Thorsten Bonato

Contraction-based Separation and Lifting for Solving the Max-Cut Problem

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Bonato, Thorsten:
Contraction-based Separation and Lifting for Solving the Max-Cut Problem
ISBN 978-3-941274-86-0

1. Edition 2011, Göttingen

URL: www.optimus-verlag.de

Printed in Germany
Paper is FSC certified (wood-free, chlorine free and acid-free, and resistant to aging ANSI 3948 and ISO 9706)

Preface

The max-cut problem consists in partitioning the nodes of an undirected weighted graph into two sets such that the aggregate weight of the edges between these sets is maximized. This well-known combinatorial optimization problem is the reformulation, in graph theoretical terms, of the unconstrained 0/1 quadratic problem which aims at optimizing a quadratic objective function over the set of all 0/1 vectors of fixed dimension. In general, the max-cut problem is **NP**-hard, although selected special cases can be solved in polynomial time. It has a number of interesting applications such as the optimal design of very-large-scale-integration (VLSI) circuits or the study of minimum energy configurations of spin glasses—alloys of magnetic impurities diluted in a nonmagnetic metal—which is among the most investigated topics in the statistical physics literature.

The present book is concerned with finding provably optimal solutions of the max-cut problem as opposed to approximate solutions. To do so, we use the established and well-working branch-and-cut method, a generic solution technique whose performance for a given type of optimization problem is mainly determined by two key elements: Firstly, a close yet manageable approximation of the polyhedron associated with the problem. Secondly, efficient methods to solve the corresponding separation problem which is to decide for an arbitrary point in the ambient space whether or not it lies inside the polyhedron just mentioned.

For complete graphs, the max-cut problem and the associated cut polytope have been extensively studied over the last decades. Their counterparts on arbitrary graphs, in particular sparse ones, on the other hand, have received much less attention. Moreover, the transferability of methods from the complete to the sparse case is limited. This is mainly because the respective methods often require certain structures that are unlikely to be found in a sparse graph. A

generic possibility to work around this problem is to make the graph artificially complete by adding zero-weighted edges. However, this technique is only effective in conjunction with an efficient way to exploit the original sparse structure. Otherwise, it will ultimately lead to the same computational complexity as the problem on the complete graph.

In this study, we investigate a new contraction-based separation approach for the max-cut problem that is primarily intended for problems on sparse graphs. The key idea is to contract edges based on their value in a given linear programming (LP) solution. In its simplest form, this technique presents an efficient way to separate so-called odd-cycle inequalities. In addition, we describe sophisticated methods to add missing edges to an already contracted graph as well as to compute suitable values to extend the corresponding LP solution accordingly. This allows us to apply solution techniques that were originally intended for problems on complete graphs and could not have been used on a sparse graph otherwise.

The book is structured as follows: In the first chapter, we introduce fundamental concepts, methods, and results from the fields of graph theory, complexity theory, linear and integer programming, as well as combinatorial optimization.

In Chapter 2 we precisely define the max-cut problem, including its reformulation in terms of quadratic optimization. We proceed with the description of two interesting applications coming from circuit layout design and statistical physics, respectively. Following a brief introduction to approximate solution techniques, we give a survey of the associated cut polytope and its facial structure. These combined results will help in devising a branch-and-cut algorithm later on. Finally, we outline the relevant literature and previous work on the max-cut problem.

Our key contribution, the new shrink separation approach, is presented in Chapter 3. Here, we elaborate on the single steps of the method and their respective underlying theory. Afterwards, we describe an actual realization of the shrink separation and point out

its deviations from the theoretical conceptual design. Finally, we investigate some of the algorithm's numerical aspects.

Chapter 4 deals with the computational experiments that we carried out to test the performance of the shrink separation. After introducing the considered test instances, we specify the experiments' setup, including the hard- and software used, chosen parameters, and tested separation scenarios. We proceed by summarizing the results for the different classes of test instances before concluding with a case study that takes an in-depth look at a particularly interesting set of instances generated from real-world data.

The last chapter comprises a recapitulation of our contributions and findings in this study, followed by our conclusions and some suggestions regarding future research directions.

Finally, Appendices A and B contain the collective tables with detailed information on the characteristics of the test instances and the results of the computational experiments, respectively.

Author's Note

This book is a revised version of my doctoral thesis submitted and defended at the University of Heidelberg in 2011. Though the technical content is essentially the same as the submitted version, several aspects have been improved. In particular, Section 3.1.3 has been reworked and extended significantly. Also, the computational results have been updated, misprints and minor errors fixed, and the writing polished.

Acknowledgment

Many people contributed to this work. Above all I want to thank my thesis advisor Prof. Dr. Gerhard Reinelt for his support and guidance over the years. Also, I am grateful to Dr. Giovanni Rinaldi, who was always glad to share his expertise in general as well as his insights into the max-cut problem in particular.

Thanks to the members of the Discrete and Combinatorial Optimization Group at the University of Heidelberg for providing a pleasant working environment. In particular, I would like to thank Dr. Marcus Oswald for his ability to convey complex concepts and for always being cheerful, enthusiastic, and supportive. Furthermore, I owe respect and gratitude to Stefan Wiesberg for all the effort he put into proofreading the entire original version of the thesis as well as parts of the revised one; of course, I take full responsibility for any remaining errors. Finally, Dr. Hanna Seitz with her talent for encouraging and motivating people was always a pleasure to work with. I appreciate our conversations and her help in putting large problems into perspective.

I would like to thank PD Dr. Frauke Liers, Prof. Dr. Christoph Buchheim, and Dr. Marcus Oswald for sharing their insights into target cuts and for providing their target cut software framework.

Thanks to the members of the Operations Research Group at the University of Klagenfurt for the great time I spent with them. I owe special gratitude to the head of the group, Prof. Dr. Franz Rendl, as well as to Dr. Angelika Wiegele for their expertise and their support.

Last but certainly not least, my heartfelt thanks go to my parents Gisela and Adolf Bonato to whom I am indebted the most.

Heidelberg, September 2011 Thorsten Bonato

Contents

1. Preliminaries and Notations

In this chapter, we introduce the fundamental concepts and general terminology used throughout this book. Definitions with a more restricted scope are provided in subsequent chapters. Some elementary definitions are given to fix the terminology and to make the presentation more self-contained.

1.1. Graphs

The present book deals with problems that are defined on graphs. Various forms of graphs are also encountered in the solution approaches to these problems. We now introduce selected topics of graph theory. The statement below is mainly adopted from the introductory chapters of [Die05, GY04].

1.1.1. Nodes, Edges, and Density

An **undirected graph** is a pair $G = (V, E)$ consisting of a nonempty set V of **nodes** and a set E of **edges** which are unordered pairs of nodes. Unless otherwise stated, we generally assume the graphs in this book to be undirected. Therefore, we will omit the term "undirected" from now on. A **weighted graph** additionally associates a **label**, or **weight**, with every edge in the graph. Weights are usually real numbers.

The node set of a graph G is referred to as $V(G)$, its edge set as $E(G)$. These notations are independent of any actual names the sets may have in a given context. We denote an edge $e = \{u, v\}$ by uv. Also, we always equate a node v and the respective 1-element set $\{v\}$.

The cardinality of the node set V, written as $|V|$, is called the **order** of G. A graph of order n can have at most $\binom{n}{2}$ edges, in

which case we call the graph **complete** and denote it by K_n. The ratio $|E|/\binom{n}{2}$ of actual and potential edges is called the **edge density**, or simply the **density**, of G. Graphs with a density near 0 and 1 are referred to as **sparse** and **dense**, respectively. However, the decision whether a given graph is considered to be sparse or dense is not absolute; it usually depends on both the context and the type of graph at hand.

A graph is called **finite** if both V and E are finite. We will exclusively deal with finite graphs.

1.1.2. Incidence and Adjacency

A node v is called **incident** with an edge e, and vice versa, if $v \in e$. The two nodes incident with an edge are its **ends**, and an edge **joins** its ends. Two nodes are **adjacent**, or **neighbors**, if they are joined by an edge. Two distinct edges $e \neq f$ are **adjacent** if they share a common end.

An edge joining a node with itself is called a **loop**. A collection of at least two edges sharing the same ends is referred to as **multiple edge** or **parallel edge**. A graph without loops or multiple edges is called **simple**.

1.1.3. Cuts and Degree

Let $U, W \subseteq V$ be two node sets. We call the set of edges with precisely one end in U a **cut** of G and denote it by $\delta(U)$. In this context, the set U and its **complement** $U^{\complement} := V \setminus U$ are referred to as the **shores** of the cut. We write $\delta(v)$ instead of $\delta(\{v\})$ for a node $v \in V$ and call $\delta(v)$ the **star** of v. We will also use the abbreviation $(U : W) := \delta(U) \cap \delta(W)$ for the set of edges with one end in U and the other end in W.

The **degree** $\deg(v)$ of a node v is the number of edges incident with v, loops counting twice. Thus, for graphs without loops, the degree of v is identical to $|\delta(v)|$.

1.1.4. Paths, Cycles, and Connectivity

A **path** is a nonempty graph $P = (V, E)$ with pairwise distinct nodes $V = \{v_i \mid i = 0, \dots, k\}$ and the edges $E = \{v_i v_{i+1} \mid i = 0, \dots, k-1\}$. The nodes v_0 and v_k are **linked** by P and are called its **ends**. The remaining nodes $v_1, \dots, v_{k-1}$ are called the **inner nodes** of P. A path is often referred to by the sequence of its nodes, i. e., $P = v_0 v_1 \dots v_k$, and is called a path **from** v_0 **to** v_k or simply a (v_0, v_k)**-path**. The number of edges of a path is its **length**. In weighted graphs, however, the length of a path commonly refers to the aggregate weight of its edges rather than their number.

Let $P = v_0 \dots v_{k-1}$ be a path of length at least 2. Then, we obtain a so-called **cycle** by adding the edge $v_{k-1}v_0$. As with paths, a cycle is often referred to by the (cyclic) sequence of its nodes, e. g., the above cycle could be written as $C = v_0 \dots v_{k-1} v_0$. The **length** of a cycle is the number of its edges (or nodes) and a cycle of length k is also called a k**-cycle**. A **chord** of a cycle C is an edge that joins two nodes of C which are not adjacent in the cycle.

Two nodes u, v in a graph G are **connected** if they are linked by a path in G. The graph G itself is **connected** if this is true for any two of its nodes. The **distance** between two nodes u and v in a graph G is the length of a **shortest** (u, v)**-path** in G; if no such path exists, we set the distance to infinity.

1.1.5. Subgraphs and Contractibility

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. If $V' \subseteq V$ and $E' \subseteq E$ then G' is a **subgraph** of G (and G a **supergraph** of G'), written as $G' \subseteq G$.

If $G' \subseteq G$ and G' contains all the edges $uv \in E$ with $u, v \in V'$ then G' is an **induced subgraph** of G. We say that V' **induces** or **spans** G' in G and write $G' = G[V']$. Thus, for any node set $U \subseteq V$, the **(node-)induced subgraph** $G[U]$ is the graph on U whose edges are exactly the edges of G with both ends in U. Finally, $G' \subseteq G$ is a **spanning subgraph** of G if V' spans all of G, i. e., if $V' = V$.

A **connected component** of G is a connected subgraph of G which is maximal with respect to edge inclusion. A complete subgraph of G is called a **clique** of G.

The operation of identifying a pair of adjacent nodes—while preserving all other adjacencies between nodes—is referred to as **elementary contraction**, or simply as **contraction**. We assume that multiple edges arising from a contraction are replaced by single edges. A graph G is called **contractible** to another graph G' if the latter can be obtained from G by a sequence of contractions.

1.1.6. Trees and Forests

An **acyclic** graph, i. e., a graph not containing any cycles, is called a **forest**. A connected forest is called a **tree**. Thus, a forest is a graph whose connected components are trees. The nodes of degree 1 in a tree are its **leaves**, the remaining nodes are its **inner nodes**.

Sometimes it is convenient to consider one node of a tree as special. Such a node is called the **root** of this tree. Note that the root of a tree is never called a leaf, even if it has degree 1. A tree with a fixed root is called a **rooted tree**.

In a rooted tree T, the nodes at distance k from the root have **height** k and form the k-th **level** of T. The root has height 0. The **height** of T itself is the maximum height of its nodes. The nodes on level $k+1$ which are adjacent to a node v on level k are called the **children** of v and v is called the **parent** of its children. We refer to children of the same parent as **siblings**. If every node in T has at most k children, $k \geq 2$, we call it a k**-ary** tree. In case of a 2-ary tree, we use the term **binary tree** instead.

1.1.7. Embeddings and Genera

Let S_k denote the **orientable surface** formed by adding k **handles** to the sphere. The sphere itself is denoted by S_0. It is topologically equivalent to the flat plane since we can create a hole in the sphere's surface and then stretch out the surface onto the plane. The **torus**

is S_1, the **double-torus** S_2, and so on. The **genus** of S_k is the number of handles, k.

An **embedding** of a graph into a surface is a drawing of the graph on the surface in such a way that its edges may intersect only at their ends. In other words, the graph can be drawn on the surface without any edges crossing. The **genus** of a graph is the minimum integer g such that the graph can be embedded into S_g.

1.1.8. Selected Classes of Graphs

A graph $G = (V, E)$ is **bipartite** if its node set V can be partitioned into two nonempty subsets V_1 and V_2 such that each edge has one end in V_1 and the other end in V_2. G is **complete bipartite** if each node in V_1 is joined to each node in V_2. We denote the complete bipartite graph with $|V_1| = m$ and $|V_2| = n$ by $K_{m,n}$.

A graph is **planar** if it can be drawn in a plane without any edges crossing, i. e., if it has genus 0. We call a nonplanar graph G **almost planar** if it contains a node v such that G becomes planar by removing v and all its incident edges.

A graph is k**-regular** if all its nodes have the same degree k. A 3-regular graph is called **cubic**.

A **grid graph** is a graph that can be mapped to a grid, i. e., each node corresponds to a grid point and each edge corresponds to a tie between the respective grid points of its ends.

1.2. Affine Geometry and Polyhedra

A basic knowledge of polyhedral theory is essential to understand the later chapters. We now give an overview of the relevant concepts of affine geometry and polyhedral theory. It is mainly based on the introductory chapters of [Brø83, Zie06]. We assume familiarity with the standard linear theory of the **real vector space** $\mathbb{R}^d$, in particular basic notions such as **subspaces**, **linear independence**, **dimension**, **scalar product**, and so forth. As a convention, vectors