow formulation with directed cuts [33]. The algorithm considers the original graph G and $G_{\overline{\pi}}$, a subgraph of G described by the saturated arcs produced by the dual solution $\overline{\pi}$. At each iteration, the method chooses a violated cut W (described by the primal formulation as a constraint) and augment the associated dual variable π_W until at least one arc $e \in W$ becomes saturated. The saturated arcs are then included in $G_{\overline{\pi}}$, until $\overline{\pi}$ describes a feasible solution. A violated cut in the given formulation of the Steiner-tree problem denotes a cut R which contains at least one terminal vertex and does not contain either the root vertex or a path connecting a

terminal vertex $t \notin R$ to R. Algorithm 7 shows the pseudo-code for Wong's method.

Algorithm 7: Wong's Dual Ascent method

input : A dual initial solution $\overline{\pi}$
output: A feasible dual solution $\overline{\pi}$'

```
1 Initialization: Build G_π̄ from the saturated arcs in π̄
2 π̄' ← π̄;
3 while exists a violated cut R ∈ G_π̄ do
4 |   W ← selectViolatedCut(G_π̄);
5 |   Augment π'_W until at least one arc in δ⁻(W) becomes saturated;
6 |   Add the newly saturated arcs in G_π̄;
7 end
8 return π̄';
```

The *selectViolatedCut* routine can be based on any selection criterion, as will be discussed later on. Given the similarity between the Steiner directed cut formulation and our formulation for the MSFBCP, we propose the following dual heuristic based on Wong's method.

The notion of violated cuts can be interpreted as cuts with an unbalanced number of positive or negative vertices. We start with a zero cost dual solution ($\overline{\pi} = 0$), where we first consider the set of unbalanced cuts containing one vertex. The selection of the violated cuts is done by two different criteria: (1) by the violated cut which contains the minimum reduced cost edge in its edge set. (2) By random choice. Although both criteria are able to produce feasible and maximal dual solutions, the quality of the lower bounds obtained for a given instance can be directly related to which criterion was chosen. A more greedy approach is most likely to produce worse lower bounds, as will be showed in Chapter 5. An integer primal solution can be easily obtained by applying the reverse delete step routine over the set of saturated arcs. The dual solution is said to be maximal, since no dual variable can be further augmented without producing negative reduced costs and infeasible solutions. The pseudo-code for the dual ascent procedure for the MSFBCP is shown in Algorithm 8. The algorithm runs in $\mathcal{O}(|E|)$ iterations in the worst case, when all arcs becomes saturated. The detection of violated cuts is done by a depth-first-search procedure over the graph $G_{\overline{\pi}}$, which runs in $\mathcal{O}(|V|)$ since $|V|$ always dominates over $|E|$ in $G_{\overline{\pi}}$. The running time for the selection of violated cuts depends directly on the criterion chosen. If the selection is made on an edge based criterion, the running time is $\mathcal{O}(|E|)$, which then determines that the dual ascent procedure runs in $\mathcal{O}(|E|^2|V|)$. If, on the other hand it is a

Algorithm 8: Dual Ascent method - Dual heuristic

input : A dual initial solution $\overline{\pi}$
output: A feasible dual solution $\overline{\pi}$'

```
1  Initialization: Build G_π̄ = (V, E) from the saturated arcs in π̄
2  π̄' ← π̄;
3  while exists a violated cut R ∈ G_π̄ do
4      W ← selectViolatedCut();
5      if Σ_{v∈W} p_v > 0 then
6          Augment π'_W until at least one arc in δ⁻(W) becomes
           saturated;
7      end
8      else if Σ_{v∈W} p_v < 0 then
9          Augment π'_W until at least one arc in δ⁺(W) becomes
           saturated;
10     end
11     Add the newly saturated arcs in G_π̄;
12 end
13 return π̄';
```

cut based criterion, the running time is pseudo-polynomial, $\mathcal{O}(|E|^2|W|)$, where $|W|$ represents the total number of violated cuts considered.

4.3.2 Dual Scaling

It is possible to improve the feasible and maximal dual solution $\overline{\pi}$ obtained in the Dual Ascent procedure through a simple routine called *Dual Scaling*. The method consists in multiplying the dual solution by a constant factor $0 < \alpha < 1$ (thus, obtaining a feasible but not maximal dual solution $\overline{\pi}$') and recomputing the Dual Ascent method over $\overline{\pi}$', which can produce a potentially better lower bound. New sets of dual variables are likely to assume a positive value, describing additional cuts not previously considered in the dual solution and thus augmenting the lower bound for the problem. This routine can be executed iteratively in a pre-fixed number of iterations, as shown by Algorithm 9.

Algorithm 9: Dual Scaling method - Dual heuristic

input : A feasible and maximal dual initial solution $\overline{\pi}$
output: A feasible and maximal potentially improved dual solution $\overline{\pi}^*$

```
1 π̄* ← π̄;
2 for it = 0 to MaxIterations do
3 |   π̄* ← π̄* × α;
4 |   π̄* ← DualAscent(π̄*);
5 |   if v(π̄*) > v(π̄*) then break;
6 end
7 return π̄*;
```

4.4 Branch-and-cut Exact Method

Based on the set of primal and dual formulations and their proposed heuristics methods, we can now devise an exact algorithm capable of solving every instance to optimality in a finite number of steps. The following proposed algorithm is based on the concept of *branch-and-bound* methods [31].

Branch-and-bound represents a class of algorithms commonly used in combinatorial optimization which implicitly enumerates every possible solution for a given instance of the problem. By relying on subroutines to compute a lower and an upper bound on the optimal solution, the algorithm builds a solution tree where each node represents a particular subproblem of the original program. Initially, the set of bounds is computed over the instance. If there is no gap between the upper and lower bounds, then the optimal solution was already found and the search is over. On the other hand, if a gap exists, the problem is partitioned into one or more subproblems. This routine is referred to as *branching*.

In the case of an integer optimization problem, the branching is done by fixing the value of one of the problem's decision variables, which are called the *branching variables*. Two new subproblems arises from every node in the tree, fixing the value of the current branching variable to 0 or 1, accordingly, as shown in Figure 4.4. Each of these subproblems is solved recursively, until the bounds coincide. If the lower bound for a particular branch becomes worse than the best upper bound found so far, the related branch is cut from the tree. At the end of the algorithm, the best integer solution found will also denote the optimal integer solution for the problem. The number of solutions considered (i.e. nodes visited in the tree) directly depends on the quality of the upper and lower bounds obtained; better routines yields less ramifications and more pruning operations. The lower bounds can be obtained heuristically by

a dual ascent procedure or by an exact method, solving the linear relaxation of the problem at each node. The proposed *branch-and-cut* method involves

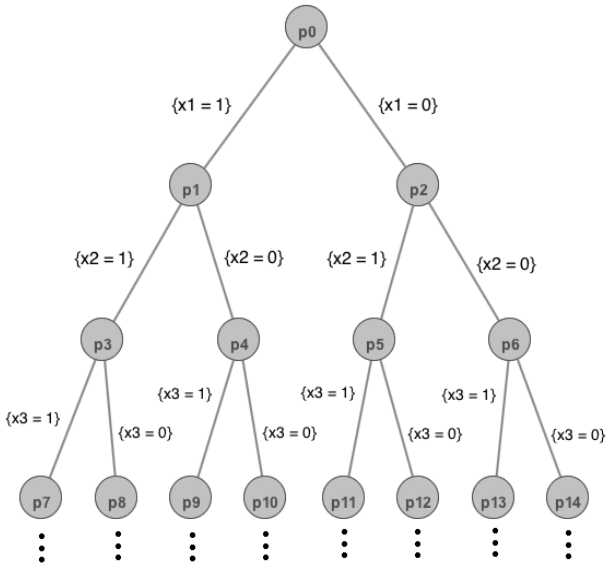


Figure 4.4: *The branch-and-bound solution tree.*

running the branch-and-bound algorithm by solving a set of linear relaxed problems. The method is composed by three sub-routines: pre-processing, initialization and branching.

Pre-processing: We start by using the set of dual heuristics from Section 4.3 to reduce the instance of the problem. We obtain an initial lower bound by constructing a dual feasible solution $\overline{\pi}$ through the Dual Ascent algorithm and later possibly improving it with the Dual Scaling procedure. Given an upper bound provided by the best known primal solution, the instance I can be reduced by removing the set of arcs whose reduced costs exceeds the gap between the lower and the upper bounds, as shown by Algorithm 10.

Initialization: The tree is initialized by solving the linear relaxed program formulated over the reduced instance I^*, using the method discussed in Section 4.2 which gradually inserts new constraints to the problem. The set of unbalanced cuts W described by the dual solution $\overline{\pi}$ ($\pi_w > 0, w \in W$) constructed by the Dual Ascent procedure can be inserted as constraints to the LP formulation as a starting point, thus saving some computational effort. The linear optimal solution found denotes a lower bound for the problem, while the upper bound can be obtained through a set of primal heuristics. If

the bounds coincide at this point, then the algorithm ends and the optimal solution has been found. If not, the branching routine is executed.

Branching: For every candidate solution S, three scenarios may occur. First, if the lower and the upper bounds coincide, then the search for an integer candidate solution in the related branch is over. Secondly, if the lower bound for the current branch is worse than the current best upper bound, the related branch is cut from the tree and the search resumes from the nearest ramification point using a backtracking procedure. Finally, if the bounds obtained are unable to prune the tree, then the branching procedure creates two new subproblems, *s1* and *s2*. Let x_s be the current branching variable. In *s1*, we constraint that x_s should be included in the solution set ($x_s = 1$), while in *s2* we constraint that it should not be included ($x_s = 0$). Generally, the chosen branching variable is the most fractional variable (i.e. the one closest to 0.5), although *Achterberg, Koch and Martin* stated in [34] that this decision is the same as a complete random choice. The subproblems are then solved recursively, always updating the best lower and upper bounds found so far. In the case where all decision variables have been fixed in the current branch, the integer solution found is compared with the current best solution.

Algorithm 10: Branch-and-cut method - Pre-process

input : The instance of the problem I
The best known primal solution P^*.

output: The reduced instance I^*
The lower bound lb
The upper bound ub

```
1  π̄ ← 0;
2  π̄ ← Dual_Ascent(π̄);
3  π̄ ← DualScaling(π̄);
4  ub ← v(P*);
5  I* ← I;
6  foreach edge e ∈ I do
7  |   if v(π̄) + c̄_π(e) > ub then
8  |   |   Remove e from I*;
9  |   end
10 end
11 lb ← v(π̄);
12 return {I*, lb, ub};
```

Algorithms 11 and 12 shows the pseudo-code for the branch-and-cut procedure. The set of variables x^* (current best solution), lb^* (current best lower

bound) and ub^* (current best upper bound) can be stored as global variables and are initialized in the pre-processing stage. The solution tree is explored in a depth-first manner in order to reduce the number of active nodes and ease the method's overall memory requirement. Once a pruning operation is executed, the search resumes from the nearest parent node in the depth search tree.

Most of the method's computational time is spent on solving the linear relaxation problems at each node. Although the lower bounds are computed more efficiently, the use of such methods can introduce a considerable overhead of time. An alternative to overcome this issue is to use the set of dual heuristics to produce lower bounds in a faster manner through the Dual Ascent procedure. More generally, the time factor is sacrificed in order to get better bounds in branch-and-cut methods, while the quality factor of such bounds can be sacrificed in the so called *branch-and-ascent* methods.

Algorithm 11: Branch-and-cut method

input : The instance of the problem I.
The best known primal solution P^*

output: The optimal integer solution x^*

1 $x^* \leftarrow P^*$;
2 $\{I^*, lb^*, ub^*\} \leftarrow Pre\text{-}process(I, x^*)$;
3 $lp \leftarrow CreateLP(I^*)$;
4 $Solve(lp)$;
5 return x^*;

The proposed branch-and-cut method closes the set of algorithms which relies on the mathematical formulations of the MSFBC problem and its heuristics. Sections 4.5 and 4.6 presents a metaheuristic algorithm which takes a completely different approach for the problem and is capable to produce feasible solutions in a much faster and straightforward manner.

4.5 Iterated Local Search Heuristic

Metaheuristic algorithms [35] represents a class of high-level heuristic methods designed to generate good quality solutions for an optimization problem. Although the metaheuristic approach does not guarantee global optimality, it can often find good or even near optimal solutions with much less computational effort than exact methods. Many metaheuristics algorithms uses a stochastic optimization routine, generating sets of random variables

and behaviours to search for the global optima. Different strategies are used to explore the search space, in such a way that there is a balance between the exploitation of the accumulated search experience and the exploration of the search space. According to *Blum et. al.* in [36], this balance is necessary in order to quickly identify potential regions with high quality candidate solutions and leave those which are far from the global optima. A common strategy used to achieve such goals is to iteratively apply a *local search* procedure in combination with a *diversification* mechanism in order to explore a wide variety of solutions.

Algorithm 12: Branch-and-cut method - Solve procedure

input : The current lp formulation *lp*

```
1  x ← Solve_lp(lp);
2  lb ← v( x );
3  ub ← Primal_heuristic(x);
4  gap = ub - lb;
5  Update lb* and ub*, accordingly.
6  if gap = 0 then
7  |  if x is better than x* then
8  |  |  x* ← x;
9  |  end
10 end
11 if lb < ub* then
12 |  x_s ← SelectUnfixedVariable(lp, x);
13 |  s1 ← lp + { x_s = 1 };
14 |  s2 ← lp + { x_s = 0 };
15 |  Solve(s1);
16 |  Solve(s2);
17 end
```

Local search algorithms move from candidate to candidate solutions in the search space by applying local changes until a better solution is found in a pre-fixed number of moves or an elapsed time bound. The concept of *neighbourhood* is essential for any local search procedure: a neighbourhood defines, for every candidate solution s', a set of additional candidate solutions that shares much of the characteristics found in s'. Typically, neighbour solutions are obtained by applying small modifications in s', where the nature of such modifications depends on the specific neighbourhood. If no improved solution is found during a local search routine, then the current best solution found is said to be a neighbourhood's local optimum. To escape such local minimum, each metaheuristic method implements different sets of techniques. One common strategy is to apply a series of perturbation steps to the

current solution, thus resuming the search from a new starting point. This behaviour is illustrated in Figure 4.5. Sections 4.5.1 and 4.5.2 proposes a set of neighbourhoods and an iterated local search method, respectively, to obtain good quality solutions for the MSFBCP in a fast and efficient manner.

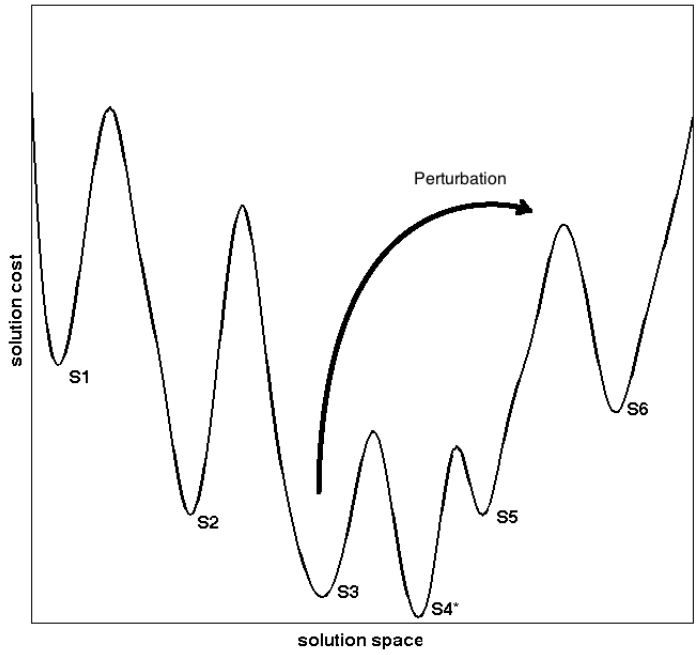


Figure 4.5: *The metaheuristic procedure and solution space*

4.5.1 Local Search and Neighbourhoods

A feasible solution s' for the 2DPU-MSFBC problem describes a set of minimum spanning trees computed over the set of partitions $P_i = (V_i, E_i)$ with a balanced group of vertices or a border point in its vertex set. The exploration of the search space S is done by randomly selecting a pair of partitions $\{P_i, P_j\}$ and applying a series of neighbourhood moves to modify the trees, aiming at improving the current solution. The overall procedure is described in Algorithm 20. Each one of the following proposed neighbourhoods tries different moves in the vertex and/or edge sets of the partitions, limited by a maximum distance radius between vertices from P_i and P_j computed in a pre-processing stage. The following subsections proposes seven neighbourhoods for the local search procedure, detailing the types of moves applied and the computational complexity of each routine. For each neighbourhood, consider the following notation: Let $P_i = (V_i, E_i)$ and $P_j = (V_j, E_j)$ be the pair of partitions randomly selected in $G_{s'}$, the graph describing the set of minimum

spanning trees in the solution set s', where $V_i = (R_i \cup B_i)$ represent the union between the residues set R_i and border points B_i in P_i. The cost $w(s')$ of the solution s' is defined by:

$$w(s') = \sum_i c(P_i), \tag{4.5.1}$$

where $c(P_i)$ is the cost of the minimum spanning tree computed over the partition P_i in $\mathcal{O}(|E_i| \log |E_i|)$. When a local search move is applied, the cost $w(s')$ is updated by only recomputing the minimum spanning trees for P_i and P_j. Each tree is connected to its closest border point if, and only if, there is an unbalanced number of positive and negative residues in its vertex set.

4.5.1.1 Relocate

The *Relocate* neighbourhood consists in relocating vertices from P_i to P_j, independently of its weight, as shown by Algorithm 13. The neighbourhood tries, in the worst case, every possible move between vertices from P_i to P_j, which leads to $\mathcal{O}(|V_i|)$ iterations. Our implementation for the Relocate neighbourhood runs in $\mathcal{O}(|V_i| \, (|E_i| \log |E_i| + |E_j| \log |E_j|))$

Algorithm 13: Relocate - Local Search

```
input  : A feasible solution s'
         The pair of randomly selected partitions P_i and P_j
output: The modified solution s''
1  s'' ← s';
2  initialCost = c(P_i) + c(P_j)
3  foreach vertex v_k in P_i do
4      if the minimum distance between v_k and the vertices of P_j
       exceeds the maximum distance radius for v_k then continue;
5      newCost = c(P_i − {v_k}) + c(P_j + {v_k});
6      if newCost < initialCost then
7          Assign v_k to P_j in s'';
8          initialCost = newCost;
9      end
10 end
11 return s";
```

4.5.1.2 C-Relocate

The *C-Relocate* neighbourhood tries to relocate pairs of positive and negative vertices from P_i to P_j, as shown by Algorithm 14. The neighbourhood tries, in the worst case, every possible move between close pairs of vertices from

P_i to P_j, leading to $\mathcal{O}(|V_i|^2)$ iterations. Our implementation for the C-Relocate neighbourhood runs in $\mathcal{O}(|V_i|^2\,(|E_i|\log|E_i| + |E_j|\log|E_j|))$

Algorithm 14: C-Relocate - Local Search

input : A feasible solution s'
The pair of randomly selected partitions P_i and P_j
output: The modified solution s''

$s'' \leftarrow s'$;
$initialCost = c(P_i) + c(P_j)$
foreach *pair of positive-negative vertices* v_k^+ *and* v_l^- *in* P_i **do**
  if *the minimum distance between* v_k^+, v_l^- *and the vertices from* P_j *exceed their maximum distance radius* **then** continue;
  $newCost = c(P_i - \{v_k^+, v_l^-\}) + c(P_j + \{v_k^+, v_l^-\})$;
  if $newCost < initialCost$ **then**
    Assign v_k^+ and v_k^- to P_j in s'';
    $initialCost = newCost$;
  end
end
return s";

4.5.1.3 Swap

The *Swap* neighbourhood consists in swapping vertices with same weight between P_i and P_j, as shown by Algorithm 13. The neighbourhood tries, in the worst case, every possible swap between vertices from P_i and P_j, leading to $\mathcal{O}(|V_i||V_j|)$ iterations. Our implementation for the Swap neighbourhood runs in $\mathcal{O}(|V_i||V_j|\,(|E_i|\log|E_i| + |E_j|\log|E_j|))$

4.5.1.4 C-Swap

Just like the C-Relocate, the *C-Swap* neighbourhood consists in swapping pairs of positive and negative vertices between P_i and P_j, as shown by Algorithm 16. In order to ease the computational effort required for such neighbourhood, the move is only applied for a pre-fixed number of closest pairs of vertices in P_i and P_j, determined in a pre-processing stage. Our implementation for the C-Swap neighbourhood runs in $\mathcal{O}(|V_i||V_j|\,(|E_i|\log|E_i| + |E_j|\log|E_j|))$

Algorithm 15: Swap - Local Search

input : A feasible solution s'
The pair of randomly selected partitions P_i and P_j
output: The modified solution s''

$s'' \leftarrow s'$;
$initialCost = c(P_i) + c(P_j)$
foreach *vertex* v_k *in* P_i **do**
  foreach *vertex* v_l *in* P_j *with the same weigth of* v_k **do**
    if *the distance between* v_k *and* v_l *exceed their maximum distance radius* **then** continue;
    $newCost = c(P_i - \{v_k\} + \{v_l\}) + c(P_j - \{v_l\} + \{v_k\})$;
    if $newCost < initialCost$ **then**
      Swap v_k and v_l in s'';
      $initialCost = newCost$;
    end
  end
end
return s";

Algorithm 16: C-Swap - Local Search

input : A feasible solution s'
The pair of randomly selected partitions P_i and P_j
output: The modified solution s''

$s'' \leftarrow s'$;
$initialCost = c(P_i) + c(P_j)$;
foreach *close pairs of positive-negative vertices* v_k^+ *and* v_l^- *in* P_i **do**
  foreach *close pairs of positive-negative vertices* v'^+_k *and* v'^-_l *in* P_j **do**
    if *the minimum distance between* $\{v_k^+, v_l^-\}$ *and* $\{v'^+_k, v'^-_l\}$ *exceeds their maximum distance radius* **then** continue;
    $newCost = c(P_i - \{v_k^+, v_l^-\} + \{v_k'^+, v_l'^-\}) + c(P_j - \{v_k'^+, v_l'^-\} + \{v_k^+, v_l^-\})$;
    if $newCost < initialCost$ **then**
      Swap $\{v_k^+, v_l^-\}$ and $\{v_k'^+, v_l'^-\}$ in s'';
      $initialCost = newCost$;
    end
  end
end
return s";

4.5.1.5 Merge

The Merge neighbourhood consists in merging two partitions $P_i = (V_i, E_i)$ and $P_j = (V_j, E_j)$ into a single partition $P_m = (V_m, E_m)$, where $V_m = \{V_i$

$\cup\ V_j\}$ and $E_m = \{E_i \cup E_j \cup E_{ij}\}$, where E_{ij} is the set of edges connecting V_i to V_j. The cost of the new partition P_m is obtained by computing the related minimum spanning tree, as shown by Algorithm 17. Our implementation for the Merge neighbourhood runs in $\mathcal{O}(\big|E_i + E_j\big| \log\big|E_i + E_j\big|)$.

Algorithm 17: Merge - Local Search

input : A feasible solution s'
The pair of randomly selected partitions P_i and P_j
output: The modified solution s''

```
1 s'' ← s';
2 initialCost = c(P_i) + c(P_j);
3 if at least one distance between vertices from P_i and P_j doesn't
  exceed their maximum distance radius then
4 |   P_m ← P_i + P_j;
5 |   newCost = c(P_m);
6 |   if newCost < initialCost then Replace {P_i, P_j} by P_m in s'';
7 |   return s";
8 end
```

4.5.1.6 Break

The *Break* neighbourhood tries to break edges in the minimum spanning tree of a given partition P_i to generate two optmized disjoint sets, P_i' and P_i'', as shown by Algorithm 18. Our implementation for the Break neighbourhood runs in $\mathcal{O}(|V_i||E_i| \log|E_i|)$.

4.5.1.7 Insert 1, Break 1

The *Insert 1, Break 1* neighbourhood is described by two steps: First, the two randomly selected partitions P_i and P_j are merged together and the minimum spanning tree of the merged component is computed. The second step is to break the longest edge in the obtained tree, in such a way that two new partitions are formed, P_i' and P_j', as shown by Algorithm 19. The neighbourhood tries to recombine close trees in such a way that vertices are better clustered in close regions of the instance. Our implementation runs in $\mathcal{O}(\big|E_i + E_j\big| \log\big|E_i + E_j\big|)$.

4.5.2 Iterated Local Search

Iterated local search (ILS, [37]) is a simple local search metaheuristic procedure that iteratively applies perturbation steps in order to escape the

Algorithm 18: Break - Local Search

input : A feasible solution s'
A randomly selected partition P_i
output: The modified solution s''

```
1  s'' ← s';
2  initialCost = c(P_i);
3  foreach edge e in the minimum spanning tree T of P_i do
4  |   Obtain {P_i', P_i''} by breaking the edge e in T;
5  |   newCost = c(P'_i) + c(P''_i);
6  |   if newCost < initialCost then
7  |   |   Replace P_i by {P'_i, P''_i} in s'';
8  |   |   break;
9  |   end
10 end
11 return s";
```

Algorithm 19: Insert 1, Break 1 - Local Search

input : A feasible solution s'
The pair of randomly selected partitions P_i and P_j
output: The modified solution s''

```
1  s'' ← s';
2  initialCost = c(P_i) + c(P_j);
3  if at least one distance between vertices from P_i and P_j does not
   exceed their maximum distance radius then
4  |   P_m ← P_i + P_j;
5  |   T ← MinimumSpanningTree(P_m);
6  |   Obtain {P_i', P_j'} by breaking the longest edge in T;
7  |   newCost = c(P'_i) + c(P''_i);
8  |   if newCost < initialCost then
9  |   |   Replace {P_i, P_j} by {P'_i, P'_j} in s'';
10 |   end
11 end
12 return s";
```


local minimum of a current search point. The ILS algorithm is described by a set of four component procedures: *GenerateInitialSolution*: generates a starting point for the walk in the space of solutions; *Shake*: generates new starting points for the local search by perturbing the current solution; *AcceptanceCriterion*: decides from which solution the walk will be continued; and *LocalSearch*: implements a set of neighbourhoods, as discussed in the previous section. The pseudo-code for the ILS algorithm is shown by Algorithm 21.

GenerateInitialSolution: The initial solution for a partition based ILS algorithm is often generated by creating a fixed set of partitions and ran-

Algorithm 20: Local Search

input : An initial solution s
output: The modified solution s'

```
1  s' ← s;
2  foreach pair of partitions P_i and P_j ∈ s selected in random order do
3      s'_1 ← Relocate(s', P_i, P_j);
4      s'_2 ← C-Relocate(s', P_i, P_j);
5      s'_3 ← Swap(s', P_i, P_j);
6      s'_4 ← C-Swap(s', P_i, P_j);
7      s'_5 ← Merge(s', P_i, P_j);
8      s'_6 ← Break(s', P_i);
9      s'_7 ← Break(s', P_j);
10     s'_8 ← Insert1-Break1(s', P_i, P_j);
11     s'_m ← arg min_{s'_i}(w_{s'_i});
12     if w_{s'_m} < w_{s'} then s' ← s_m;
13 end
14 return s';
```

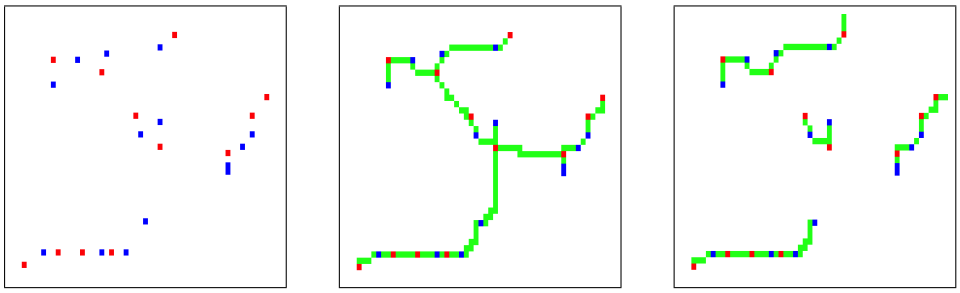

domly assigning each object to one partition at a time. In the 2DPU-MSFBC problem, the initial solution can be constructed by distributing the residues through a fixed set of trees. However, since we are dealing with the two dimensional Euclidean space in the phase unwrapping problem, such random assignment of residues is most likely to produce very bad solutions, even for an initial step. As we are searching for a minimum spanning forest, it is reasonable to initiate the algorithm by computing a minimum spanning tree for all residues and disconnecting edges which exceeds a pre-fixed or a dynamic distance parameter, creating a set of balanced and unbalanced trees. The vertex set for each tree is then described by the union between the set of residues and its closest border points; if a particular tree is unbalanced, one of its residues must be connected to the nearest border point in the image, thus balancing the tree. Most of the balanced trees with no border point are likely to be preserved in the best solution computed by the ILS algorithm, since close sets of balanced residues will be clustered efficiently by the minimum spanning tree. The procedure is shown in Figure 4.6.

Local Search: The previously proposed neighbourhoods are applied for every pair of partitions of an initial given solution, selected in random order. In order to reduce the method's computational effort required, a series of auxiliary tables and structures keep track of the modified trees and moves applied. Moves already evaluated on a pair of unmodified trees are avoided in a next local search iteration.

Algorithm 21: Iterated Local Search

input : The graph $\overline{G} = (\overline{V}, \overline{E_1})$
output: A set of balanced trees S

```
1  It_shake ← 0;
2  S ← GenerateInitialSolution(Ḡ);
3  S* ← S;
4  while It_shake < It_MAX do
5      S ← LocalSearch(S);
6      if w(S) < w(S*) then
7          S* ← S;
8          It_shake ← 0;
9      end
10     if w(S) == w(S*) then Shake(S*);
11     else
12         S ← Shake(S) or S ← Shake(S*) with 50% chance each;
13     end
14     It_shake++;
15 end
16 return S*;
```



4.6(a): 4.6(b): 4.6(c):

Figure 4.6: *Generating the initial solution for the ILS algorithm. (a) The instance with 11 negative residues and 14 positive residues. (b) The minimum spanning tree considering all residues in the image. (c) The resulting spanning forest after disconnecting long edges.*

Shake: The shake procedure randomly modifies the current local minimum solution in order to generate new starting points for the local search routine. The procedure starts by fixing a random number k of edges to be disconnected, thus creating $2k$ new disjoint trees. The trees are then randomly recombined and merged together, with no distance limitation. The value of k is fixed in such a way that the new initial solution shares most of the characteristics found in the previous local minimum.

Acceptance Criterion: After a local search routine, the method verifies if the new local minimum found is better than the best solution found so far.

If so, it updates the value of the best solution and the method continues. If no improvement is made on a previously known best solution, the shake procedure is called. If the current local minimum is equal to the best solution found so far, S^*, then the shake procedure is called for S^*. On the other hand, in the case where the current local minimum S' is worse than S^*, the shake procedure is called for either S' or S^*, with 50% of chance each. This allows a good balance between aggressive solution improvement and exploration of the search space.

The described ILS algorithm is executed in a pre-fixed number of iterations without improving the best solution found. Section 4.6 presents a set covering formulation which will lead to an ILS hybrid metaheuristic approach.

4.6 Set Covering Formulation and Hybrid Metaheuristic

The mathematical formulation proposed in Section 4.1, whose goal is to find an optimal set of edges which satisfies the cut based constraints, is just one possible edge based formulation for the MSFBCP. It can also be interpreted, for instance, as a minimum cost multi-flow problem on graphs, where the difference between the flow that enters and leaves a given vertex is equal to its weight. Based on the metaheuristic's approach of finding an optimal set of balanced partitions of vertices and edges, we can also formulate the MSFBC as a set covering problem.

The set covering problem [38] is a well-known optimization problem whose study has led to the development of fundamental techniques for the field of approximation algorithms, as stated by *Vazirani* in [39]. Given a set of elements $U = \{1,2,3,...,n\}$ and a set S of subsets of U, the set covering problem seeks to find the smallest subset of S whose union is equal to U. The problem is written as an integer linear optimization program, as follows:

$$\min \sum_{s \in S} x_s \tag{4.6.1}$$

$$\textbf{\textit{s.t.}} \sum_{s:e \in s} x_s \geq 1, \ \ \forall e \in U \tag{4.6.2}$$

$$x_s \in \{0,1\}, \forall s \in S, \tag{4.6.3}$$

where set of constraints represented by Equation 4.6.2 assures that every element $e \in U$ is covered by at least one set $s \subset S$. If we interpret the set U as a set of weighted vertices $v_i \in V$ and S as a set of balanced partitions P of V, the MSFBCP can be written in the following set covering formulation:

$$\min \sum_{p \in P} c_p x_p \tag{4.6.4}$$

$$\textbf{\textit{s.t.}} \sum_{p \in P} a_{vp} x_p = 1, \ \ \forall v \in R \tag{4.6.5}$$

$$x_p \in \{0, 1\}, \forall p \in P, \tag{4.6.6}$$

$$a_{vp} = \begin{cases} 1 & \text{if } v \in p \\ 0 & \text{if } v \notin p \end{cases} \tag{4.6.7}$$

where c_p denotes the cost of the minimum spanning tree computed over the partition p. The objective is to find the minimum cost set of balanced partitions whose union is equal to the universe U. This formulation can be used to design a hybrid metaheuristic algorithm which regularly solves restricted set covering problems in an attempt to improve its solutions.

Hybrid metaheuristics methods [40] often combines different algorithmic components from a variety of research areas in optimization, such as mathematical programming and machine learning. The strategy aims to achieve a greater performance in solving hard optimization problems. Components of a hybrid metaheuristic may run concurrently and exchange information to guide the search for the global optima.

The hybridization of the proposed ILS algorithm can be achieved by solving the set covering formulation of the MSFBCP at every pre-fixed number of iterations, using a subset of possible partitions P_i extracted from every local minimum found by the metaheuristic. However, since the set covering problem is NP-*hard* [41], the computational effort required for finding its optimal solution grows exponentially with the size of the instance. It is impractical to introduce such overhead of computational time to the metaheuristic method, and therefore, a time bound is established in order to limit the MIP solving procedure.

Algorithm 22 shows the pseudo-code of the hybrid metaheuristic algorithm. The set covering procedure is executed at every It_{SC} iterations (generally, at each third of the shake iterations). The partition set P is populated by including every balanced subset $p \in S$ at each local minimum. Chapter 5 discusses the improvements and results obtained by the hybrid metaheuristic method over the non hybrid ILS algorithm proposed in Section 5.5.

Algorithm 22: Hybrid Iterated Local Search

input : The graph $\overline{G} = (\overline{V}, \overline{E_1})$
output: A set of balanced trees S

```
1  S ← GenerateInitialSolution(Ḡ);
2  It_shake ← 0;
3  S ← I;
4  S* ← S;
5  P ← {};
6  while It_shake < It_MAX do
7      S ← LocalSearch(S);
8      if c(S) < c(S*) then
9          S* ← S;
10         It_shake ← 0;
11     end
12     foreach It_SC iterations do
13         S ← SetCovering(P)
14         if c(S) < c(S*) then
15             S* ← S;
16             It_shake ← 0;
17         end
18     end
19     foreach p ∈ S do
20         if p ∉ P then P ← P + {p};
21     end
22     S ← Shake(S) or S ← Shake(S*) with 50% chance each;
23     It_shake++;
24 end
25 return S*;
```

5
Computational Experiments

The computational experiments described in this chapter were conceived to: (1) validate and investigate the performance of the proposed methods in a set of designed benchmark instances for the MSFBC problem; (2) evaluate the performance of the MSFBC approach in the two-dimensional phase unwrapping problem, when compared to other path-following methods.

To test and identify the limitations of the proposed methods, we devised a set of instances designed to cover a wide variety of topologies of vertices, simulating the occurrence of noise and natural discontinuities in the 2DPU-MSFBC problem. In addition, these instances were also designed to represent a challenging set for the MSFBC problem and analyse the scalability factor for each method.

The experiments were conducted on an Intel i7 2.3 Ghz processor (6 MB shared level 3 cache) machine with 8 GB of RAM. The codes were written in the C++ language, using the UNIX g++ v4.2.1 compiler on a MAC OSX 10.10.1 64bit operating system. The linear and integer programs were solved using the *GUROBI* 6.0.4 optimization suite [42]. The execution times for each method were obtained using the *getrusage* function from the <*sys/time.h*> and <*sys/resource.h*> standard C++ libraries, measuring the total CPU time used by the process, in seconds.

5.1
Randomized Instances

The instances **PUC_r_p_n_x** were generated by randomly spreading p positive (+1) and n negative (-1) vertices on an $4p \times 4n$ Euclidean space. The cost *c(i,j)* for each edge (i,j) is defined by the 2D Euclidean distance between vertices i and j. Additionally, every vertex v is connected to a single border point b, where the cost *c(v,b)* represents the 2D Euclidean distance between v and its closest border point. There are 21 sets of 5 instances each, starting from 8 to 1024 nodes. Since there is no reference solutions in the literature, we have collected the best solutions ever found during the heuristics and exact methods in order to evaluate the quality of each proposed algorithm. These solution values are represented in each table by the column **BKS**. A solution is marked with a ***** when it refers to the optimal solution for that instance, as

proven by the branch-and-cut method. Solution values in bold letters indicates the best known primal solution.

5.1.1 Hybrid Metaheuristic

The hybrid ILS algorithm was executed 10 times with two termination criteria per run, whichever came first: (1) 100 iterations ($It_{MAX} = 100$) without improving the best solution found; (2) A time bound of 3600 seconds. The set covering routine is executed at every $(1/3)It_{MAX}$ iterations, with a time bound of 300 seconds. The maximum distance radius for every vertex v is limited to 25% of the shortest distances between v and the set of vertices V-$\{v\}$. The choice of such parameters was made based on a preliminary calibration of the method through a series of tests.

Tables 5.1, 5.2 and 5.3 present a summary of the experimental results. The best solution reported $\mathbf{S}_{best}$ refers to the global best solution found, while the average solution $\mathbf{S}^*_{avg\_10}$ represents the average solution found on 10 runs. Columns $\mathbf{GAP}_{best}$ **(%)** and $\mathbf{GAP}_{avg}$ **(%)** represents, respectively: (1) gap, in percentage from the best solution found during the 10 runs to the **BKS**; (2) gap in percentage from the average solution from the 10 runs to the **BKS**. Finally, the column **T(s)** reports the average time per run, measured in seconds.

Table 5.1: Results for the hybrid ILS algorithm

Instance	$\|V\|$	$\|E\|$	$\mathbf{S}_{best}$	$\mathbf{GAP}_{best}$(%)	$\mathbf{S}_{avg_10}$	$\mathbf{GAP}_{avg}$(%)	**BKS**	**T(s)**
PUC_r_4_4_1	8	72	**24.705**	0	**24.705**	0	24.705*	0.4
PUC_r_4_4_2	8	72	**16.456**	0	**16.456**	0	16.456*	0.25
PUC_r_4_4_3	8	72	**14.261**	0	**14.261**	0	14.261*	0.3
PUC_r_4_4_4	8	72	**24.991**	0	**24.991**	0	24.991*	0.23
PUC_r_4_4_5	8	72	**23.895**	0	27.29	12.437	23.895*	0.15
PUC_r_6_6_1	12	156	**54.542**	0	**54.542**	0	54.542*	0.86
PUC_r_6_6_2	12	156	**49.049**	0	**49.049**	0	49.049*	0.67
PUC_r_6_6_3	12	156	**53.592**	0	**53.592**	0	53.592*	0.72
PUC_r_6_6_4	12	156	**49.488**	0	**49.488**	0	49.488*	0.94
PUC_r_6_6_5	12	156	**46.935**	0	**46.935**	0	46.935*	0.66
PUC_r_8_8_1	16	272	**79.382**	0	**79.382**	0	79.382*	1.46
PUC_r_8_8_2	16	272	**69.676**	0	**69.676**	0	69.676*	1.46
PUC_r_8_8_3	16	272	**82.496**	0	82.86	0.439	82.496*	1.57
PUC_r_8_8_4	16	272	**79.012**	0	**79.012**	0	79.012*	1.46
PUC_r_8_8_5	16	272	**79.603**	0	81.381	2.185	79.603*	2.05
PUC_r_10_10_1	20	420	**128.158**	0	128.209	0.04	128.158*	2.15
PUC_r_10_10_2	20	420	**132.932**	0	**132.932**	0	132.932*	3.73
PUC_r_10_10_3	20	420	**112.172**	0	**112.172**	0	112.172*	3.59
PUC_r_10_10_4	20	420	**119.738**	0	120.97	1.019	119.738*	3.88
PUC_r_10_10_5	20	420	**121.932**	0	**121.932**	0	121.932*	4.64

Table 5.2: Results for the hybrid ILS algorithm (continued)

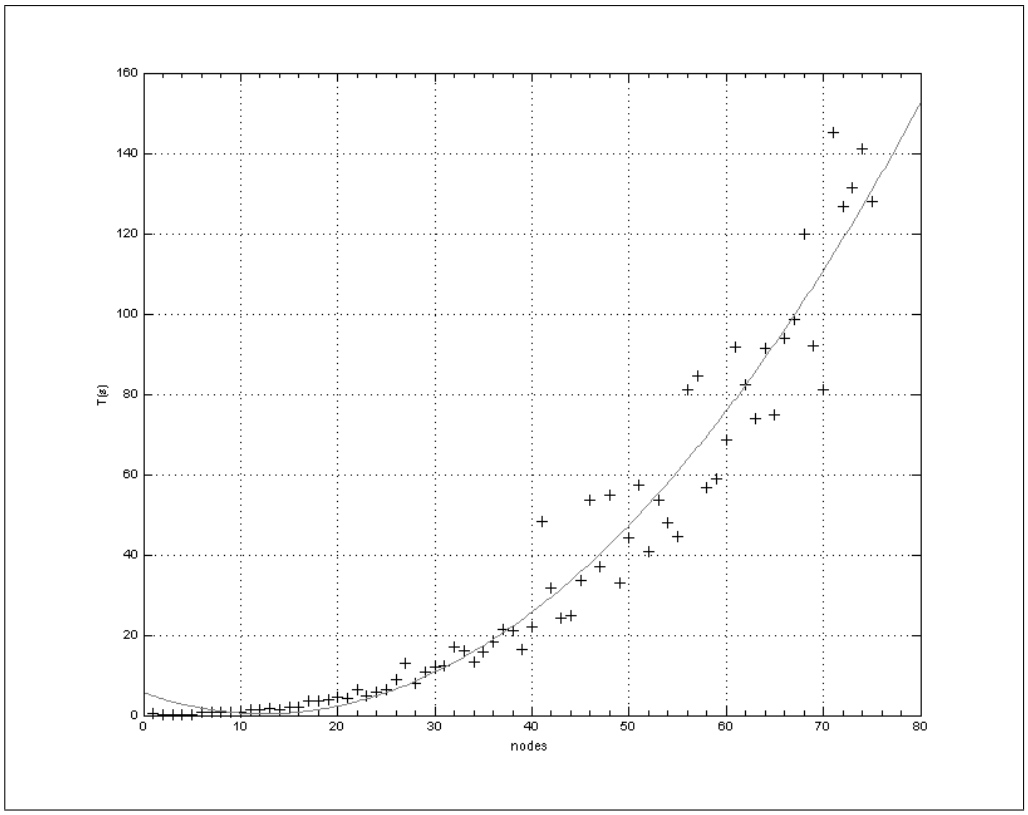
Instance	$\|V\|$	$\|E\|$	$\mathbf{S}_{best}$	$\mathbf{GAP}_{best}(\%)$	$\mathbf{S}_{avg_{10}}$	$\mathbf{GAP}_{avg}(\%)$	**BKS**	**T(s)**
PUC_r_12_12_1	24	600	**183.245**	0	183.972	0.395	183.245*	4.15
PUC_r_12_12_2	24	600	**124.074**	0	125.962	1.499	124.074*	6.3
PUC_r_12_12_3	24	600	**157.608**	0	157.649	0.026	157.608*	4.81
PUC_r_12_12_4	24	600	**163.066**	0	163.507	0.269	163.066*	5.67
PUC_r_12_12_5	24	600	**168.411**	0	170.164	1.03	168.411*	6.36
PUC_r_14_14_1	28	812	**203.863**	0	208.371	2.164	203.863*	8.87
PUC_r_14_14_2	28	812	**209.821**	0	215.034	2.424	209.821*	12.87
PUC_r_14_14_3	28	812	**215.686**	0	**215.686**	0	215.686*	7.89
PUC_r_14_14_4	28	812	**207.88**	0	208.135	0.123	207.88*	10.82
PUC_r_14_14_5	28	812	**209.477**	0	210.693	0.577	209.477*	12.04
PUC_r_16_16_1	32	1056	**259.67**	0	**259.67**	0	259.67*	12.33
PUC_r_16_16_2	32	1056	**237.027**	0	237.268	0.101	237.027*	16.99
PUC_r_16_16_3	32	1056	**222.041**	0	222.561	0.234	222.041*	16.07
PUC_r_16_16_4	32	1056	**234.291**	0	237.363	1.294	234.291*	13.36
PUC_r_16_16_5	32	1056	**284.184**	0	290.457	2.159	284.184*	15.66
PUC_r_18_18_1	36	1332	**297.529**	0	297.703	0.058	297.529*	18.39
PUC_r_18_18_2	36	1332	**302.638**	0	303.973	0.439	302.638*	21.36
PUC_r_18_18_3	36	1332	**291.269**	0	292.308	0.356	291.269*	21.21
PUC_r_18_18_4	36	1332	**286.077**	0	296.15	3.401	286.077*	16.51
PUC_r_18_18_5	36	1332	**263.764**	0	264.963	0.453	263.764*	22.1
PUC_r_20_20_1	40	1640	**347.491**	0	351.7	1.197	347.491*	48.26
PUC_r_20_20_2	40	1640	**381.249**	0	**381.249**	0	381.249*	31.67
PUC_r_20_20_3	40	1640	**391.299**	0	394.761	0.877	391.299*	24.12
PUC_r_20_20_4	40	1640	**399.453**	0	402.28	0.703	399.453*	24.86
PUC_r_20_20_5	40	1640	**366.812**	0	368.188	0.374	366.812*	33.52
PUC_r_22_22_1	44	1980	462.137	0.677	473.896	3.141	459.009*	53.56
PUC_r_22_22_2	44	1980	**413.547**	0	417.59	0.968	413.547*	36.88
PUC_r_22_22_3	44	1980	**433.645**	0	439.33	1.294	433.645*	54.8
PUC_r_22_22_4	44	1980	**461.945**	0	464.014	0.446	461.945*	33.06
PUC_r_22_22_5	44	1980	**448.584**	0	454.663	1.337	448.584*	44.2
PUC_r_24_24_1	48	2352	**460.297**	0	464.82	0.973	460.297*	57.49
PUC_r_24_24_2	48	2352	**477.413**	0	480.902	0.726	477.413*	40.9
PUC_r_24_24_3	48	2352	**418.466**	0	421.274	0.667	418.466*	53.59
PUC_r_24_24_4	48	2352	**459.073**	0	463.974	1.056	459.073*	48.02
PUC_r_24_24_5	48	2352	**480.623**	0	481.823	0.249	480.623*	44.56
PUC_r_26_26_1	52	2756	**573.996**	0	584.115	1.732	573.996*	81.22
PUC_r_26_26_2	52	2756	**515.262**	0	526.523	2.139	515.262*	84.54
PUC_r_26_26_3	52	2756	**594.353**	0	595.504	0.193	594.353*	56.76
PUC_r_26_26_4	52	2756	**472.589**	0	475.033	0.515	472.589*	58.93
PUC_r_26_26_5	52	2756	**585.796**	0	598.289	2.088	585.796*	68.72
PUC_r_28_28_1	56	3192	**640.927**	0	657.444	2.512	640.927*	91.59
PUC_r_28_28_2	56	3192	**631.737**	0	636.723	0.783	631.737*	82.25
PUC_r_28_28_3	56	3192	**594.231**	0	594.43	0.034	594.231*	73.91
PUC_r_28_28_4	56	3192	**570.737**	0	576.155	0.94	570.737*	91.29
PUC_r_28_28_5	56	3192	**544.909**	0	558.627	2.456	544.909*	74.74
PUC_r_30_30_1	60	3660	**639.401**	0	648.903	1.464	639.401*	93.92
PUC_r_30_30_2	60	3660	**722.468**	0	736.346	1.885	722.468*	98.7
PUC_r_30_30_3	60	3660	**732.902**	0	746.475	1.818	732.902*	119.74
PUC_r_30_30_4	60	3660	**668.177**	0	669.945	0.264	668.177*	92.07
PUC_r_30_30_5	60	3660	**706.65**	0	709.074	0.342	706.65*	81.2

Table 5.3: Results for the hybrid ILS algorithm (continued)

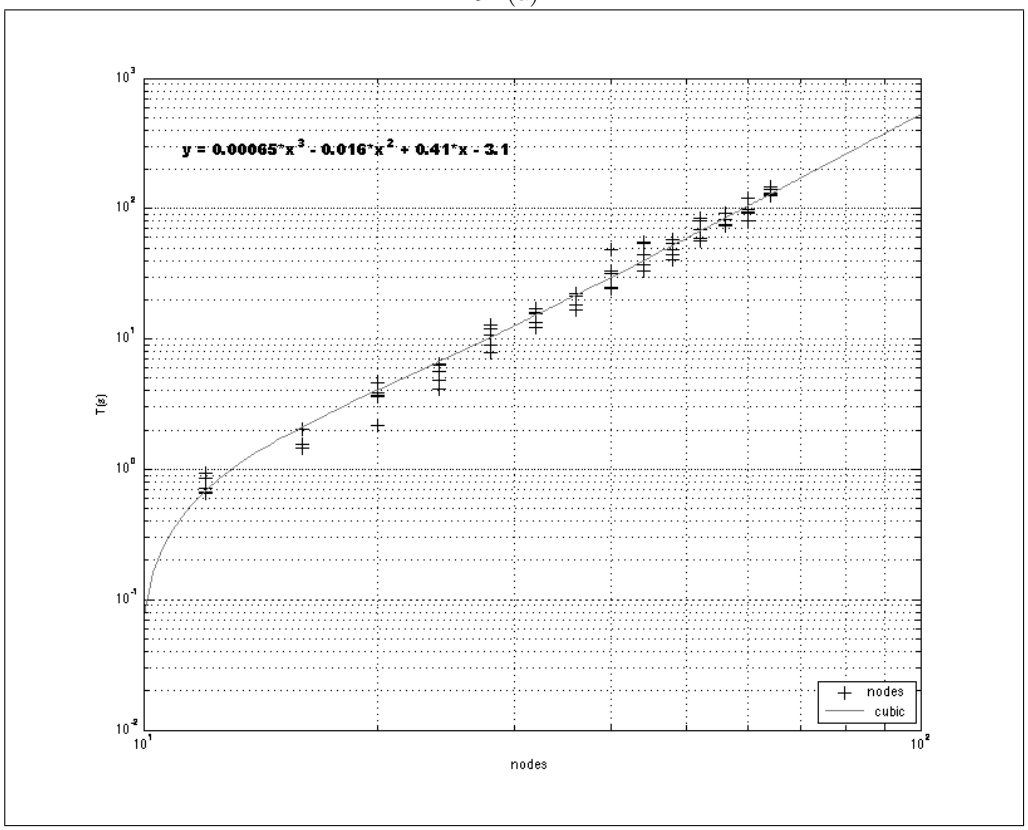
Instance	$\|V\|$	$\|E\|$	$\mathbf{S}_{best}$	$\mathbf{GAP}_{best}(\%)$	$\mathbf{S}_{avg_10}$	$\mathbf{GAP}_{avg}(\%)$	BKS	T(s)
PUC_r_32_32_1	64	4160	802.768	1.736	834.173	5.435	788.832*	145.21
PUC_r_32_32_2	64	4160	**748.656**	0	797.036	6.07	748.656*	126.61
PUC_r_32_32_3	64	4160	**754.849**	0	757.459	0.345	754.849*	131.41
PUC_r_32_32_4	64	4160	**805.771**	0	825.498	2.39	805.771*	141.08
PUC_r_32_32_5	64	4160	**789.849**	0	796.734	0.864	789.849*	127.95
PUC_r_40_40_1	80	6480	1089.771	1.187	1102.816	2.356	1076.832*	280.25
PUC_r_40_40_2	80	6480	1084.382	0.791	1124.809	4.357	1075.805*	295.59
PUC_r_40_40_3	80	6480	**1048.912**	0	1079.111	2.799	1048.912*	266.89
PUC_r_40_40_4	80	6480	**1114.463**	0	1135.72	1.872	1114.463*	329.72
PUC_r_40_40_5	80	6480	**1122.355**	0	1180.169	4.899	1122.355*	350.77
PUC_r_48_48_1	96	9312	**1348.944**	0	1421.624	5.112	1348.944*	643.02
PUC_r_48_48_2	96	9312	**1613.758**	0	1698.21	4.973	1613.758	892.48
PUC_r_48_48_3	96	9312	**1511.631**	0	1569.205	3.669	1511.631	514.21
PUC_r_48_48_4	96	9312	1495.145	0.51	1583.952	6.088	1487.522	706.41
PUC_r_48_48_5	96	9312	**1422.952**	0	1485.311	4.198	1422.952*	494.14
PUC_r_64_64_1	128	16512	2450.873	1.023	2579.993	5.977	2425.795	2482.93
PUC_r_64_64_2	128	16512	**2166.769**	0	2322.014	6.686	2166.769*	1099.27
PUC_r_64_64_3	128	16512	2576.421	4.219	2624.889	5.988	2467.719	3348.55
PUC_r_64_64_4	128	16512	**2293.612**	0	**2293.612**	0	2293.612	1735.7
PUC_r_64_64_5	128	16512	**2269.257**	0	2482.121	8.576	2269.257*	1791.81
PUC_r_128_128_1	256	65792	**7061.173**	0	7643.75	7.622	7061.173	3600
PUC_r_128_128_2	256	65792	**7748.266**	0	8008.762	3.253	7748.266	3600
PUC_r_128_128_3	256	65792	**7518.65**	0	7824.093	3.904	7518.65	3600
PUC_r_128_128_4	256	65792	7530.24	0.001	7539.174	0.119	7530.174	3600
PUC_r_128_128_5	256	65792	**6913.376**	0	8038.889	14.001	6913.376	3600
PUC_r_256_256_1	512	262656	**23517.256**	0	26140.367	10.035	23517.256	3600
PUC_r_256_256_2	512	262656	**23286.799**	0	23978.253	2.884	23286.799	3600
PUC_r_256_256_3	512	262656	**22874.346**	0	25420.154	10.015	22874.346	3600
PUC_r_256_256_4	512	262656	**23442.233**	0	24726.646	5.194	23442.233	3600
PUC_r_256_256_5	512	262656	**22863.504**	0	23672.328	3.417	22863.504	3600
PUC_r_512_512_1	1024	1048576	71011.166	2.091	71011.166	2.091	69526.195	3600
PUC_r_512_512_2	1024	1048576	69510.146	5.886	69890.542	6.399	65418.516	3600
PUC_r_512_512_3	1024	1048576	69401.363	5.401	69523.126	5.567	65652.68	3600
PUC_r_512_512_4	1024	1048576	68040.88	2.595	69141.385	4.145	66275.43	3600
PUC_r_512_512_5	1024	1048576	69884.32	4.016	71384.11	6.033	67077.516	3600

The method was able to find the optimal solution from 83 out of 105 instances, meaning a success rate of almost 80%. In addition, the solutions obtained for 12 other instances were considered to be the best known primal solution, with no guarantee on optimality.

The CPU times range from a fraction of a second on the smaller instances up to 30 minutes for problems with 128 nodes. Since the execution time is limited to 3600 seconds, the graphs presented in Figures 5.1(a) and 5.1(b) only reports the results for a range of 8 to 80 nodes. The growth of CPU time is represented as a function of the number of nodes in the linear and the log-log scales. The fitted curve presented in Figure 5.1(b) suggests that the CPU time is rising in $\mathcal{O}(n^3)$.



5.1(a):



5.1(b):

Figure 5.1: *The growth of CPU time for the hybrid ILS algorithm. (a) Linear scale. (b) Log-log scale.*

5.1.2 Dual Heuristics

Table 5.4 shows the results for running the Dual Ascent and Dual Scaling algorithms on the PUC set of instances. The selection of violated cuts was tested with two different criteria: (**minrc**) by the minimum reduced cost arc in the graph; (**random**) by randomly selecting a non maximal dual variable and saturating at least one of its arcs.

Each execution is composed by two steps: First, the Dual Ascent and Dual Scaling algorithms are executed to obtain a lower bound. Later, a primal feasible solution is obtained over the dual solution with the Reverse Delete Step (RDS) routine. Columns $\mathbf{GAP}_{mrc\_LB}(\%)$ and $\mathbf{GAP}_{rand\_LB}(\%)$ reports the average gap between the dual bounds and the best known primal solutions, per group of instance. The average gap between the primal bounds using the RDS algorithm and the best known primal solutions are reported in columns $\mathbf{GAP}_{mrc\_BKS}(\%)$ and $\mathbf{GAP}_{rand\_BKS}(\%)$. Finally, columns $\mathbf{T}_{mrc}(\mathbf{s})$ and $\mathbf{T}_{rand}(\mathbf{s})$ reports the average CPU time, in seconds.

Table 5.4: Results for the dual heuristics

Instance	$\mathbf{GAP}_{mrc\_LB}(\%)$	$\mathbf{GAP}_{rand\_LB}(\%)$	$\mathbf{GAP}_{mrc\_BKS}(\%)$	$\mathbf{GAP}_{rand\_BKS}(\%)$	$\mathbf{T}_{mrc}(\mathbf{s})$	$\mathbf{T}_{rand}(\mathbf{s})$
PUC_r_4_4_x	2.098	0.000	6.897	0.000	0.001	0.0003
PUC_r_6_6_x	7.337	0.000	7.399	2.974	0.001	0.0005
PUC_r_8_8_x	2.751	3.559	19.119	14.384	0.003	0.0009
PUC_r_10_10_x	6.464	1.458	13.038	4.506	0.004	0.0015
PUC_r_12_12_x	3.783	3.780	28.575	23.644	0.009	0.0028
PUC_r_14_14_x	8.922	3.345	19.637	11.343	0.010	0.0038
PUC_r_16_16_x	9.198	3.717	25.659	9.898	0.017	0.0051
PUC_r_18_18_x	14.634	7.610	28.747	20.525	0.011	0.0086
PUC_r_20_20_x	13.050	2.960	17.635	11.631	0.024	0.0104
PUC_r_22_22_x	13.821	3.743	25.055	14.032	0.028	0.0132
PUC_r_24_24_x	4.787	3.223	19.594	13.632	0.020	0.0156
PUC_r_26_26_x	11.933	4.216	22.791	8.495	0.031	0.0206
PUC_r_28_28_x	10.804	3.297	21.314	12.326	0.030	0.0271
PUC_r_30_30_x	10.328	3.995	18.934	18.063	0.078	0.0312
PUC_r_32_32_x	11.143	4.152	27.785	17.380	0.082	0.0395
PUC_r_40_40_x	15.946	6.871	24.861	16.862	0.133	0.0735
PUC_r_48_48_x	18.695	9.194	20.873	15.779	0.113	0.1298
PUC_r_64_64_x	16.373	10.288	17.099	16.878	0.360	0.3007
PUC_r_128_128_x	34.818	16.481	17.653	11.204	10.485	2.6413
PUC_r_256_256_x	33.877	18.185	17.938	13.875	53.379	20.0289
PUC_r_512_512_x	40.441	26.155	22.168	13.252	1312.558	169.2926

Although similar methods performed considerably well for the Steiner-tree problem [43], the results showed that they were unable to produce good quality bounds for the PUC instance set of the MSFBCP. Strictly greedy methods showed to be inefficient at producing quality lower and upper bounds. The **random** criterion was able to produce better dual bounds for larger instances in a considerable less amount of time, suggesting that a cut based criterion can be a better approach for the PUC set of instances.

5.1.3 Branch-and-cut

Tables 5.5, 5.6, 5.7 and 5.8 summarizes the computational results obtained by running the branch-and-cut algorithm for every instance. A time limit of 3600 seconds has been set for the resolution. The following information is reported:

- $|\mathbf{E}_R|$: The number of edges after using the best primal and dual bounds to fix arcs by reduced cost in the pre-processing stage.
- $\mathbf{P}_{ILS}$: The best primal solution found by the hybrid ILS algorithm, used as an initial upper bound for the exact method.
- **Root LP**: the LP value obtained after solving the root node in the tree.
- **LB** and **UB**: The best lower and upper bounds obtained.
- **GAP(%)**: The gap, in percentage, between the best lower and upper bounds.
- **N**: Number of nodes explored in the branch-and-bound tree.
- **D**: The maximum depth reached in the solution tree.
- **T(s)**: The CPU time measured in seconds.
- **FS(%)**: The total percentage of time spent to find the unbalanced connected components while solving the linear program.
- **PtPS(%)**: The total percentage of time spent to solve the min-cut/max-flow algorithm for every pair of vertices inside a same connected component while solving the linear program.

Many primal solutions obtained by the ILS method proved to be optimal. 20 out of 105 instances were not solved to optimality, with an average gap of 17% between the best lower and upper bounds. As expected, the separation of cuts by the min-cut/max-flow procedure took more than 50% of the running time in many instances, with an average overhead of 28%. In contrast, the simple detection of unbalanced components using a depth-first-search algorithm showed to introduce an insignificant overhead of time to the solving procedure, with an average overhead of 0.42%. The results also showed that the running times are not totally correlated with the size of the instances, but rather with their topologies. This behaviour is illustrated in Figure 5.2.

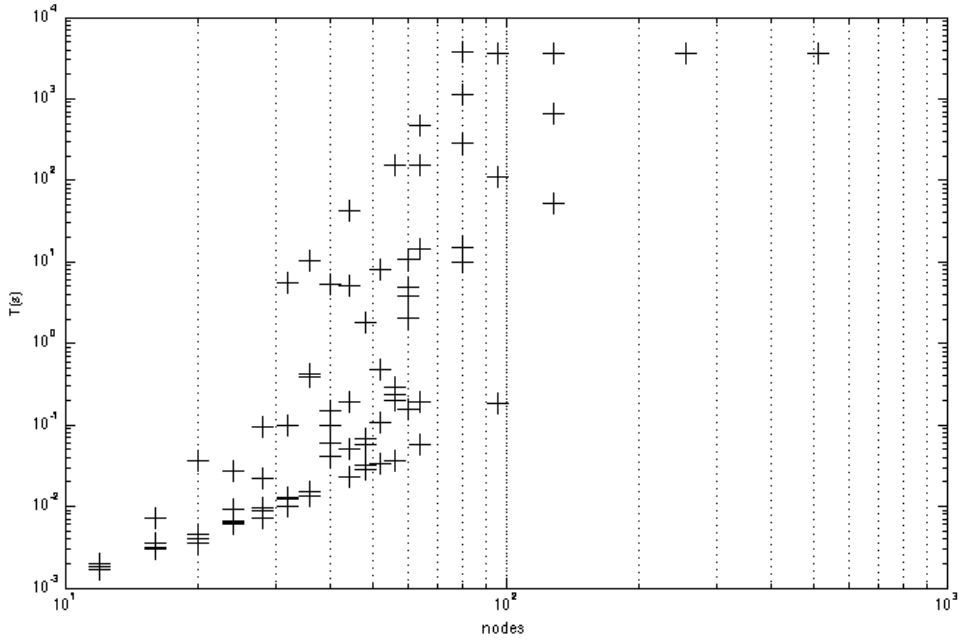


Figure 5.2: *The CPU running times for the branch-and-cut algorithm in the log-log scale.*

Table 5.5: Results for the branch-and-cut algorithm

Instance	**|V|**	**|E|**	$\|\mathbf{E}\|_R$	$\mathbf{P}_{ILS}$	**LP**	**LB**	**UB**	**GAP**	**N**	**D**	**T(s)**	**FS(%)**	**PtPS(%)**
PUC_r_4_4_1	8	72	10	24.705	24.705	24.705	24.705*	0	1	0	0.001	0.350	0.088
PUC_r_4_4_2	8	72	10	16.456	16.456	16.456	16.456*	0	1	0	0.001	0.267	0.000
PUC_r_4_4_3	8	72	9	14.261	14.261	14.261	14.261*	0	1	0	0.001	0.560	2.478
PUC_r_4_4_4	8	72	10	24.991	24.991	24.991	24.991*	0	1	0	0.001	0.356	0.089
PUC_r_4_4_5	8	72	14	23.895	23.895	23.895	23.895*	0	1	0	0.002	0.863	19.700
PUC_r_6_6_1	12	156	12	54.542	54.542	54.542	54.542*	0	1	0	0.002	0.299	0.000
PUC_r_6_6_2	12	156	11	49.049	49.049	49.049	49.049*	0	1	0	0.002	0.300	0.060
PUC_r_6_6_3	12	156	12	53.592	53.592	53.592	53.592*	0	1	0	0.002	0.295	0.000
PUC_r_6_6_4	12	156	13	49.488	49.488	49.488	49.488*	0	1	0	0.002	0.283	0.057
PUC_r_6_6_5	12	156	14	46.935	46.935	46.935	46.935*	0	1	0	0.002	0.448	3.833
PUC_r_8_8_1	16	272	21	79.382	79.382	79.382	79.382*	0	1	0	0.003	0.528	9.851
PUC_r_8_8_2	16	272	15	69.676	69.676	69.676	69.676*	0	1	0	0.003	0.372	5.437
PUC_r_8_8_3	16	272	35	82.496	82.199	82.496	82.496*	0	3	1	0.007	0.755	18.957
PUC_r_8_8_4	16	272	20	79.012	79.012	79.012	79.012*	0	1	0	0.004	0.420	6.807
PUC_r_8_8_5	16	272	25	79.603	79.603	79.603	79.603*	0	1	0	0.003	0.466	7.138
PUC_r_10_10_1	20	420	23	128.158	128.158	128.158	128.158*	0	1	0	0.005	0.307	9.380
PUC_r_10_10_2	20	420	18	132.932	132.932	132.932	132.932*	0	1	0	0.004	0.200	0.000
PUC_r_10_10_3	20	420	19	112.172	112.172	112.172	112.172*	0	1	0	0.004	0.317	4.629
PUC_r_10_10_4	20	420	41	119.738	117.229	119.738	119.738*	0	13	6	0.036	1.038	26.473
PUC_r_10_10_5	20	420	26	121.932	121.932	121.932	121.932*	0	1	0	0.004	0.301	4.443
PUC_r_12_12_1	24	600	36	183.245	183.245	183.245	183.245*	0	1	0	0.006	0.286	11.395
PUC_r_12_12_2	24	600	27	124.074	124.074	124.074	124.074*	0	1	0	0.006	0.230	3.466
PUC_r_12_12_3	24	600	31	157.608	157.608	157.608	157.608*	0	1	0	0.009	0.299	3.083
PUC_r_12_12_4	24	600	31	163.066	163.066	163.066	163.066*	0	1	0	0.007	0.263	6.906
PUC_r_12_12_5	24	600	306	168.411	168.090	168.411	168.411*	0	5	2	0.027	0.643	20.382
PUC_r_14_14_1	28	812	188	203.863	196.663	203.863	203.863*	0	23	5	0.095	0.584	21.861

Table 5.6: Results for the branch-and-cut algorithm (continued)

Instance	**|V|**	**|E|**	$\|\mathbf{E}\|_R$	$\mathbf{P}_{ILS}$	**LP**	**LB**	**UB**	**GAP**	**N**	**D**	**T(s)**	**FS(%)**	**PtPS(%)**
PUC_r_14_14_2	28	812	243	209.821	202.976	209.821	209.821*	0	3	1	0.022	0.576	15.786
PUC_r_14_14_3	28	812	35	215.686	215.686	215.686	215.686*	0	1	0	0.009	0.245	15.244
PUC_r_14_14_4	28	812	31	207.880	207.880	207.880	207.88*	0	1	0	0.007	0.238	3.444
PUC_r_14_14_5	28	812	33	209.477	209.477	209.477	209.477*	0	1	0	0.010	0.176	3.585
PUC_r_16_16_1	32	1056	32	259.670	259.670	259.670	259.67*	0	1	0	0.012	0.163	2.406
PUC_r_16_16_2	32	1056	38	237.027	237.027	237.027	237.027*	0	1	0	0.013	0.398	7.570
PUC_r_16_16_3	32	1056	35	222.041	222.041	222.041	222.041*	0	1	0	0.010	0.269	6.792
PUC_r_16_16_4	32	1056	141	234.291	230.052	234.291	234.291*	0	17	4	0.097	1.014	39.927
PUC_r_16_16_5	32	1056	497	284.184	267.385	284.184	284.184*	0	471	17	5.515	0.333	15.894
PUC_r_18_18_1	36	1332	615	297.529	285.399	297.529	297.529*	0	579	16	10.475	0.464	51.575
PUC_r_18_18_2	36	1332	49	302.638	301.859	302.638	302.638*	0	3	1	0.014	1.104	54.570
PUC_r_18_18_3	36	1332	42	291.269	291.269	291.269	291.269*	0	1	0	0.015	0.296	7.269
PUC_r_18_18_4	36	1332	744	286.077	271.128	286.077	286.077*	0	25	6	0.428	0.263	14.330
PUC_r_18_18_5	36	1332	751	263.764	258.370	263.764	263.764*	0	29	6	0.396	0.586	34.084
PUC_r_20_20_1	40	1640	62	347.491	347.491	347.491	347.491*	0	1	0	0.041	0.329	34.362
PUC_r_20_20_2	40	1640	203	381.249	375.428	381.249	381.249*	0	11	3	0.059	1.393	50.038
PUC_r_20_20_3	40	1640	74	391.299	388.299	391.299	391.299*	0	21	5	0.149	1.564	117.337
PUC_r_20_20_4	40	1640	518	399.453	386.459	399.453	399.453*	0	415	19	5.420	0.521	36.286
PUC_r_20_20_5	40	1640	857	366.812	362.779	366.812	366.812*	0	7	2	0.099	0.560	22.120
PUC_r_22_22_1	44	1980	397	462.137	442.542	459.009	459.009*	0	1971	22	41.849	0.349	39.520
PUC_r_22_22_2	44	1980	242	413.547	413.333	413.547	413.547*	0	7	3	0.051	1.415	45.124
PUC_r_22_22_3	44	1980	567	433.645	417.588	433.645	433.645*	0	309	15	5.019	0.339	21.941
PUC_r_22_22_4	44	1980	254	461.945	456.464	461.945	461.945*	0	21	7	0.192	0.941	45.804
PUC_r_22_22_5	44	1980	69	448.584	447.535	448.584	448.584*	0	3	1	0.023	0.964	64.094
PUC_r_24_24_1	48	2352	743	460.297	449.519	460.297	460.297*	0	133	12	1.782	0.366	30.602
PUC_r_24_24_2	48	2352	163	477.413	471.406	477.413	477.413*	0	9	4	0.068	0.757	37.332

Table 5.7: Results for the branch-and-cut algorithm (continued)

Instance	**|V|**	**|E|**	$\|\mathbf{E}\|_R$	$\mathbf{P}_{ILS}$	**LP**	**LB**	**UB**	**GAP**	**N**	**D**	**T(s)**	**FS(%)**	**PtPS(%)**
PUC_r_24_24_3	48	2352	308	418.466	417.656	418.466	418.466*	0	7	2	0.057	1.034	38.383
PUC_r_24_24_4	48	2352	55	459.073	459.073	459.073	459.073*	0	1	0	0.032	0.236	12.952
PUC_r_24_24_5	48	2352	96	480.623	479.040	480.623	480.623*	0	5	2	0.028	1.121	56.297
PUC_r_26_26_1	52	2756	225	573.996	555.044	573.996	573.996*	0	509	19	8.082	0.693	31.375
PUC_r_26_26_2	52	2756	59	515.262	515.262	515.262	515.262*	0	1	0	0.034	0.232	13.949
PUC_r_26_26_3	52	2756	473	594.353	591.540	594.353	594.353*	0	11	3	0.109	0.825	35.254
PUC_r_26_26_4	52	2756	868	472.589	460.842	472.589	472.589*	0	23	4	0.487	0.581	26.455
PUC_r_26_26_5	52	2756	314	585.796	583.152	585.796	585.796*	0	7	3	0.109	0.708	54.720
PUC_r_28_28_1	56	3192	637	640.927	625.607	640.927	640.927*	0	9	3	0.204	0.358	42.700
PUC_r_28_28_2	56	3192	2156	631.737	618.707	631.737	631.737*	0	3351	33	152.307	0.339	48.586
PUC_r_28_28_3	56	3192	446	594.231	592.016	594.231	594.231*	0	13	5	0.239	0.778	60.185
PUC_r_28_28_4	56	3192	726	570.737	565.029	570.737	570.737*	0	19	5	0.288	0.700	26.197
PUC_r_28_28_5	56	3192	62	544.909	544.909	544.909	544.909*	0	1	0	0.036	0.162	7.996
PUC_r_30_30_1	60	3660	298	639.401	630.131	639.401	639.401*	0	219	16	4.971	0.709	45.587
PUC_r_30_30_2	60	3660	915	722.468	705.385	722.468	722.468*	0	345	14	10.810	0.396	62.843
PUC_r_30_30_3	60	3660	588	732.902	729.395	732.902	732.902*	0	7	2	0.154	0.669	47.423
PUC_r_30_30_4	60	3660	2004	668.177	660.781	668.177	668.177*	0	111	9	3.865	0.333	22.469
PUC_r_30_30_5	60	3660	1213	706.650	693.540	706.650	706.65*	0	83	11	2.054	0.396	20.798
PUC_r_32_32_1	64	4160	1743	802.768	769.781	788.832	788.832*	0	2403	23	153.347	0.219	28.960
PUC_r_32_32_2	64	4160	333	748.656	746.038	748.656	748.656*	0	7	3	0.192	0.511	72.189
PUC_r_32_32_3	64	4160	407	754.849	754.848	754.849	754.849*	0	3	1	0.059	0.703	63.288
PUC_r_32_32_4	64	4160	1807	805.771	786.561	805.771	805.771*	0	8689	27	476.687	0.379	53.608
PUC_r_32_32_5	64	4160	990	789.849	773.009	789.849	789.849*	0	519	22	14.419	0.605	58.368
PUC_r_40_40_1	80	6480	2561	1089.771	1055.601	1076.832	1076.832*	0	159	17	9.719	0.371	35.372
PUC_r_40_40_2	80	6480	3785	1084.382	1044.578	1075.806	1075.805*	0	38033	37	3841.795	0.442	49.026
PUC_r_40_40_3	80	6480	5089	1048.912	1024.105	1048.912	1048.912*	0	11647	33	1129.916	0.134	9.345

Table 5.8: Results for the branch-and-cut algorithm

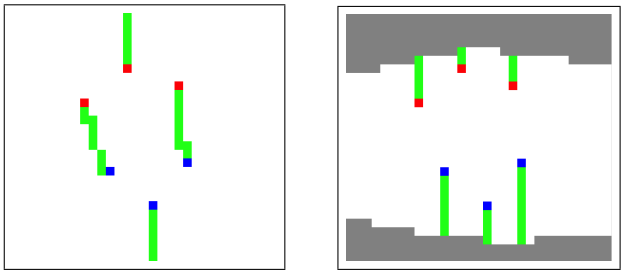
Instance	**|V|**	**|E|**	$\|\mathbf{E}\|_R$	$\mathbf{P}_{ILS}$	**LP**	**LB**	**UB**	**GAP**	**N**	**D**	**T(s)**	**FS(%)**	**PtPS(%)**
PUC_r_40_40_4	80	6480	4436	1114.463	1100.404	1114.463	1114.463*	0	4075	28	291.181	0.190	28.201
PUC_r_40_40_5	80	6480	972	1122.355	1102.941	1122.355	1122.355*	0	255	13	15.207	0.533	71.083
PUC_r_48_48_1	96	9312	913	1348.944	1338.588	1348.944	1348.944*	0	2139	28	110.450	0.517	64.550
PUC_r_48_48_2	96	9312	8460	1613.758	1558.175	1577.914	1613.758	2.221	26356	63	3600.000	0.191	60.489
PUC_r_48_48_3	96	9312	6988	1511.631	1467.523	1499.526	1511.631	0.801	35299	47	3600.000	0.231	45.315
PUC_r_48_48_4	96	9312	7043	1495.145	1450.212	1487.522	1487.522	0	32970	53	3600.000	0.169	35.546
PUC_r_48_48_5	96	9312	801	1422.952	1420.794	1422.952	1422.952*	0	5	2	0.180	0.494	64.952
PUC_r_64_64_1	128	16512	13608	2450.873	2356.620	2248.345	2425.795	7.315	12641	70	3600.000	0.130	25.815
PUC_r_64_64_2	128	16512	1634	2166.769	2134.436	2166.769	2166.769*	0	4311	26	657.490	0.367	65.692
PUC_r_64_64_3	128	16512	6756	2576.421	2340.008	2460.774	2467.719	0.281	26396	50	3600.000	0.312	43.087
PUC_r_64_64_4	128	16512	13690	2293.612	2183.651	2167.674	2293.612	5.491	12463	66	3600.000	0.116	29.226
PUC_r_64_64_5	128	16512	1197	2269.257	2258.864	2269.257	2269.257*	0	283	17	52.685	0.255	62.094
PUC_r_128_128_1	256	65792	65474	7061.173	6483.476	7017.541	7061.173	0.618	896	136	3600.000	0.035	53.197
PUC_r_128_128_2	256	65792	65327	7748.266	6867.976	7257.440	7748.266	6.335	1036	164	3600.000	0.050	43.097
PUC_r_128_128_3	256	65792	65792	7518.650	6437.926	7442.549	7518.65	1.012	958	159	3600.000	0.046	30.388
PUC_r_128_128_4	256	65792	61682	7530.240	6643.135	7411.020	7530.174	1.582	1310	139	3600.000	0.042	13.903
PUC_r_128_128_5	256	65792	61820	6913.376	6660.020	6835.855	6913.376	1.121	1085	130	3600.000	0.045	29.568
PUC_r_256_256_1	512	262656	262656	23517.256	19304.997	19760.603	23517.256	15.974	59	58	3600.000	0.010	12.998
PUC_r_256_256_2	512	262656	262656	23286.799	19383.696	19742.804	23286.799	15.219	58	57	3600.000	0.008	31.360
PUC_r_256_256_3	512	262656	262656	22874.346	18741.290	19231.867	22874.346	15.924	59	58	3600.000	0.015	29.410
PUC_r_256_256_4	512	262656	262656	23442.233	19100.228	19649.885	23442.233	16.177	65	64	3600.000	0.011	8.211
PUC_r_256_256_5	512	262656	262656	22863.504	19179.963	19793.053	22863.504	13.429	65	64	3600.000	0.008	13.216
PUC_r_512_512_1	1024	1049600	1049600	71011.166	55306.063	55340.467	69526.195	20.403	4	3	3600.000	0.002	12.655
PUC_r_512_512_2	1024	1049600	1049600	69510.146	54360.603	54543.150	65418.516	16.624	3	2	3600.000	0.002	6.630
PUC_r_512_512_3	1024	1049600	1049600	69401.363	54354.107	54387.779	65652.68	17.158	3	2	3600.000	0.002	28.730
PUC_r_512_512_4	1024	1049600	1049600	68040.880	53175.929	53177.495	66275.43	19.763	4	3	3600.000	0.003	12.707
PUC_r_512_512_5	1024	1049600	1049600	69884.320	54478.709	54513.879	67077.516	18.73	4	3	3600.000	0.002	17.063

5.2
2D Phase Unwrapping Instances

We will now illustrate the performance of the MSFBC approach in the 2DPU domain. We have established three metrics in order to evaluate and compare the quality of each solution with other path-following methods: (**N**) the total number of absolute phase gradients that differ from their wrapped counterparts; (**L**) The total length of the branch-cuts (not applicable for Goldstein's algorithm) and; (**C**) The number of connected components of the branch-cuts.

The instances available in [2] are widely used as benchmarks for phase unwrapping algorithms. We will perform our tests on three of these instances: *Long's Peak*, *Isola's Peak* and *Head Magnetic Resonance Image (MRI)*. Each one of them has special particularities and simulate real world phase unwrapping applications.

The following sections will first give a brief introduction for each instance, discuss its challenges, present the results obtained by the path-following methods described in Sections 4.1.1 and 4.1.2 and finally present the results for the MSFBC approach. The set of the MSFBC solutions used for comparative results refers to the best primal solutions obtained considering all proposed methods, with a time bound of 3600 seconds. One important note is that although the first two instances were provided with additional information to mask out regions with bad quality pixels, our tests will only use this information in the unwrapping process and not for redefining the position of border points. When masks are available, border points can be considered as the limits of the masked regions and not the boundaries of the image, as shown in Figure 5.3.



5.3(a): 5.3(b):

Figure 5.3: *The use of masks to redefine border points and remove regions with bad quality pixels from the unwrapping procedure. (a) The branch-cut solution obtained without masked regions. (b) The same instance with the new boundaries established by the mask information (in gray).*

5.2.1 Long's Peak

Figure 5.4(a) shows the wrapped phase image of a real steep-relief mountainous region of Long's Peak, Colorado (USA), obtained through a high-fidelity InSAR [2] simulator. The topology of residues clearly suggests natural phase discontinuities caused by the terrain's elevation and the additional occurrence of noise. There are 846 residues (422 positives and 426 negatives) distributed over a 152x458-pixel image. The greatest challenge imposed by this instance is to efficiently cluster the sparse group of residues. There is a large number of regions with a high density of residues that could mislead a less careful heuristic approach, specially concerning the occurrence of isolated regions.

Figures 5.4(c) and 5.4(d) show that even though Goldstein's algorithm was able to cluster close groups of residues, the number of unnecessary long connections and isolated regions had a direct impact in the unwrapping result. The minimum cost matching algorithm (Figures 5.4(e) and 5.4(f)) failed to preserve the structural delimitations suggested by the topology of residues, thus introducing critical discontinuities to the unwrapped solution. On the other hand, the MSFBC approach was able to efficiently cluster groups of close residues and respect their topology. The efficiency of the method is supported by the quality of the unwrapped result, where the number of discontinuities introduced by the branch-cuts are greatly diminished. Table 5.9 shows the comparative results between the three approaches. The MSFBC approach

Table 5.9: Comparative results for Long's Peak

Method	**N**	**L**	**C**
Goldstein	1437	-	49
Minimum Cost Matching (MCM)	1075	1545.38187	429
MSFBC	975	1264.31324	68

had an improvement of 32.15% over Goldstein's algorithm and 9.30% over the minimum cost matching algorithm in the number of discontinuity points. Also, the total length of the branch-cuts was improved by 18% when compared to the MCM algorithm. Figure 5.5 highlight some of the visual unwrapping improvements obtained with the MSFBC approach when compared to the other methods, while Figure 5.6 shows the 2π discontinuity maps produced by each method.

5.2.2 Isola's Peak

Figure 5.7(a) shows the wrapped phase image of another real steep-relief mountainous region in Colorado (USA) called Isola's Peak, obtained through a high-fidelity InSAR simulator. The challenge on this problem is to correctly unwrap phase data around and between the numerous regions with natural phase discontinuities, without propagating errors in the unwrapped surface. There are 1234 residues (616 positives and 618 negatives) distributed over a 157x458-pixel image. Again, the objective is to efficiently cluster the sparse group of residues and handle regions with a high density of residues.

The results showed in Figure 5.7 enlightens that although the unwrapping results are visually quite similar, the branch-cuts configuration obtained by the MSFBC approach is clearly more efficient (Figure 5.7(g)). The groups of residues were clustered in a much more efficient way and the branch-cuts were able to respect many of the structural delimitations. Table 5.10 compares the results of the three approaches.

Table 5.10: Comparative results for Isola's Peak

Method	**N**	**L**	**C**
Goldstein	2127	-	39
Minimum Cost Matching (MCM)	1825	2545.06421	625
MSFBC	1609	1850.23338	57

The MSFBC approach had an improvement of 24.35% over Goldstein's algorithm and 11.83% over the minimum cost matching algorithm in the number of discontinuity points. The total length of the branch-cuts was improved by 27.3% when compared to the MCM algorithm. Figure 5.8 highlight some of the visual unwrapping improvements obtained with the MSFBC approach when compared to the other methods, while Figure 5.12 shows the 2π discontinuity maps produced by each method.

5.2.3 Head Magnetic Resonance Image (MRI)

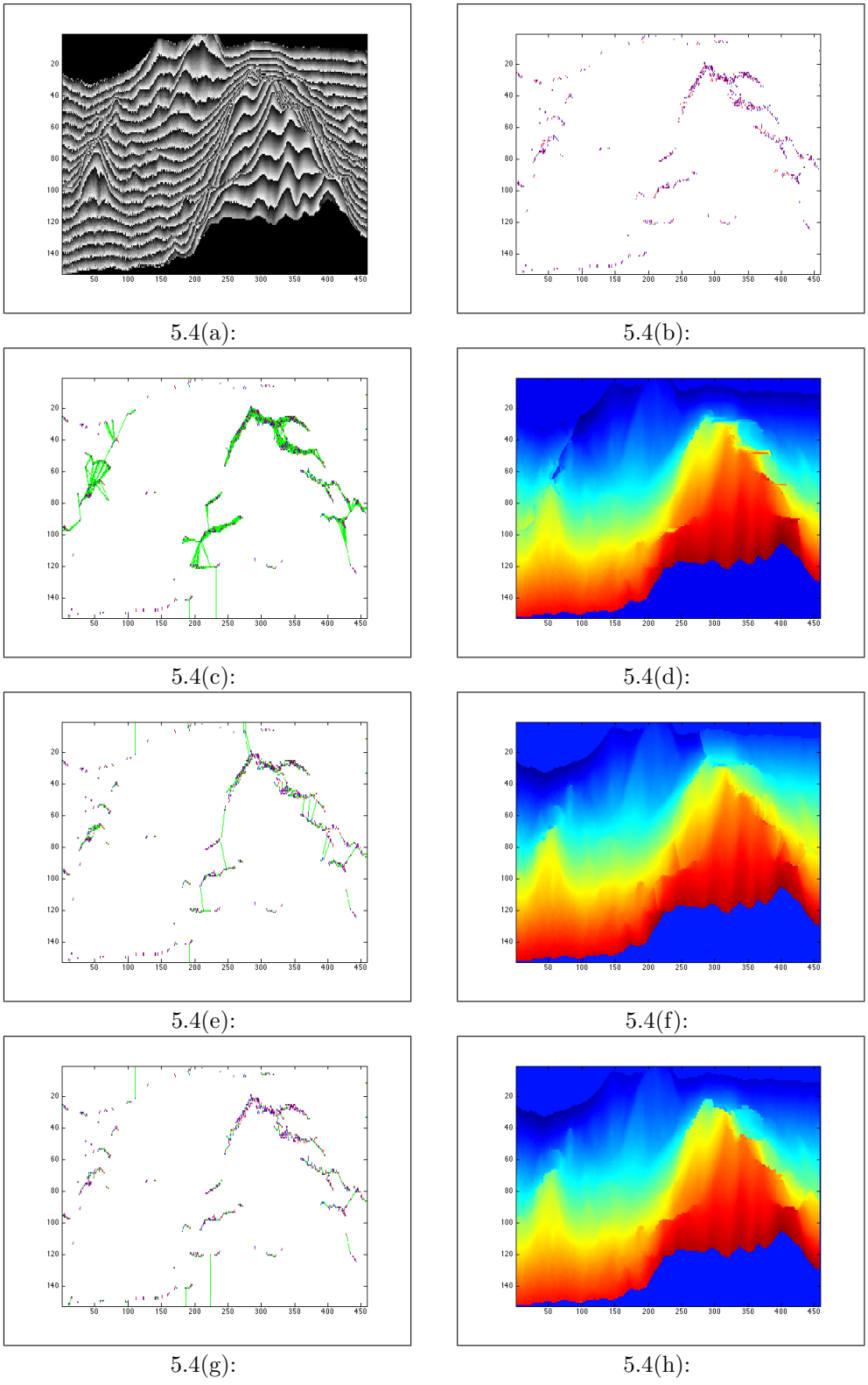
The water/fat separation problem arising from magnetic resonance imaging is another application where the need for phase unwrapping arises. The signal obtained from conventional MRI procedures captures the water and fat intensity values from tissues, shifted by a phase value in each sample. In order to separate the water and fat information, the continuous phase signal must be reconstructed. Figure 5.10(a) shows a wrapped phase image of an head MRI experiment with 1926 residues (963 positives and 963 negatives) defined on a 256x256-pixel grid. This instance is considered to pose a difficult problem to the unwrapping procedure since various regions are delimited by residues and appear to be completely isolated from one another.

Figure 5.10(c) shows that Goldstein's algorithm failed to cluster groups of close residues, creating many long connections and isolated regions. The outcome of such poor placement of branch-cuts is illustrated in the unwrapping result (Figure 5.10(d)). The MCM and MSFBC approaches, on the other hand, produced very similar solutions, both in the placement of branch-cuts as in the unwrapping result. Since there is a high density of residues mainly composed by close dipoles, it was expected that the MCM algorithm would perform much better than Goldstein's. Table 5.11 summarizes the solution quality obtained by each method.

Table 5.11: Comparative results for Head's MRI

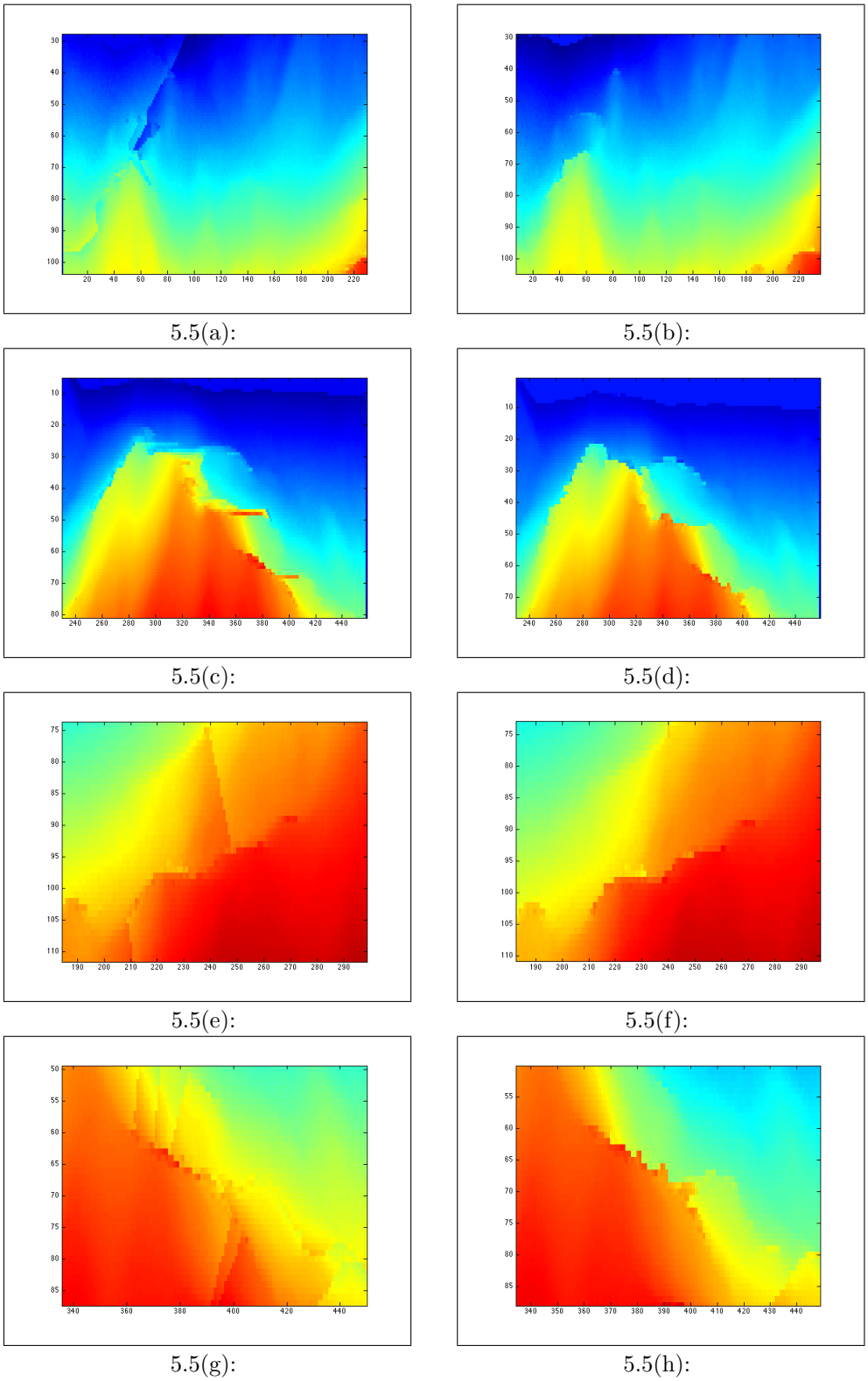
Method	N	L	C
Goldstein	2570	-	153
Minimum Cost Matching (MCM)	1789	1588.7282	963
MSFBC	1810	1722.564819	57

The MSFBC approach had an improvement of 29.87% over Goldstein's algorithm in the number of discontinuity points and a 1.16% gap over the minimum cost matching algorithm, which was able to produce an improvement of 7.76% in the total length of the branch-cuts. The results showed that the matching algorithm found a near optimal solution for the L0-norm, mainly because of the topology of residues. There are many pairs of close dipoles spread along the whole instance, which gives the MCM a clear advantage. However, since the MSFBC solutions were obtained by heuristic methods, there is no guarantee on optimality. Figure 5.11 highlight some of the local unwrapping differences between the MSFBC approach and the minimum cost matching algorithm, while Figure 5.12 shows the 2π discontinuity maps produced by each method.



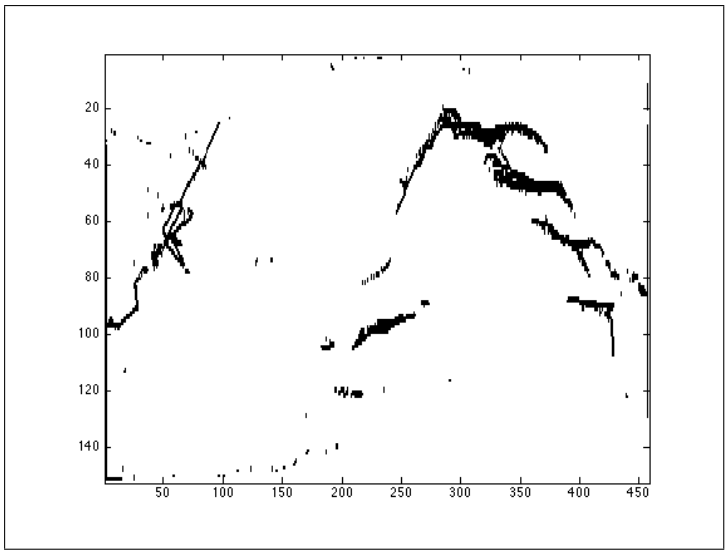
5.4(a): 5.4(b): 5.4(c): 5.4(d): 5.4(e): 5.4(f): 5.4(g): 5.4(h):

Figure 5.4: *Long's Peak instance with 846 residues distributed over a 152x458-pixel image. Comparative results between Goldstein's, the minimum cost matching algorithm and the MSFBC approach. (a) The wrapped phase image. (b) The topology of residues. (c) Goldstein's branch-cuts configuration. (d) Goldstein's unwrapped solution. (e) The minimum cost matching algorithm branch-cut configuration. (f) The minimum cost matching algorithm unwrapped solution. (g) The MSFBC branch-cut configuraion. (h) The MSFBC unwrapped solution.*

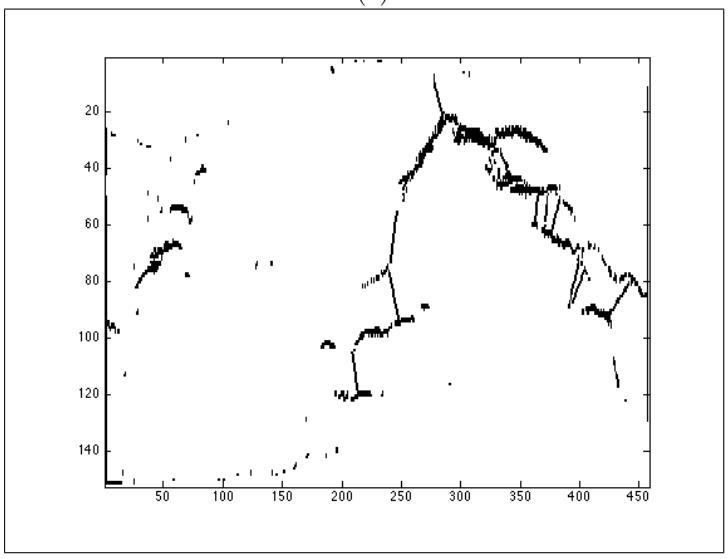


5.5(a): 5.5(b): 5.5(c): 5.5(d): 5.5(e): 5.5(f): 5.5(g): 5.5(h):

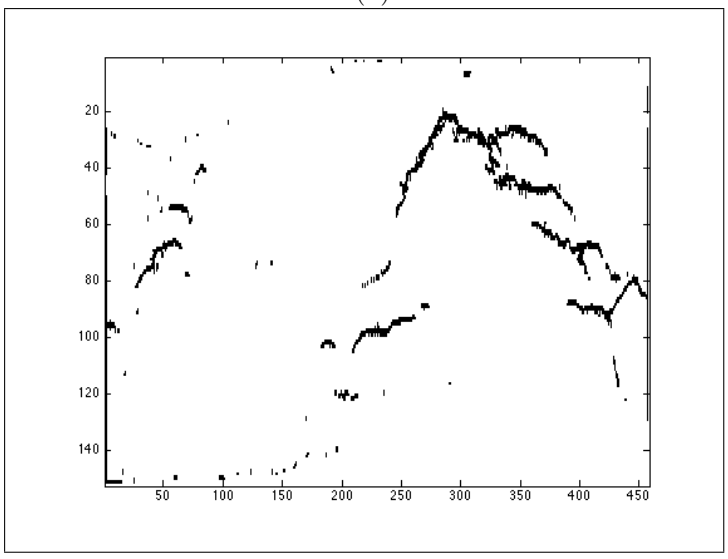
Figure 5.5: *Some of the improvements obtained by the MSFBC approach over Goldstein's and MCM for Long's Peak. (a) and (c) Different regions using Goldstein's algorithm. (b) and (d) The equivalent regions with the MSFBC approach. (e) and (g) Different regions with the MCM algorithm. (f) and (h) The equivalent regions with the MSFBC approach.*



5.6(a):

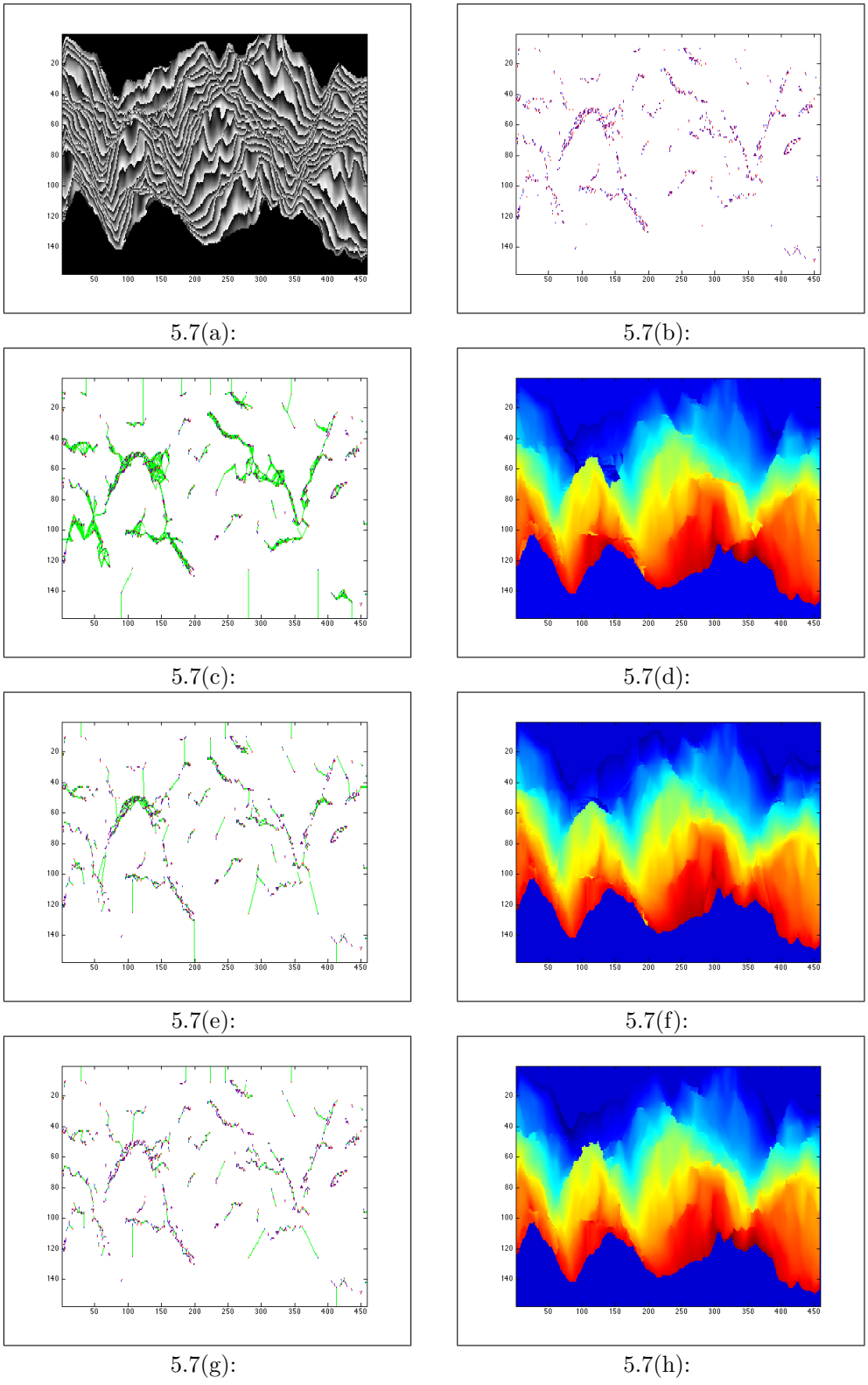


5.6(b):



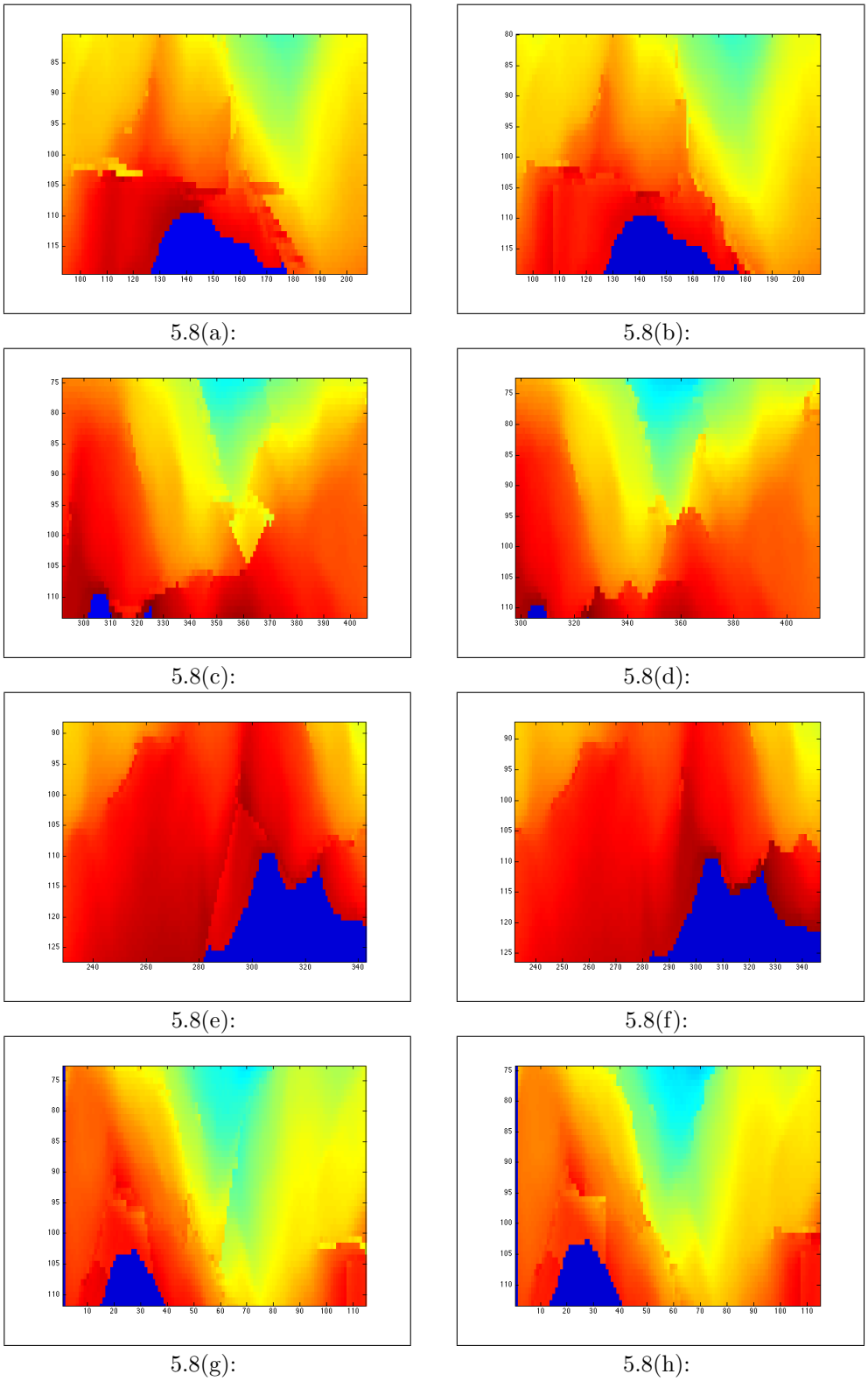
5.6(c):

Figure 5.6: *The discontinuity maps produced by each method for Long's Peak. (a) Goldstein's discontinuity map. (b) The minimum cost matching algorithm's discontinuity map. (c) The MSFBC approach discontinuity map.*



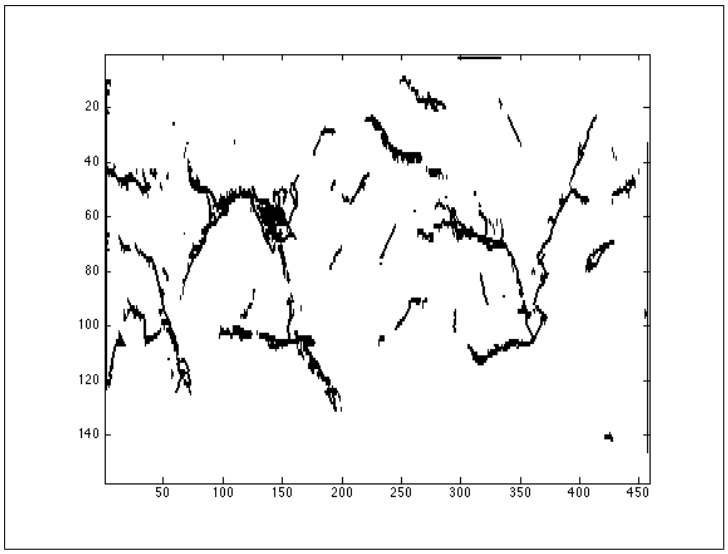
5.7(a): 5.7(b): 5.7(c): 5.7(d): 5.7(e): 5.7(f): 5.7(g): 5.7(h):

Figure 5.7: *Isola's Peak instance with 1234 residues distributed over a 157x458-pixel image. Comparative results between Goldstein's, the minimum cost matching algorithm and the MSFBC approach. (a) The wrapped phase image. (b) The topology of residues. (c) Goldstein's branch-cuts configuration. (d) Goldstein's unwrapped solution. (e) The minimum cost matching algorithm branch-cut configuration. (f) The minimum cost matching algorithm unwrapped solution. (g) The MSFBC branch-cut configuraion. (h) The MSFBC unwrapped solution.*

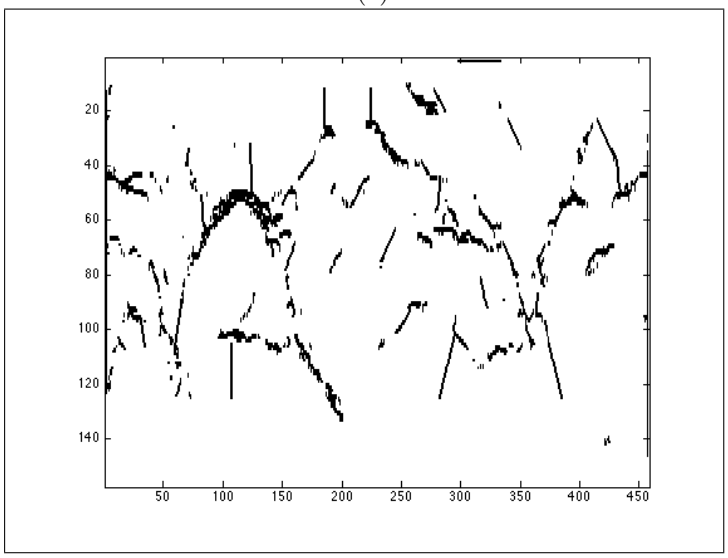


5.8(a): 5.8(b): 5.8(c): 5.8(d): 5.8(e): 5.8(f): 5.8(g): 5.8(h):

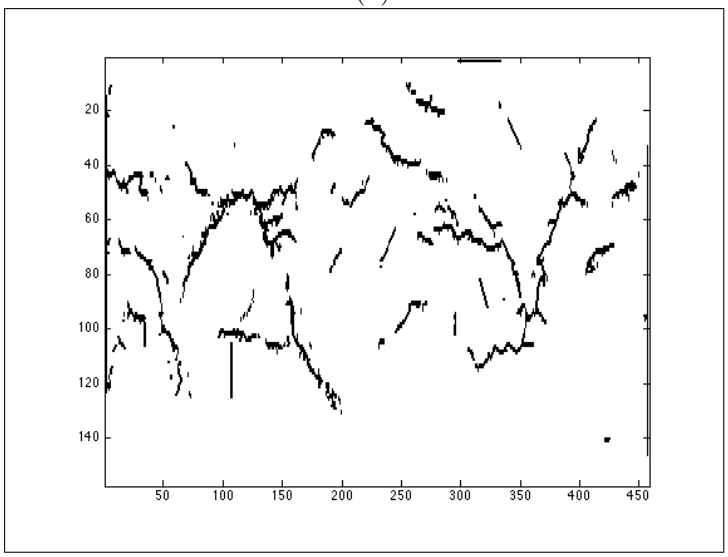
Figure 5.8: *Some of the improvements obtained by the MSFBC unwrapped solution over Goldstein's and MCM for Isola's Peak. (a) and (c) Different regions using Goldstein's algorithm. (b) and (d) The equivalent regions with the MSFBC approach. (e) and (g) Different regions with the MCM algorithm. (f) and (h) The equivalent regions with the MSFBC approach.*



5.9(a):

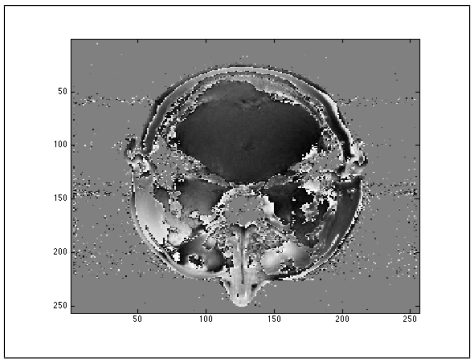


5.9(b):

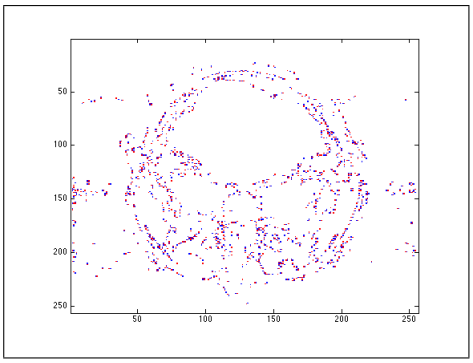


5.9(c):

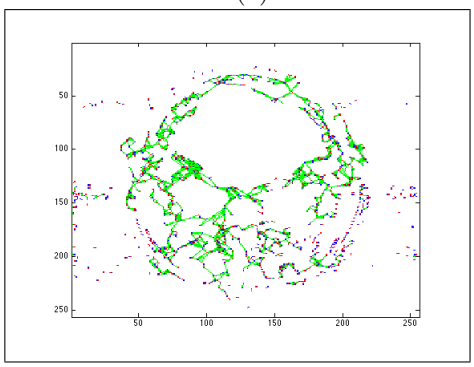
Figure 5.9: *The discontinuity maps produced by each method for Isola's Peak. (a) Goldstein's discontinuity map. (b) The minimum cost matching algorithm's discontinuity map. (c) The MSFBC approach discontinuity map.*



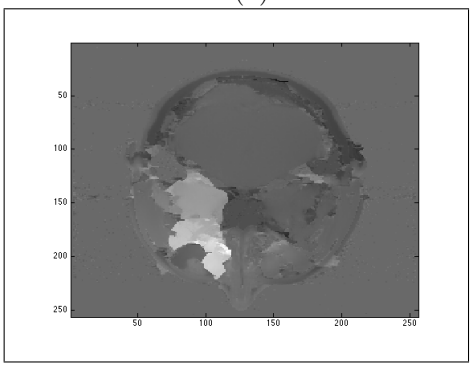
5.10(a):



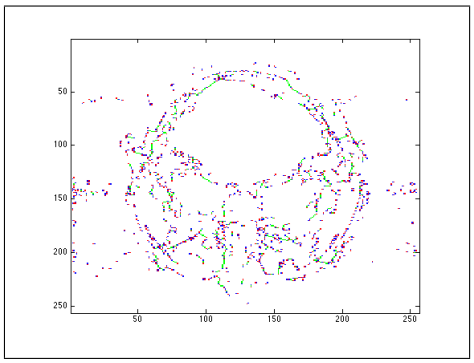
5.10(b):



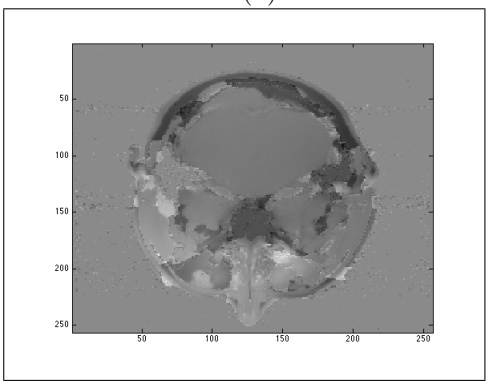
5.10(c):



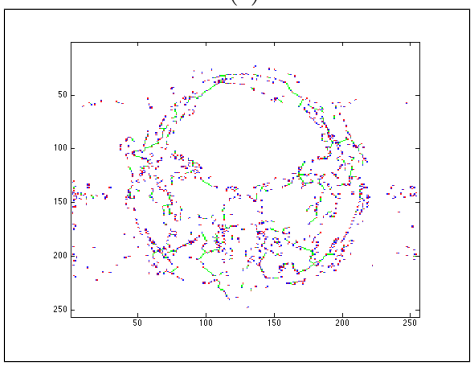
5.10(d):



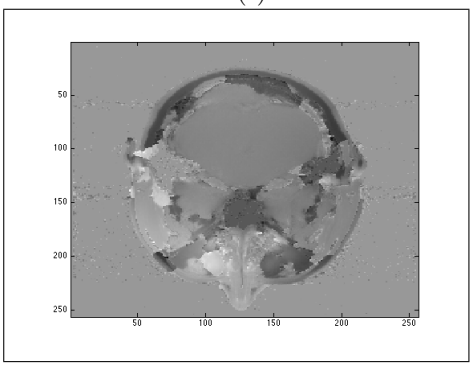
5.10(e):



5.10(f):

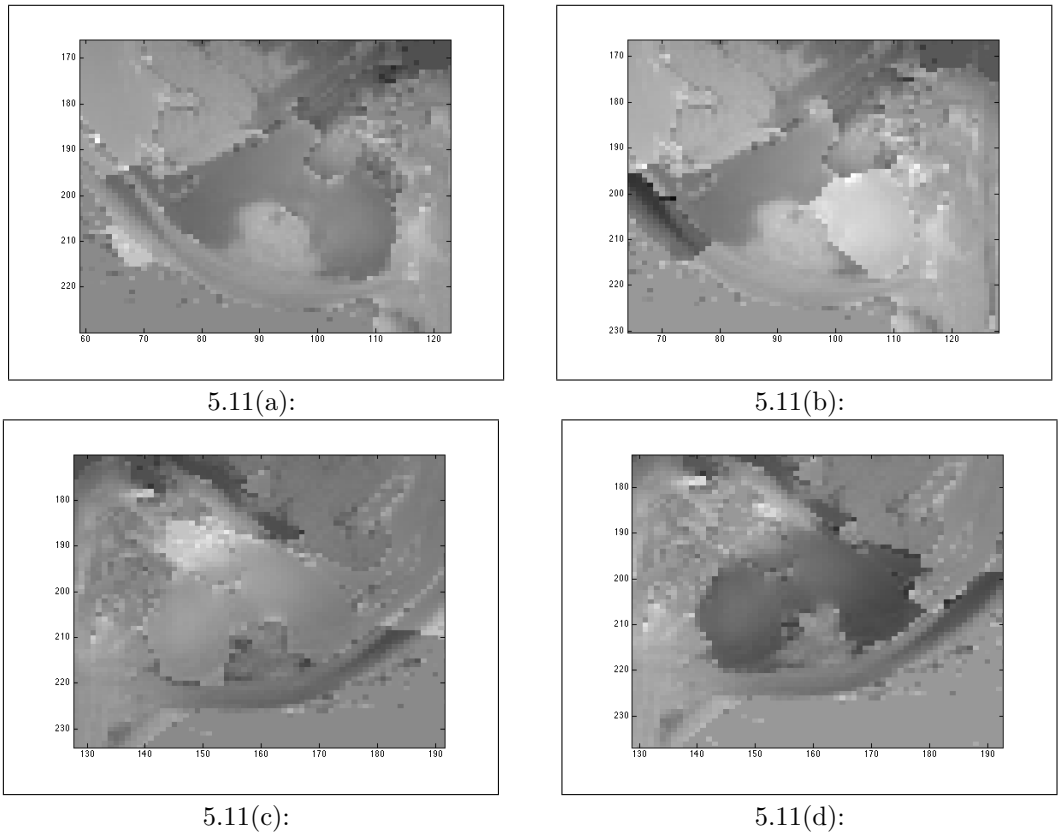


5.10(g):



5.10(h):

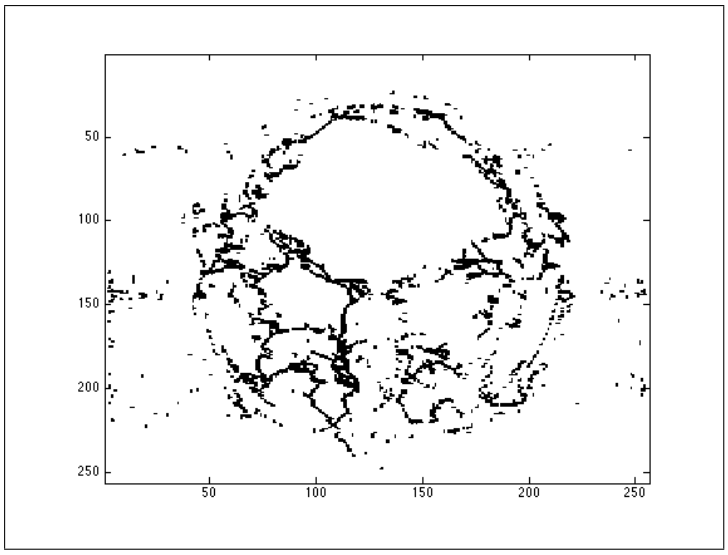
Figure 5.10: *Head MRI instance with 1926 residues distributed over a 256x256-pixel image. Comparative results between Goldstein's, the minimum cost matching algorithm and the MSFBC approach. (a) The wrapped phase image. (b) The topology of residues. (c) Goldstein's branch-cuts configuration. (d) Goldstein's unwrapped solution. (e) The minimum cost matching algorithm branch-cut configuration. (f) The minimum cost matching algorithm unwrapped solution. (g) The MSFBC branch-cut configuraion. (h) The MSFBC unwrapped solution.*



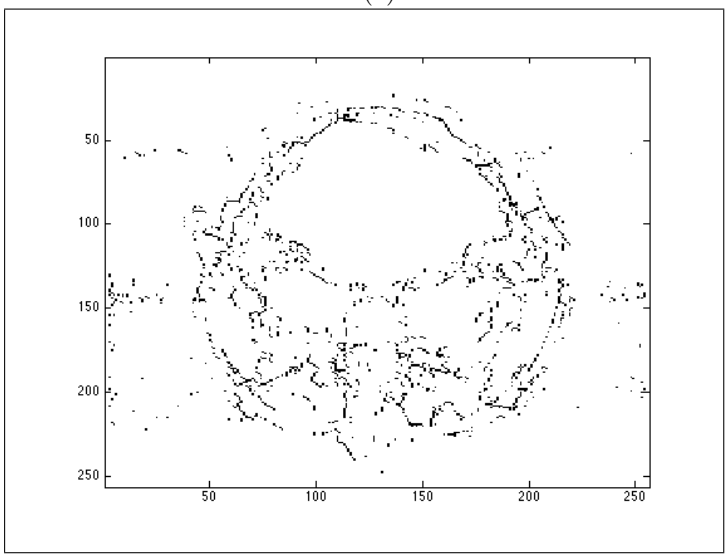
5.11(a): 5.11(b):

5.11(c): 5.11(d):

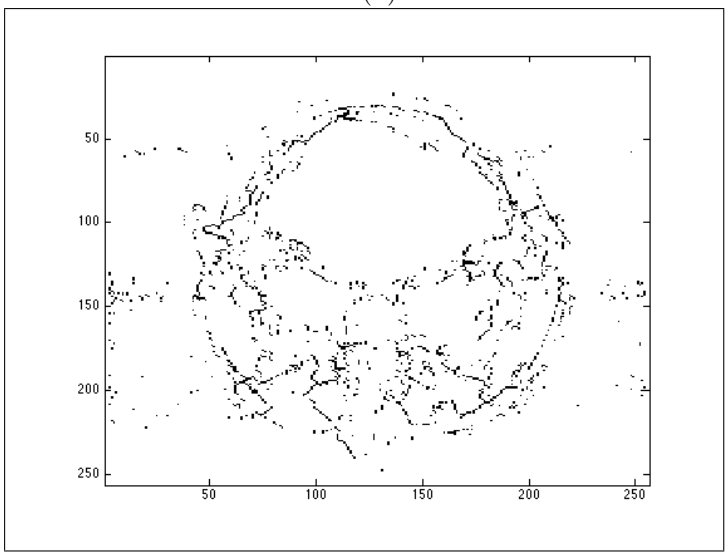
Figure 5.11: *Some of the improvements obtained by the MSFBC unwrapped solution over Goldstein's and MCM for Isola's Peak. (a) and (c) Different regions using Goldstein's algorithm. (b) and (d) The equivalent regions with the MSFBC approach. (e) and (g) Different regions with the MCM algorithm. (f) and (h) The equivalent regions with the MSFBC approach.*



5.12(a):



5.12(b):



5.12(c):

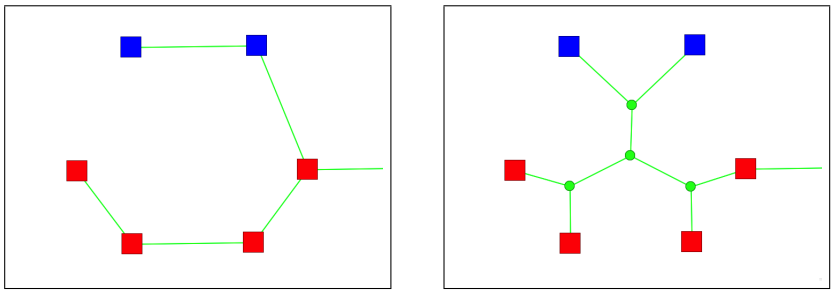
Figure 5.12: *The discontinuity maps produced by each method for Head's MRI. (a) Goldstein's discontinuity map. (b) The minimum cost matching algorithm's discontinuity map. (c) The MSFBC approach discontinuity map.*

6
Concluding Remarks

In this dissertation, we have proposed a new approach for the 2D phase unwrapping problem, along with a new set of mathematical formulations and methods to address the L^0-norm of the problem, considered to be highly desirable but intractable. We developed efficient methods known from the field of optimization and operational research which were able to solve instances to optimality with up to 128 residues in a reasonable computational time. We also designed a hybrid metaheuristic approach which produces approximate solutions in a reasonable computational time for larger instances.

The proposed methods constituted a better approximation for the L^0-norm of the 2DPU problem in two of the three benchmark instances tested, when compared to other path-following state-of-the-art methods. However, since the solutions obtained were produced by heuristic methods, there is no guarantee on optimality. In fact, the optimal solution for the MSFBC approach would be theoretically better than any solution found by path-following methods whose goal is to minimize the total length of the branch-cuts.

As discussed in Chapter 3, the true minimization of the L^0-norm reduces to the problem of finding an optimal Euclidean Steiner forest, where every Steiner tree respects the balance constraints of positives and negatives residues. Now, remark that the same transformation as Section 4.1 can be used to model any Steiner point as a pair of (negative/positive) residues. This would allow, in future research, to evaluate our spanning tree approach on the exact L^0-norm objective.



6.1(a): 6.1(b):

Figure 6.1: *The difference between connecting residues with a minimum spanning tree and a Steiner tree for a group of residues. (a) The Minimum Spanning tree soltuion. (b) The Steiner-tree solution.*

In the previous chapters we discussed the fact that finding a branch-cut configuration that respects the topology of residues would lead to visually better unwrapped solutions. Figure 6.1 shows an example of the different branch-cut configurations produced by the MSFBC approach and Steiner's. Clearly, the discontinuities introduced by the branch-cuts would produce different visual unwrapped solutions. It is not clear that, in certain cases, the insertion of Steiner points could decharacterize these structural delimitations. This scenario raises questions about using the true minimization of the L^0-norm as an optimal unwrapped solution criterion.

For the MSFBC problem itself, future works include: (1) designing new sets of instances tailored to test more thoroughly the limitations of the proposed methods; (2) devise a column generation approach for the MSFBC problem; (3) devise general improvements over the heuristic methods, new neighbourhoods for the hybrid ILS algorithm and new dual heuristics with different criteria for the selection of violated cuts; (4) investigate the performance of heuristic methods for the Steiner-tree problem over MSFBC instances; (4) devise the reduction from the Steiner forest with balance constraints problem to the MSFBC

In the 2DPU domain, many paths of future work are possible. We highlight: (1) investigate the performance of using Steiner heuristics to find the optimal branch-cut configuration; (2) test the proposed heuristic methods on new sets of phase unwrapping instances; (3) Include the information of quality maps and masks to the MSFBC methods; (4) Devise new metrics to better compare the solutions obtained by different methods; (5) Compare the results with general 2DPU approaches, including L^p-norm methods.

7
Bibliography

[1] GOLDSTEIN, R. M.; ZEBKER, H. A. ; WERNER, C. L. **Radio science**. Satellite radar interferometry: Two-dimensional phase unwrapping, v.23, n.4, p. 713–720, 1988.

[2] GHIGLIA, D. C.; PRITT, M. D. **Two-dimensional phase unwrapping: theory, algorithms, and software**. Wiley New York:, 1998.

[3] OPPENHEIM, A. V.; LIM, J. S. **Proceedings of the IEEE**. The importance of phase in signals, v.69, n.5, p. 529–541, 1981.

[4] ITOH, K. **Applied Optics**. Analysis of the phase unwrapping algorithm, v.21, n.14, p. 2470–2470, 1982.

[5] JERRI, A. J. **Proceedings of the IEEE**. The Shannon sampling theorem: Its various extensions and applications: A tutorial review, v.65, n.11, p. 1565–1596, 1977.

[6] GHIGLIA, D. C.; MASTIN, G. A. ; ROMERO, L. A. **JOSA A**. Cellular-automata method for phase unwrapping, v.4, n.1, p. 267–280, 1987.

[7] AGARWAL, R. P.; PERERA, K. ; PINELAS, S. **Cauchy's residue theorem**. In: An Introduction to Complex Analysis, p. 207–214. Springer, 2011.

[8] HUNTLEY, J.; BUCKLAND, J. **JOSA A**. Characterization of sources of 2π phase discontinuity in speckle interferograms, v.12, n.9, p. 1990–1996, 1995.

[9] GUO, L.; KANG, L. ; WANG, D. **Journal of Southern Medical University**. Improvement of magnetic resonance phase unwrapping method based on Goldstein branch-cut algorithm, v.33, n.2, p. 239–242, 2013.

[10] DERAUW, D. **Phase unwrapping using coherence measurements**. In: Satellite Remote Sensing II, p. 319–324. International Society for Optics and Photonics, 1995.

[11] BUCKLAND, J.; HUNTLEY, J. ; TURNER, S. **Applied Optics**. Unwrapping noisy phase maps by use of a minimum-cost-matching algorithm, v.34, n.23, p. 5100–5108, 1995.

[12] FRANK, A. **Naval Research Logistics**. On Kuhn's Hungarian method: a tribute from Hungary, v.52, n.1, p. 2–5, 2005.

[13] FORD, L.; FULKERSON, D. R. **Flows in networks**, v.1962, Princeton, NY, USA: Princeton University Press, 1962.

[14] GDEISAT, M.; AREVALILLO-HERRÁEZ, M.; BURTON, D. ; LILLEY, F. **Optics letters**. Three-dimensional phase unwrapping using the hungarian algorithm, v.34, n.19, p. 2994–2996, 2009.

[15] ROTH, M.; LAUREL, M. **Johns Hopkins University Applied Physics Lab Technical Report**. Phase unwrapping for interferometric SAR by the least-error path, v.30, 1995.

[16] ZHAO, M.; HUANG, L.; ZHANG, Q.; SU, X.; ASUNDI, A.; KEMAO, Q. ; OTHERS. **Applied optics**. Quality-guided phase unwrapping technique: comparison of quality maps and guiding strategies, v.50, n.33, p. 6214–6224, 2011.

[17] LU, Y.; WANG, X. ; ZHANG, X. **Optik-International Journal for Light and Electron Optics**. Weighted least-squares phase unwrapping algorithm based on derivative variance correlation map, v.118, n.2, p. 62–66, 2007.

[18] FRIED, D. L. **JOSA**. Least-square fitting a wave-front distortion estimate to an array of phase-difference measurements, v.67, n.3, p. 370–375, 1977.

[19] HUDGIN, R. H. **Journal of the Optical Society of America**. Wave-front reconstruction for compensated imaging, v.67, n.3, p. 375–378, Mar 1977.

[20] GHIGLIA, D. C.; ROMERO, L. A. **JOSA A**. Robust two-dimensional weighted and unweighted phase unwrapping that uses fast transforms and iterative methods, v.11, n.1, p. 107–117, 1994.

[21] CHEN, C. W.; ZEBKER, H. A. **JOSA A**. Two-dimensional phase unwrapping with use of statistical models for cost functions in nonlinear optimization, v.18, n.2, p. 338–351, 2001.

[22] FLYNN, T. J. **JOSA A**. Two-dimensional phase unwrapping with minimum weighted discontinuity v.14, n.10, p. 2692–2701, 1997.

[23] COSTANTINI, M. **Geoscience and Remote Sensing, IEEE Transactions on**. A novel phase unwrapping method based on network programming, v.36, n.3, p. 813–821, 1998.

[24] CHOPRA, S.; RAO, M. **Mathematical Programming**. The Steiner tree problem i: Formulations, compositions and extension of facets, v.64, n.1-3, p. 209–229, 1994.

[25] GRÖTSCHEL, M.; LOVÁSZ, L. ; SCHRIJVER, A. **Combinatorica**. The ellipsoid method and its consequences in combinatorial optimization, v.1, n.2, p. 169–197, 1981.

[26] GOLDBERG, A. V.; TARJAN, R. E. **Journal of the ACM**. A new approach to the maximum-flow problem, v.35, n.4, p. 921–940, 1988.

[27] BERTHOLD, T. **Primal heuristics for mixed integer programs**. 2006. Master's thesis - Technische Universitat Berlin.

[28] KARLOF, J. K. **Integer programming: theory and practice**. Boca Raton, FL, USA: CRC Press, 2005.

[29] CHVATAL, V. **Linear Programming**. Series of books in the mathematical sciences. W.H. Freeman, 1983.

[30] LUENBERGER, D. G. **Introduction to linear and nonlinear programming**, volume 28. Reading, MA, USA: Addison-Wesley, 1973.

[31] WOLSEY, L. **Integer Programming**. Wiley Series in Discrete Mathematics and Optimization. New York, NY, USA: Wiley, 1998.

[32] WONG, R. T. **Mathematical programming**. A dual ascent approach for Steiner tree problems on a directed graph, v.28, n.3, p. 271–287, 1984.

[33] CLAUS, A.; MACULAN. **Une nouvelle formulation du probleme de Steiner sur un graphe**. Technical report, Montréal: Université de Montréal, Centre de recherche sur les transports, 1983.

[34] ACHTERBERG, T.; KOCH, T. ; MARTIN, A. **Operations Research Letters**. Branching rules revisited, v.33, n.1, p. 42–54, 2005.

[35] GLOVER, F.; KOCHENBERGER, G. A. **Handbook of metaheuristics**. Norwell, MA, USA: Springer Science & Business Media, 2003.

[36] BLUM, C.; ROLI, A. **ACM Computing Surveys**. Metaheuristics in combinatorial optimization: Overview and conceptual comparison, v.35, n.3, p. 268–308, 2003.

[37] LOURENÇO, H. R.; MARTIN, O. C. ; STÜTZLE, T. **Iterated local search**. Norwell, MA, USA: Springer, 2003.

[38] CHVATAL, V. **Mathematics of operations research**. A greedy heuristic for the set-covering problem, v.4, n.3, p. 233–235, 1979.

[39] VAZIRANI, V. V. **Approximation Algorithms**. New York, NY, USA: Springer-Verlag Inc., 2001.

[40] RAIDL, G. R. **A unified view on hybrid metaheuristics**. In: Hybrid Metaheuristics, p. 1–12. Springer, 2006.

[41] KARP, R. M. **Reducibility among combinatorial problems**. Yorktown Heights, NY, USA: Springer, 1972.

[42] OPTIMIZATION, I. G. **Gurobi Optimizer Reference Manual**. 2015. Disponível em: <http://www.gurobi.com>.

[43] DE ARAGÃO, M. P.; UCHOA, E. ; WERNECK, R. F. **Electronic Notes in Discrete Mathematics**. Dual heuristics on the exact solution of large Steiner problems, v.7, p. 150–153, 2001.