Max 2SAT-3, Net, Euclidea: Techniques and Results in Computational Inapproximability

by

Victor Luo

B.S. Computer Science and Engineering and Mathematics, MIT, 2023

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

© 2024 Victor Luo. This work is licensed under a CC BY-NC-ND 4.0 license.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Victor Luo
Department of Electrical Engineering and Computer Science
May 17, 2024

Certified by: Erik D. Demaine
Professor of Electrical Engineering and Computer Science, Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Max 2SAT-3, Net, Euclidea: Techniques and Results in Computational Inapproximability

by

Victor Luo

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

This Master's thesis investigates three diverse problem domains through the lens of computational inapproximability: Max 2SAT-3, the *Net* tile-rotating puzzle family, and the mobile game *Euclidea*.

Max 2SAT-3 is a problem long known to be APX-complete, but finding a clear proof is harder than one might expect. We examine the history of Max 2SAT-3, addressing past misconceptions and clarifying where the reduction chain has been opaque, and present a novel proof of its APX-completeness.

Net variants form a wide class of puzzles with lots of potential for future research. We introduce a natural optimization variant of *Net* and demonstrate its inapproximability, as well as consolidate existing findings and present other new results.

Euclidea is a mobile game based on Euclidean straightedge-and-compass constructions. We define the game as an optimization problem and establish its APX-hardness, as well as discuss challenges in upper-bounding its complexity, relating to current knowledge gaps regarding the constructible and algebraic numbers.

Thesis supervisor: Erik D. Demaine
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

First and foremost, all chapters in this work are based on results initiated during open problem solving in the MIT class Algorithmic Lower Bounds: Fun with Hardness Proofs (6.5440), taught by Erik Demaine in Fall 2023. In Chapter 2, the process of literature review relied heavily on Erik's notes for this course, especially the notes on inapproximability, accessible as Fall 2023 Lectures 17, 18, and 19. I am grateful not only to those who collaborated with me on these works, but to all participants of the class, for all the helpful discussions and providing an inspiring and supportive atmosphere to do research in.

Thank you, Erik, for working with me this year, coaching me through my thesis proposal and through when I eventually resigned and switched my thesis topic. You really are an immense inspiration both as a person, and to the part of me which really, really wanted to pursue research while keeping software in my life. This work would not be possible without the collaborative problem solving environments you brought to MIT, as well as the Coauthor, Cocreate, and Comingle suite. The Co-suite made working on problems in geometry and other visual-heavy fields very easy to do remotely, to the point where I found it one of the most fun things I could do over the pandemic. Almost all the custom figures in this work, in fact, are generated in Cocreate, software relating to which I had hoped to work on for my Master's thesis. I apologize for the situation this semester and for abruptly leaving for the software industry, though I had been planning to do so eventually. I hope we can find opportunities to continue working together in the future!

I would also like to acknowledge all the other people who have supported me through my thesis, through my college years, and throughout my life.

Thank you to all the friends I made over my years at MIT, even the many of you I fell out of touch with.

Thank you, ET—through all the good times, the bad times, the times when I come home after an awful day and dinner's on the table, and the times when I come home after an awful day and realize I forgot I needed to scrub 6 toilets, I've realized that living at ET is something I couldn't have lived without.

Thank you, Alex, for showing me *Puzzle Pirates*, among other things. Section 3.4 in particular would not be possible without you.

Thank you, osu!mit, for being the one place where I could be myself the most. I've worked hard over the last five years to make that less and less the case, but somehow it's still true.

Finally, a special thank you to Mom, Dad, and Nash. Thank you, Mom, for always being the first to understand me, and the first to give me real advice. Thank you, Dad, for giving me my love of puzzles and hard problems, and always reminding me just how cool science is. Thank you both, for the true home-cooked meals I can't get anywhere else. Thank you, Nash, my brother, for being cool, I suppose. More than just as a musician, I really do look up to you in many ways, now.

Contents

Chapter 1

Introduction

When a student learns in an intro complexity theory course that the Max Cut and Clique problems are both NP-complete, they might be tempted to chalk them up as roughly the same. Max Cut and Clique are both graph problems that take in a graph G and target parameter k, and it's NP-complete to determine whether G has "the thing you want" of size at least k. When the student's parents ask what they're studying, they may also simplify a little: "There's this class of problems called 'NP-complete', and you can show that any NP-complete problem is just as hard to solve as any other."

However, when stepping into the world of approximation and inapproximability, Max Cut and Clique start to look very different. The Max Cut problem is approximable to a ratio of about 0.878 [1], meaning that there's a polynomial-time algorithm that produces a cut that's at least 87% of the optimal value. Meanwhile, the Clique problem is Poly-APX-complete, meaning that unless P = NP, there is no polynomial-time algorithm that can even consistently come within a factor better than $O(n^{1-\epsilon})$ of optimal, for any $\epsilon > 0$ [2].

Serious study of inapproximability dates back to at least 1991, when Papadimitriou and Yannakakis defined a new approximation complexity class and showed a list of problems complete for it [3]. Even within the realm of NP optimization problems, the corresponding decision problems of which lie in NP, there are plenty of complexity classes for problems to fall into. For this reason, the field of inapproximability is only growing, and it continues to have many interesting open problems to offer.

In this work, we present a collection of novel results relating to the computational hardness of approximation. We also include some other computational complexity results related to the specific topics discussed in this work, as well as bring fresh explanations to existing results, where needed.

1.1 Inapproximability Background

This section will serve as a brief guide to the various types of approximation hardness and reductions used in this work.

In an *optimization problem*, potential solutions to an input are either rejected as invalid

or accepted as valid, in which case the solution is assigned a *cost*, a score of how good or bad the solution is. The optimization problem then essentially asks: given an input, can you find a solution with the highest (or lowest) cost? The optimal cost for an instance x of the problem is widely denoted $\mathrm{OPT}(x)$.

NP optimization problems (NPO) are a broad class consisting of optimization problems where all the problem functions run in polynomial time, and the *decision problem* of whether $\mathrm{OPT}(x)$ is at least (or at most) a threshold k is in NP.

APX is the subset of NPO which can be approximated to a constant factor in polynomial time. This is the class we will concern ourselves the most with in this work.

A PTAS, or Polynomial-Time Approximation Scheme, is an polynomial-time algorithm to approximate a problem to within a ratio of $(1 + \epsilon)$, for any $\epsilon > 0$. Our work often deals with APX-hardness, one of the implications of which is that no PTAS exists for the problem unless P = NP.

A problem is APX-hard if there is a PTAS-reduction from any APX problem to it. An L-reduction is a stricter form of PTAS-reduction; these reduction types will be explained in more detail where used in this work.

Finally, an c-gap version of an optimization problem is the decision problem of deciding between $\mathrm{OPT} \leq k$ and $\mathrm{OPT} > c \cdot k$, given that one of the two cases must be true. Gap hardness is mentioned in a few sections, but primarily used only in Chapter 3.

1.2 Overview of Chapters

In Chapter 2, we present a novel proof of the APX-completeness of Max 2SAT-3, which is a foundational APX-complete problem used in many reductions as the most restricted form of the Max nSAT-k family. We also present a history of the problem, as well as an analysis of how mistakes in the reduction chain have occurred, as presentations of the reduction chain of Max 2SAT-3 have often been confusing and convoluted in the literature.

In Chapter 3, we define a natural optimization variant of *Net*, a popular tile-rotating puzzle type, and prove inapproximability with a gap-producing reduction. Under a further variation where most pieces can be rearranged arbitrarily, we prove a perhaps surprising decision NP-hardness result. We also collect existing results on *Net* variants into a table, and present an insightful approach to an alternate proof for one of the results.

Finally, in Chapter 4, we present a novel analysis of the game *Euclidea*, which revolves around performing Euclidean constructions in a minimum number of steps. We prove that the game itself is APX-hard under this natural optimization target, for two definitions of a Euclidean construction step. We also discuss challenges in upper-bounding the complexity of *Euclidea*, with applications reaching into the theory of constructible and algebraic numbers.

References

[1] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *Journal of the ACM*, vol. 42, no. 6, pp. 1115–1145, Nov. 1995, ISSN: 0004-5411, 1557-735X. DOI: 10.1145/227683.227684. URL: https://dl.acm.org/doi/10.1145/227683.227684.

[2] J. Håstad, "Clique is hard to approximate within $n^{1-\epsilon}$," *Acta Mathematica*, vol. 182, no. 1, pp. 105–142, 1999, ISSN: 0001-5962. DOI: 10.1007/BF02392825. URL: http://projecteuclid.org/euclid.acta/1485891205.

[3] C. H. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," *Journal of Computer and System Sciences*, vol. 43, no. 3, pp. 425–440, 1991, ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(91)90023-X. URL: https://www.sciencedirect.com/science/article/pii/002200009190023X.

Chapter 2

Max 2SAT-3

The proof that Max 2SAT is APX-complete[1] dates back to Papadimitriou and Yannakakis' seminal 1991 paper on approximation algorithm hardness [1]. The same paper also shows that Max 3SAT-k is APX-complete, for a fixed constant k.

In the time since, it has been shown and is widely accepted that Max 2SAT-3 is APX-complete—the strongest result one can hope for. However, the reduction can split up in a few different ways, and the parts are often scattered across the literature or sometimes left as exercises to the reader. In addition, one step of the reduction has caused confusion in the past—the reduction from Max 2SAT-k to Max 2SAT-3.[2]

In this chapter, we provide a resource showing the APX-completeness of Max 2SAT-3, focusing on this difficult step of the reduction. In Section 2.1, we briefly explain some preliminaries, and discuss the Max 2SAT-k to Max 2SAT-3 reduction step. In Section 2.2, we give an overview of previous works related to these results, and discuss how the Max 2SAT-k to Max 2SAT-3 reduction has historically caused confusion. In Section 2.3, we discuss the techniques that previous proofs of APX-hardness have used, and present a new, simple L-reduction from Max 2SAT-k to Max 2SAT-3, inspired by the core ideas of these techniques.

This chapter represents joint work with Zi Song Yeoh, Lily Chung, Erik Demaine, and others in the MIT class Algorithmic Lower Bounds: Fun with Hardness Proofs (6.5440), taught by Erik Demaine in Fall 2023.

2.1 Preliminaries

Just as reducing from 3SAT remains a very popular choice for complexity theorists to prove problems NP-hard, researchers of approximation hardness often reduce from Max 2SAT or Max 3SAT to prove problems APX-hard. It is often helpful in such reductions for a variable's number of occurrences to be upper-bounded by a constant k, which gives

[1] MAX SNP-complete in the original work.

[2] As discussed in Section 2.1, equivalent reductions are often historically presented as from Max 3SAT-k to Max 3SAT-3.

rise to Max 2SAT-k or Max 3SAT-k. When it matters that the number of occurrences is bounded but the bound itself doesn't matter, this can also be called Max 2SAT-$O(1)$ or Max 3SAT-$O(1)$.

The two main approximation types we will consider in this chapter will be PTAS-reductions [2] [3] and L-reductions [1]. PTAS-reductions are the basis of the definition of APX-hardness, so at the core, we will be analyzing whether certain reductions meet the definition of a PTAS-reduction. The reductions we present will also be L-reductions; introduced earlier, they are a strengthening of PTAS-reductions, and also remain one of the most popular reduction types for showing APX-hardness.

We will mainly focus on analyzing the steps which reduce from Max 2SAT-k or Max 3SAT-k to the equivalent problem with a smaller constant k, as these are the reduction steps which have been the most varied and posed the most challenges.

Many of these reductions generally rely on adding clauses of size 2, and as such, can be used to reduce from Max 2SAT-$O(1)$ or Max 3SAT-$O(1)$. Throughout the remainder of this chapter, we will refer to these reductions in terms of Max 2SAT, except in instances where the reductions were used historically for Max 3SAT.

2.2 History of Max nSAT-k reductions

First, we list some of the past works that have showed APX-hardness results for 2SAT and 3SAT; we go into more detail on the techniques used in Section 2.3.1. Then, we explain how an easily-made assumption can lead to a fallacious reduction from Max 2SAT-$O(1)$ to Max 2SAT-5 or Max 2SAT-3, and discuss how this has occurred in the past.

2.2.1 Summary of Past Work

One of the earliest known works in this field is Papadimitriou and Yannakakis' 1991 paper [1], where they define the L-reduction and demonstrate many problems that are L-reducible from Max SAT.[3] As part of these results, they showed Max 3SAT to be APX-complete, as well as Max 3SAT-$O(1)$. They reduce directly from Max 3SAT to Max 3SAT-8, using expander graphs and a clever argument to weaken the expander graph requirements.[4]

The proof that Max 2SAT is APX-complete also dates back to [1]. First, they reduce from Max 3SAT-k to Max-degree-$k+1$ Independent Set, for any bound k on variable occurrences. The reduction to Max 2SAT again increases the bound by 1, from Max-degree-k Independent Set to Max 2SAT-$k+1$. Though not explicitly stated, combined with their reduction to Max 3SAT-8, these reductions directly show APX-completeness for Max 2SAT-10.

As early as 1994, the textbook *Computational Complexity* [4], also written by Papadimitriou, gave a proof of Max 3SAT-3 APX-completeness in Theorem 13.10. Using the same

[3] Which we now know to be equivalent to APX-completeness.

[4] The authors claim that the construction can reduce to Max 3SAT-6, with more care in connecting the cubic expander graph to the binary trees.

reduction chain from [1], they explicitly claim Max 2SAT-5 APX-complete as well. The proof is very dense, but it uses similar expander graph requirement weakening, as well as defining an *amplifier*, which corresponds fairly closely to our *designated directed expander graph*, defined in Section 2.3.1. The textbook, however, uses non-constructive arguments to show that amplifiers exist, and relies on external sources to prove that an algorithm to generate them in $\log(n)$ space. As such, the last part of the argument becomes very difficult to follow.

A 1997 paper by Trevisan [5] was the first we could find to not only claim that Max 2SAT-3 is APX-hard, but also to reduce from it. As a combination of previous results, the paper proves this claim almost in passing—the justification is "(apply to Max 2SAT the reduction from Max 3SAT to Max 3SAT-3 described in [4])."

In the 1998 paper *A Threshold of ln n for Approximating Set Cover* [6], Feige shows as an intermediate result that E3SAT-E5 is *gap inapproximable*, meaning that $(1-\epsilon, 1)$-gap E3SAT-E5 is NP-hard for some $\epsilon > 0$. As we will discuss in Section 2.2.3, this result and proof can be easily mistaken at a glance to fill in the missing link of Max 3SAT-$O(1)$ to Max 3SAT-5, especially after [7] left it as an exercise to the reader. The key difference is that the reduction is only a *gap reduction*, and as we will see, the difference is crucial.

Ausiello's 1999 textbook, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties* [7] reduces from Max 3SAT to Max 3SAT-3 in three parts: Max 3SAT $\rightarrow$ Max 3SAT-29 $\rightarrow$ Max 3SAT-5 $\rightarrow$ Max 3SAT-3. They present an L-reduction from Max 3SAT to Max 3SAT-29 using expander graphs, and from Max 3SAT-5 to Max 3SAT-3 via explicit construction; however, the reduction from Max 3SAT-29 to Max 3SAT-5 is left as an exercise. Despite this fact, the textbook is a commonly-cited resource for the APX-completeness of Max 3SAT-3, including a citation on the Wikipedia page for the problem [8].

Finally, Berman and Karpinski showed in 1999 that Max E2SAT-E3 is gap inapproximable [9]. This is the most specific possible class one can hope to prove hardness results for in the nSAT-k family. However, as well as using a different type of reduction, gap inapproximability is a weaker result than APX-completeness—although the former also implies that a PTAS cannot exist, it does not imply the latter. Their approach passes through Max-degree-3 Max Cut, the inapproximability proof of which also uses techniques similar to expander graphs.

To summarize, although each work has its own merits and most aimed to prove other results, none of these works have demonstrated a correct, complete, and clear APX-hardness reduction.

2.2.2 Erroneous Reduction Example

The specific step in the reduction chain that has proved difficult is that of reducing from Max 2SAT, or Max 2SAT-k with a large constant k, down to Max 2SAT-k with a small constant, such as 5 or 3. Mistakes in this reduction mostly revolve around proving that when splitting up a variable into multiple copies, the optimum for any particular instance can be achieved when each copy takes the same truth value in the resulting graph. If this is

not the case, then the definition of an L-reduction, or even a PTAS-reduction, may not be satisfied.

To illustrate this point, we will first present an example of such a reduction, and then discuss a few examples where it has appeared.

Theorem 2.2.1. *For all $B > 3$, there exists an L-reduction from Max 2SAT-B to Max 2SAT-3.*

Proof (erroneous). Given a 2CNF-B formula ϕ, we will first define the corresponding 2CNF-3 formula ϕ'. Consider a variable x in ϕ which occurs k times. Replace x with k separate variables $\{x_1, \ldots, x_k\}$, and add clauses $(\neg x_i \vee x_{i+1})$ for $1 \leq i \leq k$ (where $x_{k+1} = x_1$). This forms a chain of implications, as the clauses are logically equivalent to $(x_i \to x_{i+1})$. An example is illustrated by the diagram in Figure 2.1.

Now, given assignment A to ϕ, and corresponding assignment A' to ϕ', let $\text{Score}_{\text{2SAT-}B}(\phi, A)$ and $\text{Score}_{\text{2SAT-3}}(\phi', A')$ denote the number of clauses satisfied by A and A' in their respective problems. By construction, all x_i for a particular variable x take on the same value due to the cycle of implications in ϕ'. Thus, if ϕ has N clauses, then ϕ' has $N' \leq 3N$ clauses,[5] and

$$\text{Score}_{\text{2SAT-3}}(\phi', A') = \text{Score}_{\text{2SAT-}B}(\phi, A) + N' - N.$$

The error in this proof has already been made, so we can now show the L-reduction properties. OPT for a 2SAT instance, or any CNF formula, is always at least 1/2 the number of clauses, as each clause is either satisfied by the all-True or all-False input.[6] Thus,

$$\begin{aligned}
\text{OPT}_{\text{2SAT-3}}(\phi') &= \text{OPT}_{\text{2SAT-}B}(\phi) + N' - N \\
&\leq \text{OPT}_{\text{2SAT-}B}(\phi) + 2N \\
&\leq 5 \cdot \text{OPT}_{\text{2SAT-}B}(\phi) \\
&= O(\text{OPT}_{\text{2SAT-}B}(\phi)),
\end{aligned}$$

and for a pair of assignments A, A',

$$\begin{aligned}
&\text{OPT}_{\text{2SAT-}B}(\phi) - \text{Score}_{\text{2SAT-}B}(\phi, A) \\
&= (\text{OPT}_{\text{2SAT-3}}(\phi') - N' + N) - (\text{Score}_{\text{2SAT-3}}(\phi', A') - N' + N) \\
&= O(\text{OPT}_{\text{2SAT-3}}(\phi') - \text{Score}_{\text{2SAT-3}}(\phi', A')),
\end{aligned}$$

showing both necessary big O relations. □

Unfortunately, this is not a valid L-reduction, because we need to account for *many* possible assignments A' corresponding to the same A, and provide a well-defined function

[5] Each clause in ϕ accounting for itself, and one link in the variable cycle for each of its (up to 2) variables.

[6] If given an EnSAT instance, where each clause has exactly n distinct variables, then we can use an expected value argument. Each clause is only unsatisfied $1/2^n$ of the time, so the average assignment satisfies $1 - (1/2^n)$ of the clauses, and OPT must be at least that much.

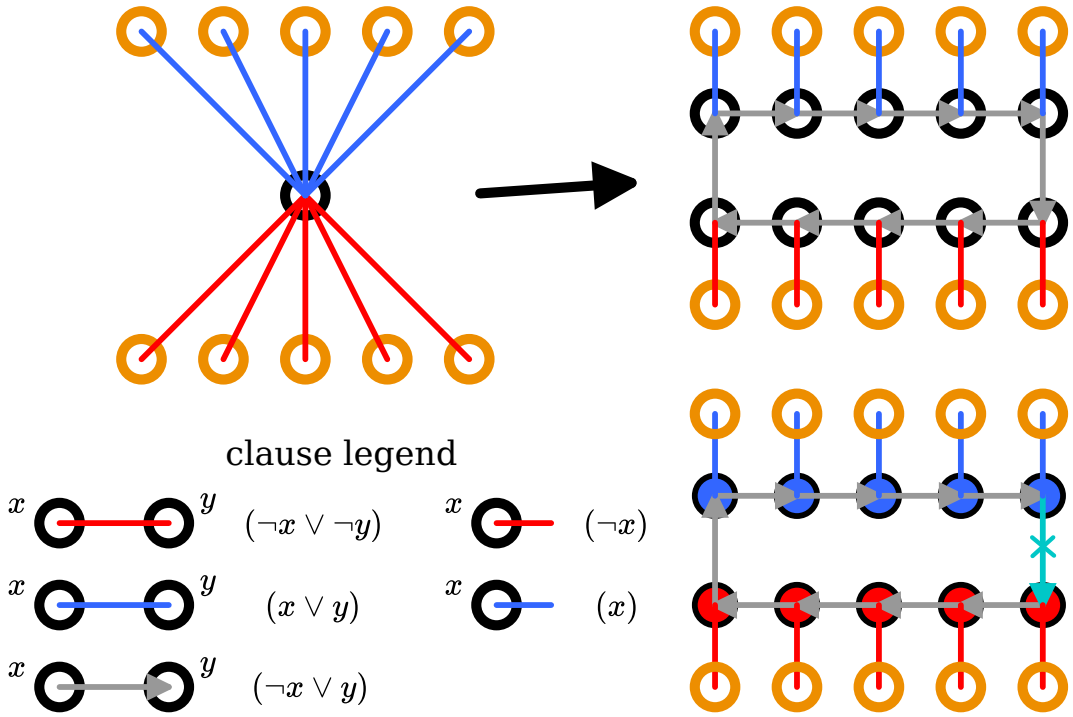

Figure 2.1. Illustration of an erroneous reduction from 2SAT-$O(1)$ to 2SAT-3. Bottom left: clause legend for this figure and future figures in this chapter. A one-variable clause is denoted with a segment connecting only to that variable. Gray arrows represent clauses logically equivalent to an implication ($x \to y$). Top: the original variable x on the left has been replaced with many copies x_i on the right, each appearing in one of the clauses that x was a part of. These copies x_i are connected in a cycle of implications. Bottom right: if we assign all x_i's on the top row true and all x_i's on the bottom row false, we can satisfy all but one clause in the cycle. However, if we must pick a truth value to use for all x_i, up to 5 clauses may be unsatisfied.

for this correspondence. The simplest such function might be: given an assignment A' to ϕ', we construct A by taking a majority vote of the truth values of x_i's, breaking ties arbitrarily.

If alternate values for A' perform worse than the "canonical" one we assumed in the erroneous proof, then there is no harm done to the proof—OPT will not change, and the error $\text{OPT}_{\text{2SAT-3}}(\phi') - \text{Score}_{\text{2SAT-3}}(\phi', A')$ can only be greater than that for the "canonical" A'.

However, we will demonstrate that this is not the case for using the majority function to map assignments A' back to A—an optimal solution to ϕ' will be mapped to a non-optimal solution to ϕ. We will show this by concrete example with the majority function to demonstrate the concept, and it should be clear that any other choice of function $A' \mapsto A$ is also susceptible to the same issue.

Our counterexample ϕ uses 4 variables, x_1, x_2, x_3, x_4, and 16 clauses:[7]

- (x_i) and $(\neg x_i)$, for each $i \in \{1, 2, 3, 4\}$,
- 3 copies each of $(\neg x_1 \vee \neg x_3)$ and $(\neg x_2 \vee \neg x_4)$
- 1 copy of $(x_1 \vee x_2)$ and $(x_3 \vee x_4)$.

An optimal solution to the 2SAT-6 instance ϕ would be to have x_1 and x_3 be true, and x_2 and x_4 false, or vice versa. This satisfies all but 4 clauses, which is optimal because one clause per variable must go unsatisfied, by construction.

One way to transform this instance into a 2SAT-3 instance ϕ', via the erroneous reduction, is shown in Figure 2.2. The resulting ϕ' has 40 clauses, and one can show using the implication cycles that OPT must be at most 36—there must still be at least one unsatisfied clause corresponding to each variable.

One such optimal solution is shown in Figure 2.3, along with how this solution is mapped back onto ϕ via the majority function. The resulting assignment is the all-False assignment, which is not an optimal solution to ϕ. Both $(x_1 \vee x_2)$ and $(x_3 \vee x_4)$ are also left unsatisfied, bringing the total to 6 unsatisfied clauses.

This violates the requirement for an L-reduction, as due to the right hand side being $O(0)$,

$$\text{OPT}_{\text{2SAT-}B}(\phi) - \text{Score}_{\text{2SAT-}B}(\phi, A) \notin O(\text{OPT}_{\text{2SAT-3}}(\phi') - \text{Score}_{\text{2SAT-3}}(\phi', A')).$$

This in fact fails the PTAS-reduction condition as well, as the assignment A' is exactly optimal, so falls within a $(1+\delta)$-approximation to $\text{OPT}_{\text{2SAT-3}}(\phi')$ for any $\delta > 0$, while the resulting A is only an $(12/10)$-approximation to $\text{OPT}_{\text{2SAT-}B}(\phi)$.

Thus, the reduction does not fulfill the role of showing APX-completeness for Max 2SAT-k for small values of k.

[7] Credit to Zi Song Yeoh for discovery of the original version of this counterexample, reproduced here with only minor changes.

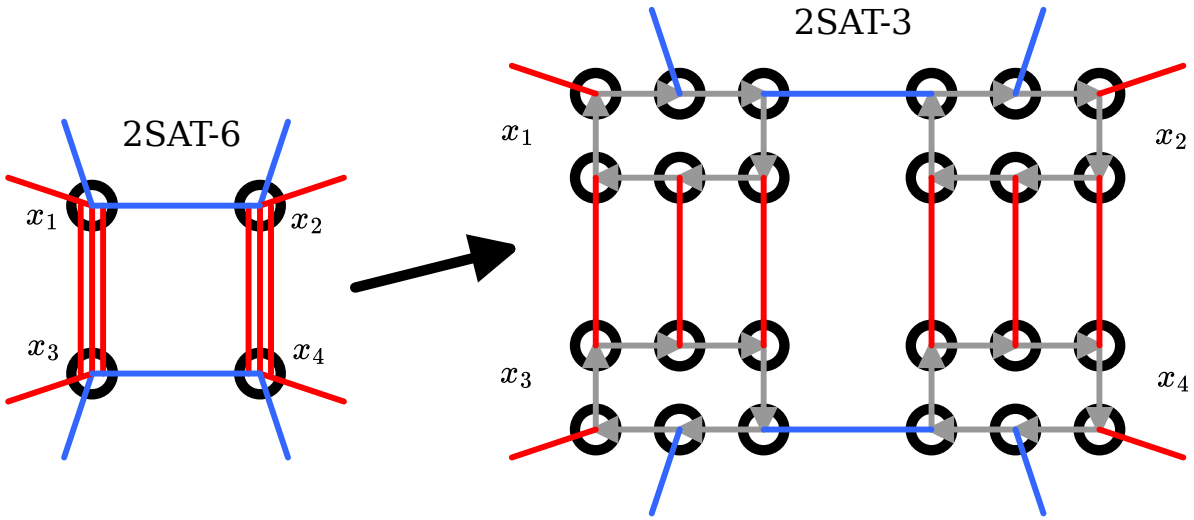

Figure 2.2. A 2SAT-6 instance, and one possible way to transform it into a 2SAT-3 instance. Clauses are drawn as denoted in the clause legend in Figure 2.1. Each cluster of 6 variables in the 2SAT-3 instance is connected with an implication cycle, and corresponds with one x_i in the 2SAT-6 instance.

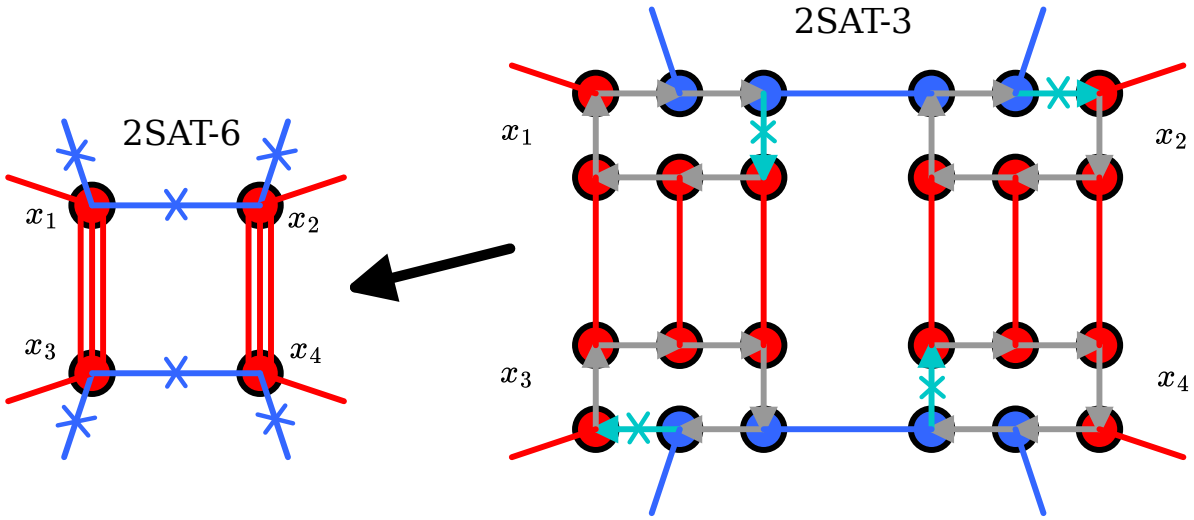

Figure 2.3. A solution to the reduced 2SAT-3 instance which violates the L-reduction condition, and the corresponding 2SAT-6 instance using the majority function. Broken clauses are denoted with X's. Although this is an optimal solution to the 2SAT-3 instance, every implication cycle is broken. The majority of variables in each cycle are False, which results in a suboptimal solution when mapped back to the 2SAT-6 instance.

2.2.3 Confusion and Omissions

After its publication, the majority of citations for APX-completeness of Max 2SAT-3 and Max 3SAT-3 appeared to shift towards Ausiello's 1999 textbook [7]. Although the textbook is a good aid in understanding most of the reduction chain, the fact that it doesn't present the entire proof leaves a hole for those aiming to fully understand the results. Some papers cited Berman and Karpinski [9] as well for its numerical bounds on gap approximation hardness, but inadvertently labeled it APX-hardness, adding to the confusion.[8]

Feige's gap inapproximability result for E3SAT-E5 [6] is shown fairly early in the results, in Proposition 2.1.2. Although this result does not imply that the problem is APX-hard, as we will see, the setup and reduction are very similar to an APX-hardness argument.

The proof of this result is only sketched, and proof that the construction works is left to the reader. The given construction turns out to be very similar to the erroneous L-reduction presented in Section Section 2.2.2, the only difference being that each link in the cycle of x_i's has a two-way implication, so that each is part of 5 clauses.

Though not explicitly stated, we can assume from context that this is a *gap-preserving* reduction. This reduction only differs from an L-reduction in one way: when transforming an assignment to the reduced 3SAT-5 instance back to the original 3SAT-k instance, the proof is concerned with ensuring the number of unsatisfied clauses stays within a constant factor, rather than ensuring the difference from the optimal solution stays within a constant factor.

In this case, the construction correctly works as a gap-preserving reduction. Although an optimal solution to the reduced CNF formula may become suboptimal when mapped to the original problem, the number of *unsatisfied clauses* cannot grow by more than a factor of $k/2$, since each variable cycle of x_i's broken in the reduced instance adds 2 unsatisfied clauses, and can only help satisfy the up to k clauses that x was a member in.[9]

The fact that this central idea is left as an exercise to the reader, combined with the gap reduction's similarity to an L-reduction, and the fact that the sketched reduction takes the APX-hardness of 3SAT-k as a starting point, all adds to confusion when attempting to understand the Max 2SAT-3 reduction chain.

2.3 Max 2SAT-3 APX-completeness

To start, for any n, k, Max nSAT-k is in APX via a simple 2-approximation—as previously mentioned, every clause is satisfied by either the all-True or all-False input, so one of these inputs must satisfy at least half the clauses.[10]

As such, this section will focus on the APX-hardness proof, in particular on the difficult

[8] As far as we could find, [9] does not attempt to prove APX-hardness of these problems.

[9] Note that this gap reduction would work just as well with a single directed cycle, which would show gap hardness for 3SAT-3.

[10] For Max 3SAT, Karloff-Zwick is another polynomial-time algorithm which provides a 7/8-approximation if the formula is satisfiable.

step of reducing from 2SAT-$O(1)$ to 2SAT-3. We will first cover some techniques introduced by earlier papers in Section 2.3.1, then using the core design motivation behind these techniques, present a new L-reduction from 2SAT-$O(1)$ to 2SAT-3.

2.3.1 Previously Used Techniques

The techniques we encountered for reducing the number of variable occurrences mainly revolve around the theory of *expander graphs*. We will begin by discussing the framework of these reductions and the desired properties of a construction. We will then show how expander graphs have these desired properties, and explicitly define a variant of expander graphs which embody these properties more directly.

There are two main applications for these reductions: ones that start from Max 2SAT-$O(1)$, and ones that start from Max 2SAT with unbounded variable occurrences.[11]

In either case, the central idea is to split up a variable x into many copies, to spread out the clauses containing x among them. However, as discussed in Section 2.2.2, we must ensure that it is optimal to assign all copies of x the same truth value—that is, that "cheating" by assigning some copies of x true and others false cannot increase the total number of satisfied clauses. Thus, we want some way of connecting copies of x with clauses such that when the assignment to copies of x is split, enough of these clauses are violated to guarantee this property.

Now, we can define an expander graph, and show how it can be used in such a construction.

Definition 2.3.1. An *expander graph* is a graph $G = (V, E)$ so that any partition $V = V_1 \sqcup V_2$ has at least $\min(|V_1|, |V_2|)$ edges between vertices in V_1 and V_2.

Suppose we have a 2CNF variable x with k occurrences, and suppose an expander graph $G = (V, E)$ on k nodes has max degree k'. Then, replace x with k copies $x_1, \ldots, x_k$, corresponding to vertices in V. For each original clause containing x, arbitrarily replace the occurrence of x with a distinct member $x_i \in V$. Then, for each edge $x_i x_j$ in E, add the clauses $x_i \vee \neg x_j$ and $x_j \vee \neg x_i$, which enforce equality.

We can prove that it's always optimal to set all x_i equal and satisfy all edge conditions. Otherwise, the definition ensures that compared to such a state where all x_i are equal, changing the value of c different x_i will satisfy up to c additional clauses, but invalidate at least c clauses within G.

This argument allows us to reduce from Max 2SAT-k to Max 2SAT-$2k' + 1$ given a single expander graph, or from Max 2SAT if given an infinite family of expander graphs with bounded max degree.

However, our necessary condition is weaker than just expander graphs, in two ways. First, expander graphs are undirected, but end up being split into two "directed" clauses. Second, we can allow $|G|$ to be larger than the number of occurrences of x, so not all nodes must connect to an outside clause.

[11] Most historical applications did work with Max 3SAT, but we will continue defaulting to "Max 2SAT" when a technique is applicable to either one.

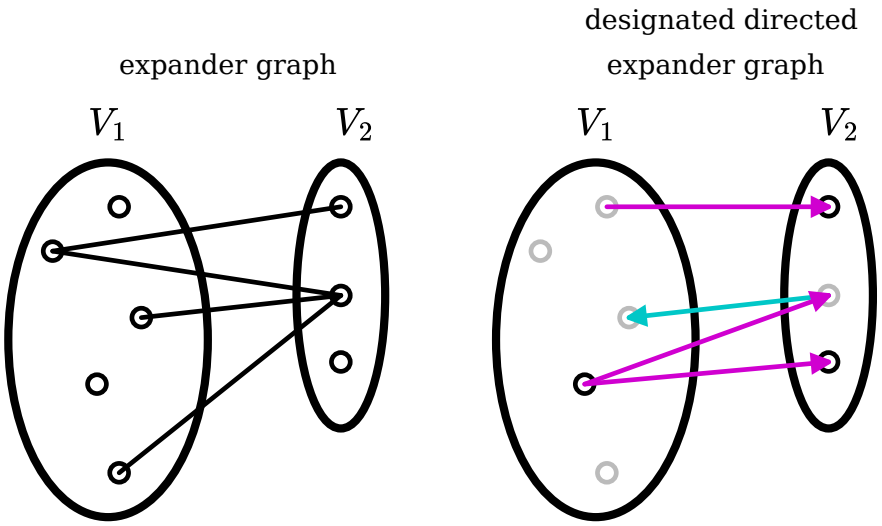

Figure 2.4. Left: an example of partitioning an expander graph. Since $\min(|V_1|, |V_2|) = 3$, there must be at least three edges between V_1 and V_2, the positions of which don't matter. Right: an example of partitioning a designated directed expander graph. Exterior nodes are still shown in black, while interior nodes are shown in gray. Edges are now directed and colored based on direction. Here, $\min(|V_1 \cap V'|, |V_2 \cap V'|) = 1$, so there must be at least one edge pointing from V_1 into V_2, and from V_2 into V_1.

With this in mind, we lastly define the variant on expander graphs which is directly applicable to these reductions. See Figure 2.4 for a comparison between this variant and a traditional expander graph.

Definition 2.3.2. A *designated directed expander graph* is a directed graph $G = (V, E)$ with *exterior nodes* $V' \subseteq V$, so that any partition $V = V_1 \sqcup V_2$ has at least $\min(|V_1 \cap V'|, |V_2 \cap V'|)$ edges pointing from V_1 into V_2, and at least $\min(|V_1 \cap V'|, |V_2 \cap V'|)$ edges pointing from V_2 into V_1.

Then, we can make the following reduction using a similar argument as before.

Lemma 2.3.3. *Suppose we have a designated directed expander graph on B exterior nodes, with max degree B'_{ext} on exterior nodes and overall max degree B'_{int}. Let $B' = \max(B'_{ext} + 1, B'_{int})$. Then, there is an L-reduction from Max 2SAT-B to Max 2SAT-B'.*

Proof. Let $G = (V, E)$ be the designated directed expander graph, and let $V' \subseteq V$ be the set of exterior nodes, where $|V'| = B$.

Suppose we are given a 2CNF-B formula ϕ; we will first construct a new 2CNF formula ϕ'. As before, consider a variable x in ϕ; suppose x appears k times. We first create a copy of G corresponding to x, denoted G_x, in ϕ' as follows. First, create a variable in ϕ' corresponding to each vertex $v \in V$. Then, for each directed edge $(v_1, v_2) \in E$, connect the corresponding variables with the clause $(\neg v_1 \vee v_2)$, logically equivalent to $(v_1 \to v_2)$. Finally, pick k of the

exterior nodes $\{v_1, \ldots, v_k\} \subseteq V'$ and add the k original clauses containing x in ϕ, replacing x with distinct v_i.[12]

In this construction, each variable is used in at most $B' = \max(B'_{ext} + 1, B'_{int})$ clauses in ϕ', since only exterior nodes may connect to a single clause outside the graph.

Then, given an assignment A' of ϕ', we produce the assignment A of ϕ using the majority function on only the exterior nodes. We will show that any assignment of ϕ' can be converted into one with at least as many satisfied clauses, such that for each variable x, its exterior nodes have the same value. Consider an assignment A' of ϕ', and fix a variable x occurring k times in ϕ; we will focus on clauses involving G_x.

There are k exterior nodes $\in V'_x$ connected to external clauses. Suppose i of them are assigned True, and $k-i$ are assigned False. Then, let V_T, V_F be a partition of G_x on whether each node was assigned True or False, respectively. By the designated directed expander graph condition, at least $\min(|V_T \cap V'_x|, |V_F \cap V'_x|) = \min(i, k-i)$ clauses are implications pointing from V_T to V_F. Thus, there are at least $\min(i, k-i)$ unsatisfied clauses internal to G_x.

This safely defends against the issues raised in Section 2.2.2, since any assignment A' that doesn't have consensus across G_x is locally inferior to one that does: by changing all nodes in G_x to the majority vote, the number of satisfied interior edges increases by at least $\min(i, k-i)$, while the number of satisfied exterior edges decreases by at most $\min(i, k-i)$. The resulting assignment still maps to the same A, as well.

We can now formally demonstrate that this is an L-reduction. If ϕ has M variables and N clauses, where $M \leq 2N$ due to ϕ being a 2SAT instance, then ϕ' has $N' = |V|M + N \leq (2|V| + 1)N$ clauses, and

$$\text{Score}_{\text{2SAT-}B'}(\phi', A') \leq \text{Score}_{\text{2SAT-}B}(\phi, A) + N' - N. \tag{$*$}$$

Note that this inequality is proved as a result of the above argument.

Now,

$$\begin{aligned}
\text{OPT}_{\text{2SAT-}B'}(\phi') &\leq \text{OPT}_{\text{2SAT-}B}(\phi) + N' - N \\
&\leq \text{OPT}_{\text{2SAT-}B}(\phi) + 2|V|N \\
&\leq (4|V| + 1) \cdot \text{OPT}_{\text{2SAT-}B}(\phi) \\
&= O(\text{OPT}_{\text{2SAT-}B}(\phi)),
\end{aligned}$$

and for an assignment A' of ϕ' which maps to an assignment A of ϕ,

$$\begin{aligned}
&\text{OPT}_{\text{2SAT-}B}(\phi) - \text{Score}_{\text{2SAT-}B}(\phi, A) \\
&\leq (\text{OPT}_{\text{2SAT-}B'}(\phi') - N' + N) - (\text{Score}_{\text{2SAT-3}B'}(\phi', A') - N' + N) \\
&= O(\text{OPT}_{\text{2SAT-}B'}(\phi') - \text{Score}_{\text{2SAT-}B'}(\phi', A')),
\end{aligned}$$

noting that the inequality $(*)$ for Score is actually tight for OPT, via the canonical A' where

[12] All clauses of ϕ will appear exactly once in ϕ', with each variable x of the clause replaced by some exterior node in G_x.

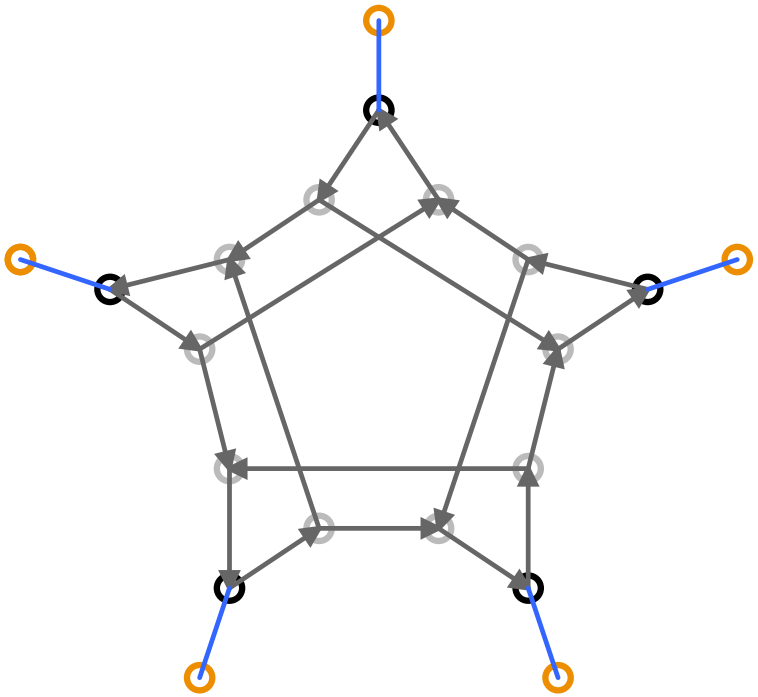

Figure 2.5. Redrawing of the graph depicted by Figure 8.7 in [7].[a] This gadget is an instance of a designated directed expander graph (proof left to the reader), and was used for the Max 3SAT-5 to Max 3SAT-3 reduction. Blue segments represent connections of this gadget to other clauses, via the external nodes.

[a] The graph was edited to fix a presumed drawing mistake.

all nodes in G_x take on the truth value of x in A. This shows both necessary big O relations, so we have given a valid L-reduction. □

Note that the explicit graph used for the Max 3SAT-5 → Max 3SAT-3 reduction in [7], shown in Figure 2.5 is an example of such a designated directed expander graph, using 15 total nodes to achieve such a graph with $k = 5, k'_1 = 2, k'_2 = 3$.

2.3.2 Simple L-Reduction from Max 2SAT-$O(1)$ to Max 2SAT-3

In this section, we provide a simple L-reduction from Max 2SAT-$O(1)$ to Max 2SAT-3.

As mentioned in Section 2.2.1, the reductions used in [1] are enough to show APX-hardness of Max 2SAT-10. Thus, when combined with this foundational work, this reduction will complete a full proof that Max 2SAT-3 is APX-hard.

The central idea of the following reduction is to define a designated directed expander graph on k exterior nodes $x_1, \ldots, x_k$ by instantiating a large directed cycle for each x_i, then simulating implications of the form $x_i \to x_j$ by connecting a node in x_i's cycle to one in x_j's

cycle.[13]

Theorem 2.3.4 (Restatement of Theorem 2.2.1)**.** *For all $B > 3$, there exists an L-reduction from Max 2SAT-B to Max 2SAT-3.*

Proof. Using Lemma 2.3.3, it suffices to provide a designated directed expander graph on B exterior nodes, satisfying $B'_{ext} = 2$ and $B'_{int} = 3$. Figure 2.6 provides an example of this graph for the $B = 4$ case.

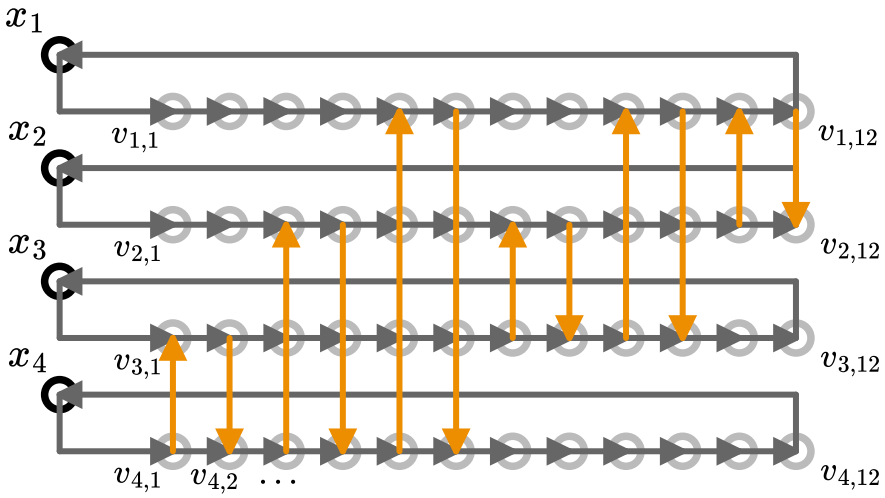

Figure 2.6. A designated directed expander graph for some variable x occurring 4 times in the Max 2SAT-$O(1)$ instance. The purple nodes on the left are the exterior nodes, are connected to one original clause each.

Define the graph $G = (V, E)$ containing the following:

- B exterior nodes, $V' = \{x_1, \ldots, x_B\}$.
- For each exterior node x_i, $B(B-1)$ interior nodes $v_{i1}, \ldots, v_{i,B(B-1)}$.
- For each exterior node x_i, a directed cycle connecting it with all its interior nodes: $x_i \to v_{i1} \to v_{i2} \to \cdots \to v_{i,B(B-1)} \to x_i$.
- Enumerate all ordered pairs of exterior nodes. For each pair x_i, x_j which is n^{th} in the enumeration, add the edge $v_{in} \to v_{jn}$.

This graph has B directed cycles of $B(B-1)+1$ nodes each;[14] however, the size of this graph is irrelevant to showing the L-reduction, as B is a constant.

[13] Credit to Lily Chung and Zi Song Yeoh for the discovery and initial writeup of this reduction.

[14] Interior nodes per exterior node can be reduced from $B(B-1)$ to $B(B-1)/\lfloor B/2 \rfloor \approx 2B$, by partitioning K_B into perfect or maximal matchings to make more efficient use of space. The size can be reduced even further by using expander graph techniques to reduce the interconnection between cycles. However, this demonstrates that our technique can turn even a complete graph into a cubic designated directed expander graph.

Each external node x_i is only connected to v_{i1} and $v_{i,B(B-1)}$, and has degree 2. Each internal node v_{in} is connected to some v_{jn}, as well as two adjacent nodes in the cycle, so has degree 3. Thus, the degrees satisfy $B'_{ext} = 2$ and $B'_{int} = 3$.

Now, we will show that G satisfies the designated directed expander graph property. Given an arbitrary partition $V = V_1 \sqcup V_2$, call an index i *consistent* if $\{x_i, v_{i1}, \ldots, v_{i,B(B-1)}\}$ is a subset of either V_1 or V_2, and *inconsistent* otherwise. Suppose without loss of generality that $|V_1 \cap V'| < |V_2 \cap V'|$, that is, fewer x_i are in V_1 than in V_2. We want to show that there are $|V_1 \cap V'|$ edges between V_1 and V_2 in both directions.

Every inconsistent index contributes at least 1 edge in each direction, since the directed cycle passes through both V_1 and V_2. Every pair of consistent indices i, j where x_i and x_j are in opposite partitions also contributes at least 1 edge in each direction, due to the $v_{in} \to v_{jn}$ and $v_{jn} \to v_{in}$ edges.

Suppose there is some consistent j such that $x_j \in V_2$. Then, all indices i in $|V_1 \cap V'|$ are either inconsistent and contribute 1 directly, or are consistent and the pair i, j contributes 1. If there is no such j, then $|V_2 \cap V'| > |V_1 \cap V'|$ indices are inconsistent. In either case, the designated directed expander graph property has been shown. □

References

[1] C. H. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," *Journal of Computer and System Sciences*, vol. 43, no. 3, pp. 425–440, 1991, ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(91)90023-X. URL: https://www.sciencedirect.com/science/article/pii/002200009190023X.

[2] P. Crescenzi and A. Panconesi, "Completeness in approximation classes," *Information and Computation*, vol. 93, no. 2, pp. 241–262, Aug. 1991, ISSN: 0890-5401. DOI: 10.1016/0890-5401(91)90025-W. URL: https://www.sciencedirect.com/science/article/pii/089054019190025W.

[3] P. Crescenzi and L. Trevisan, "On Approximation Scheme Preserving Reducability and Its Applications," in *Proceedings of the 14th Conference on Foundations of Software Technology and Theoretical Computer Science*, Berlin, Heidelberg: Springer-Verlag, Dec. 1994, pp. 330–341, ISBN: 9783540587156.

[4] C. H. Papadimitriou, *Computational Complexity*. Reading, Mass.: Addison-Wesley, 1994, ISBN: 9780201530827. URL: http://archive.org/details/computationalcom0000papa.

[5] L. Trevisan, "When Hamming meets Euclid: the approximability of geometric TSP and MST (extended abstract)," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing - STOC '97*, El Paso, Texas, United States: ACM Press, 1997, pp. 21–29, ISBN: 9780897918886. DOI: 10.1145/258533.258541. URL: http://portal.acm.org/citation.cfm?doid=258533.258541.

[6] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM*, vol. 45, no. 4, pp. 634–652, Jul. 1998, ISSN: 0004-5411. DOI: 10.1145/285055.285059. URL: https://dl.acm.org/doi/10.1145/285055.285059.

[7] G. Ausiello, A. Marchetti-Spaccamela, P. Crescenzi, G. Gambosi, M. Protasi, and V. Kann, *Complexity and Approximation*. Springer Berlin Heidelberg, 1999, ISBN: 9783642584121. DOI: 10.1007/978-3-642-58412-1. URL: http://dx.doi.org/10.1007/978-3-642-58412-1.

[8] Wikipedia contributors, *MAX-3SAT — Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=MAX-3SAT&oldid=1169265069, [Online; accessed 20-Nov-2023], 2023.

[9] P. Berman and M. Karpinski, "On Some Tighter Inapproximability Results (Extended Abstract)," in *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1999, pp. 200–209, ISBN: 9783540485230. DOI: 10.1007/3-540-48523-6_17. URL: http://dx.doi.org/10.1007/3-540-48523-6_17.

Chapter 3

The *Net* Tile Rotation Puzzle Family

Net is one of the puzzles in *Simon Tatham's Portable Puzzle Collection*, dating back to 2004 [1]. In *Net*, you are given a grid of rotatable tiles, each with a section of pipe that has 1, 2, 3, or 4 loose ends connecting to some adjacent tiles. An example grid is shown in Figure 3.1. When two adjacent tiles have loose ends that match up, the two loose ends become a connection between the tiles, instead. The goal of the game is to connect all the tiles, with no loose ends remaining.

This basic puzzle format dates back to even before *Net*, and variants of this concept have since become ubiquitous as puzzle minigames and standalone games. Versions of this game have gone by many names, including *FreeNet*, *Pipes*, *KPlumber*, *Infinity Loop*, and the *Patching* minigame from *Puzzle Pirates*, to name a few examples.

In this chapter, we will mainly consider *Net* as it appears in *Simon Tatham's Portable Puzzle Collection*, as well as *KPlumber* and the *Puzzle Pirates* variant. We first summarize our results in Section 3.1, then introduce these main variants in Section 3.2. Then, in Section 3.3, we'll give an alternate proof of a known result, that *Net* is NP-complete. Finally, in Section 3.4, we'll show that *Puzzle Pirates*' *Patching* minigame is itself NP-complete as a decision problem, and a closely-inspired optimization variant of *Net* is $\Theta(n)$-gap NP-hard as well, meaning it has no constant-factor approximation algorithm if P $\neq$ NP.

This chapter represents joint work with Andy Tockman, Della Hendrickson, Erik Demaine, and others in the MIT class Algorithmic Lower Bounds: Fun with Hardness Proofs (6.5440), taught by Erik Demaine in Fall 2023.

3.1 Results Summary

In this section, we summarize existing a new results in complexity of *Net* variants, prove a minor result from the table, and discuss an open problem of note.

Section 3.1 shows current progress on characterizing the complexity of *Net* variants with limited piece types. The table covers *Net* (as "connected tree"), *KPlumber* (with neither modifier), and a third variant ("connected") which to our knowledge has not been studied before. For this section, we will refer to the "connected" variant as *Cycle-Net*. The table

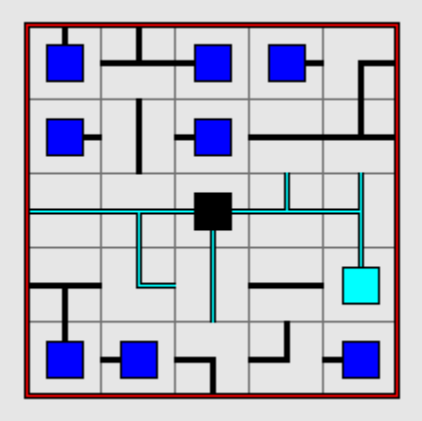

Figure 3.1. An example game of *Net*, from Simon Tatham's Portable Puzzle Collection. The source piece is marked with a black square, and the network is highlighted in blue.

does not cover the optimization variant introduced in Section 3.4, as the format would be too different.[1]

The *KPlumber* results in the table are from [2]. The polynomial-time algorithms given in Theorems 3 and 4 of [2] are applicable to *Cycle-Net* as well, because these *KPlumber* cases have a single valid solution, if any.

NP-completeness of *Net* with or without blanks and Q pieces will be discussed in Section 3.3. Since "tree" is an input restriction, NP-completeness results carry over from *Net* to *Cycle-Net*. Polynomial-time algorithms carry over from *Cycle-Net* to *Net*. In addition, there are no instances of *Net* when Q pieces are disallowed. This is because the condition requires that the average degree of pieces be strictly less than 2.

Now, we include here a brief result:

Theorem 3.1.1. *Cycle-Net is solvable in polynomial time with O, Q, I, X pieces.*

Proof. Note that if a Q, I, or X piece is known to have a neighbor with a half-edge pointing into it, then its orientation is determined. Thus, we can solve a *Net* instance via breadth-first search of all the pieces, starting at an X piece, if there is one. For each new piece adjacent to the existing network, we determine its orientation, so the problem will be solved when the traversal finishes, or the algorithm fails due to forcing a loose end.

In the case where there is no X piece, then the instance is only solvable if it consists of consecutive I pieces on a single row or column, bookended by two Q pieces. □

Finally, we would like to remark on the last remaining *KPlumber* case, that of *KPlumber* with only Q and I pieces.[2] The authors of [2] found this case especially pernicious, as

[1] We considered making "connected" a trichotomy: All tiles must be connected, tiles may be disconnected, or maximize the connected component of tiles. This trichotomy would seem to imply that the optimization variant also requires no loose ends anywhere, however.

[2] All other pieces besides L are optional, as [2] showed that all remaining cases are equivalent.

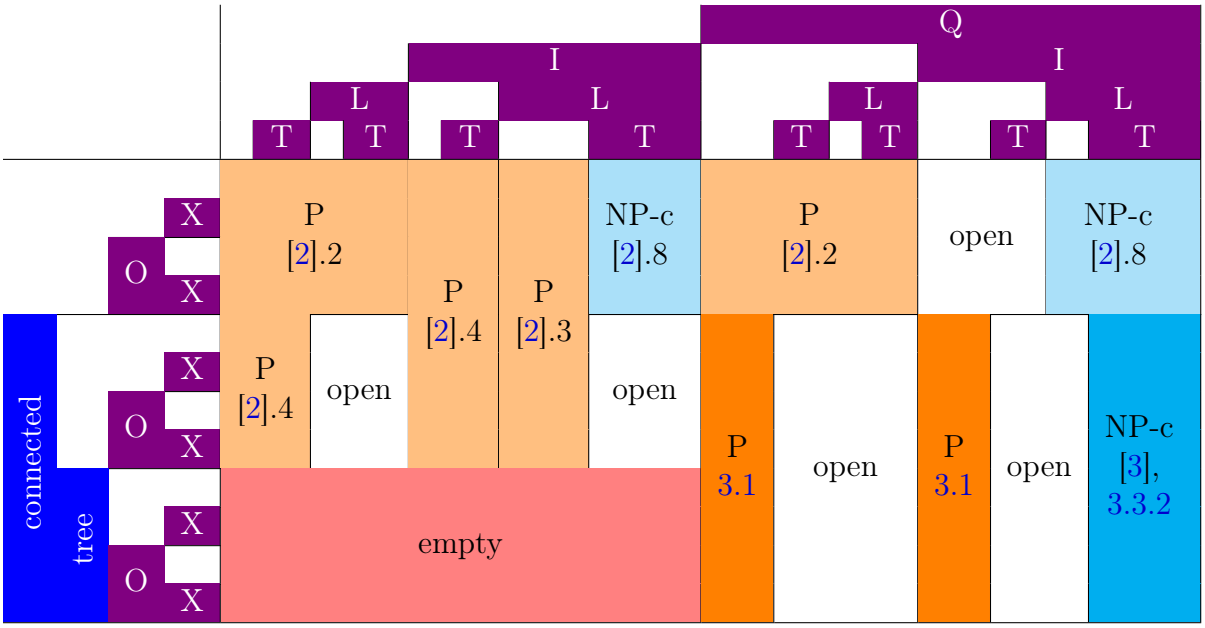

Figure 3.2. Table showing progress on characterizing *Net* variants with limited piece types.[a] Piece type headers (shown in purple) specify whether the piece type is allowed; "connected" indicates that solutions must form a single connected network, and "tree" indicates that additionally, the input is constrained to inputs that satisfy the tree constraint defined in Section 3.2.2. The "connected tree" variant is equivalent to *Net*, and without either modifier is equivalent to *KPlumber*.

[a] Credit to Della Hendrickson for the design, as well as generating the LaTeX source for this table.

did we. Although we will not include our findings on this problem, we recommend it as a simply-stated but tough problem to work on.

3.2 Rules, Variants, and Prior Work

In this section, we will first introduce the terminology we will be using throughout the chapter. Then, we'll introduce the rules for *Net* and notable variant rulesets, and survey prior work relating to each of them.

3.2.1 Terminology and Piece Types

Most *Net* variants use their own terminology, based on the theming of the game. For instance, the goal of *Net* is to connect tiles into a network, the goal of *Patching* is to connect tears in a sail to patch them, and many variants revolve around connecting pipes to a water source, to fill them all with water. Thus, we will introduce here some common terminology

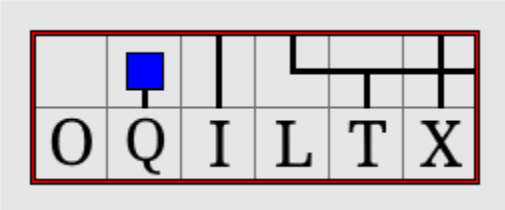

Figure 3.3. Net piece types, as displayed in Simon Tatham's Portable Puzzle Collection. Styling may vary between implementations of the game, but Q pieces are often marked with a symbol in the center, to distinguish them from loose ends that must be connected.

we will use to describe the pieces and rules for all variants.

- The *tiles*, or *pieces* are the individual squares of the grid, which may be rotated. A tile may have up to four *half-edges* or *segments*, represented by line segments emanating from the center towards a side of the square. The number of segments a tile or piece has is its *degree*. Two adjacent tiles are *connected* if each has a segment pointing towards the other. Segments which are not part of connections are called *loose ends*.
- A *piece type* describes the shape of a single tile, up to rotation. There are six different piece types, which we will refer to by letters, which serve as a visual aid mimicking the shape that a tile's segments form. There is only one piece type with each of degrees 0, 1, 3, and 4; these are called O pieces (or blanks), Q pieces, T pieces, and X pieces, respectively. The two piece types with degree 2 are the I pieces, with two segments opposite each other, and L pieces, with segments at a right angle. The six distinct piece types and corresponding letters are shown in Figure 3.3.
- In some variants, the *source* is just a special non-blank piece, and the *network* is the connected component of the source, that is, all tiles with a path of connections to the source. The source and network in *Net* can be seen in Figure 3.1. Pieces that must be connected to the source, usually Q pieces, are called *required sinks*.

3.2.2 *Net*

In *Net*, the player starts with a grid of tiles, none of which are blank. There is a source tile, and the goal is to rotate each tile until all tiles are part of the network. In this variant, the input must satisfy that

$$(\text{total } \# \text{ of half-edges on the board}) = 2(\# \text{ of tiles on the board} - 1),$$

which guarantees that the resulting network forms a tree, and will not have any loose ends or cycles.[3]

[3]This directly follows from the fact that in a connected graph $G = (V, E)$, $|E| = |V| - 1$ is equivalent to G being a tree.

There is also an option to enable *toroidal wrapping* on the grid, meaning that a tile on the bottom row of the grid is adjacent to the corresponding tile on the top row, and similarly for columns.

With these rules, *Net* was shown to be NP-complete by Marzio De Biasi in 2012 [3].[4] In Section 3.3, we will describe this proof in more detail, as well as present an alternate proof of this result.

3.2.3 *Puzzle Pirates*

Puzzle Pirates [4] is an MMO, or massively multiplayer online game, released in 2003. The game revolves around life as a pirate aboard a pirate ship, which can enter combat with other pirate ships. As the name suggests, much of the gameplay is based on puzzles, which are generally simple but may involve time pressure and teamwork to complete.

Puzzle Pirates is one of many games to feature *Net*-like gameplay as a minigame. It features the *Patching* [5] puzzle, in which players attempt to patch a torn sail by rotating squares of cloth so that the tears line up; otherwise risking tearing the entire sail.[5]

The *Patching* variant features a source, represented by a spool of thread, and arbitrarily many required sinks, called tie-offs. The goal is to make the source's network as large as possible, under the conditions that required sinks must be included, and the network must have no loose ends. Pieces outside the network, however, are allowed to have loose ends.

The variant also features *blockers*, which cannot be moved and don't connect to anything, essentially rendering it an immobile blank.

The final rule unique to this variant is that pieces can be dragged around and swapped with each other. We will analyze the variant both with and without this rule in Section 3.4, so the existence of immobile blockers will be useful for the version allowing pieces to move.

3.3 *Net* is NP-Complete: Another Proof

The proof that *Net* is NP-Complete in [3] is based on a reduction from Hamiltonian cycle on grid graphs of max degree 3. In Section 3.3.1, we will first give a brief overview of the reduction that De Biasi used. Then, in Section 3.3.2, we will present an alternate proof of the same result, using a reduction from the Tree-Residue Vertex-Breaking problem, or TRVB.

3.3.1 Existing Reduction from Grid Graph Hamiltonicity

Hamiltonian cycle is one of Karp's 21 NP-complete problems, as one of the foundational NP-completeness results. Many puzzles are shown to be NP-hard by reduction from Hamiltonian cycle; however, for many puzzles, it is difficult to construct representations of arbitrary

[4]Both with and without toroidal wrapping; however, the wrapping is not very significant in the context of this chapter.

[5]We recognize that being able to freely rotate squares to line up tears would not be possible in reality.

graphs. As such, researchers have proved Hamiltonian cycle NP-hard on many restrictions of graphs, including the set of *grid graphs* with max degree 3.

Grid graphs can be formulated as follows: pick a set of vertices (x, y) on the coordinate grid, and connect two vertices if and only if they are adjacent—that is, if they are distance 1 apart. Thus, the result on Hamiltonian cycle on max degree 3 grid graphs states that even when restricted to these grid graphs which have no degree-4 vertices, it is still NP-hard to determine whether such a graph contains a Hamiltonian cycle. We will use the problem of Hamiltonian cycle on grid graphs again in Section 3.4.2.

The application of this problem to *Net* relies on the fact that *Net* requires its solution to be connected into a single network, and it's not enough for just all loose ends to be filled. Thus, [3] designs gadgets to simulate vertices of such a grid graph, such that all the pieces are connected if and only if the grid graph has a Hamiltonian cycle.[6] Consult [3] for full details of the reduction.

3.3.2 Reduction from Tree-Residue Vertex-Breaking

Tree-Residue Vertex-Breaking [6], or TRVB, is a problem intended for use in NP-hardness reductions, introduced by Erik Demaine and Mikhail Rudoy in 2018. The problem takes as input an arbitrary multigraph G, with each node labeled either *breakable* or *unbreakable*. To break a breakable node v with degree d, replace v with new degree-1 vertices $v_1, \ldots, v_d$, each connected to one of v's old neighbors.

The authors study constraints on the input multigraph, specifically restrictions to planar graphs, and restrictions on the possible degrees of breakable and unbreakable vertices. Thus, for $B, U \subseteq \mathbb{N}$, the (B, U)-TRVB problem restricts breakable vertices to have degrees in B, and unbreakable vertices to have degrees in U. The Planar Graph (B, U)-TRVB problem additionally restricts the multigraph to be simple and planar.

A template for a gadget using Planar Graph $(\{4\}, \{\})$-TRVB is shown in Figure 3.4. When extended to an entire graph G, if gadgets of this form are connected together by wires, the thick black lines will end up forming a single cycle if and only if the shaded interior forms a tree.[7]

This construction forms the central idea behind our reduction, which we now present.

Theorem 3.3.1. *Net, as defined in Section 3.2.2, is NP-hard.*

Proof. We reduce from Planar Graph $(\{4\}, \{\})$-TRVB. Given a 4-regular planar graph G, we first find a planar embedding of G satisfying the following:

- All vertices lie on lattice points, with its four edges pointing in the four cardinal directions, and
- All edges are drawn with a combination of axis-aligned segments.

[6] The input constraint then guarantees that there are no cycles or loose ends, but this is also true by direct construction.

[7] This fact is proven in Section 1 of [6], and some details are described in the proof of Theorem 3.3.1.

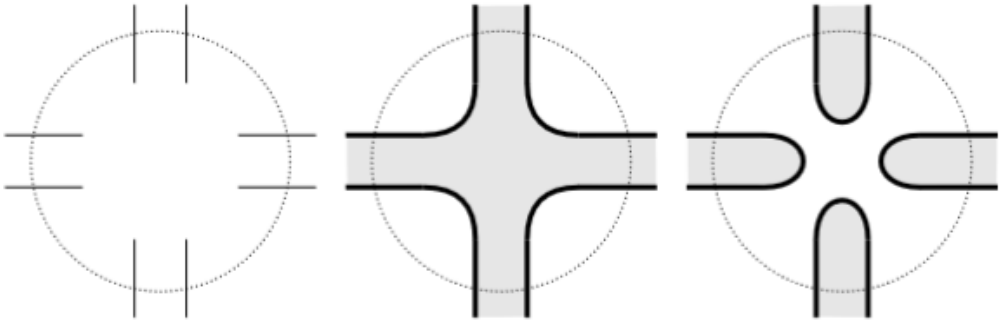

Figure 3.4. Figure 3 from [6]. This diagram illustrates how the authors intended a possible vertex-breaking gadget to work. On the left is the skeleton of the gadget, which can be filled to represent an unbroken or broken vertex, shown respectively in the center and on the right. The shaded regions represent the interior of the gadget.

In *Net* terminology, this is equivalent to topologically constructing G with only X, I, L, and O tiles.[8]

Now, we take each tile of this construction, and replace it with a corresponding 14×14 metacell, of those shown in Figure 3.5. For X tiles, default arbitrarily to the unbroken vertex gadget metacell.

The vertex gadget functions exactly as the theoretical vertex gadget posed by [6]. In the gadget tile itself, each solution has four connected components, corresponding to the thick black lines in Figure 3.4.[9] The edge gadgets have two connected components representing the boundaries of the edge, to connect the interior of the tree. Each metacell can be proved to only have the solutions shown, by starting from the outside and working in. A formal proof of this is omitted, but the vertex gadget case will be discussed in Section 3.3.3.

Suppose the TRVB instance $G = (V, E)$ has the solution of breaking vertices $V' \subseteq V$, producing a tree G'. Then, for each vertex gadget in the reduced *Net* instance, rotate tiles to either the unbroken or broken position, corresponding to the vertex's membership in V'. The metacells used to construct G will now be connected in a single cycle, from the connected component analysis above. As described in Section 1 of [6], this is because the shaded interior of the gadgets is topologically equivalent to the tree G', so is simply connected.

It is also important to note that if the vertex gadgets are arranged in a way that doesn't correspond to a valid TRVB solution, then the shaded interior will not be simply connected. If G' has a cycle, then the cycle corresponds to a hole in the shaded interior, the inside loop of which is disconnected from the rest of the boundary. If G' is disconnected, then the shaded interior and its boundary are also disconnected.

At this point, one might notice two remaining issues, the remedies to which are shown in Figure 3.6. First, only the metacells used in the construction of G are considered in the

[8] One can also imagine this as a Microsoft PowerPoint diagram with rectilinear arrows.

[9] In square dancing terms, an unbroken vertex connects each dancer with their corner, and a broken vertex connects each dancer with their partner.

above analysis, and O tile metacells are ignored. This can be fixed simply using a scheme for connecting a blank metacell with an adjacent metacell—their border is composed mostly of Q pieces, so select one pair of adjacent Q pieces in the middle, and replace them with I pieces. One can check that this does not interfere with the metacells' ability to solve uniquely. Then, after placing the metacells for G, we can fill space by placing blank metacells one by one, connecting them with an adjacent existing metacell.

The second issue is that as mentioned above, the tiles are connected in a single cycle, which violates the tree input constraint described in Section 3.2.2. This can also be fixed simply—since we are only over by one edge, we define a modified I tile metacell to include the source tile, and break the cycle in one location. □

3.3.3 Reduction Design Discussion

Now, we present the design motivation behind the vertex gadget, which also serves as a brief intuition behind why this gadget has only two solutions.

One early attempt at such a vertex gadget is shown in Figure 3.7. This gadget had an issue where besides the two solutions corresponding to the unbroken and broken TRVB vertex, there was a third solution that connected the entire gadget into one connected component. This is fatal to the construction, since the reduction relied on the fact that lack of a TRVB solution implies that the network must be disconnected, and the gadget's solutions' careful partitions into four connected components is what helps achieve this.

Rather than just accepting this and trying to construct something different, though, we analyzed why this solution failed, which proved to be very insightful. Restricting to the tiles of the gadget which are not fixed (colored in Figure 3.7), and even further to just the specific segments of tiles which were not fixed, we saw that the two solutions actually differed in a very predictable way—where one had an edge the other would not, and they would swap in an alternating fashion. This makes sense when considering that most tiles had only one free segment, or half-edge, and these half-edges needed to be paired together to form edges.

Taking the union of these conditionally-existing edges led to two rectangular loops (shown in pink in Figure 3.7, with colored dots indicating membership to solutions). Tiles with one free segment form the majority of the loops, and simply enforce that in any possible solution, exactly one neighboring edge is active. L tiles with both free segments form the loop crossings. These tiles are entirely free to move, but can be thought of as two separate conditions—if you take an X tile and remove one of each pair of opposite segments, you will always be left with an L tile in some orientation. Thus, the same logical conditions that force alternating carry through crossings, traveling straight through.

Now, the problem with our gadget attempt is clear—we were able to construct a mixed solution because there were two loops, meaning four solutions in total.

To fix this issue, we set out to design a gadget with just one loop, but still had the TRVB connectivity properties. We started with a single loop that would travel twice around the center, and then filled in the rest to ensure that the two possible solutions had the correct

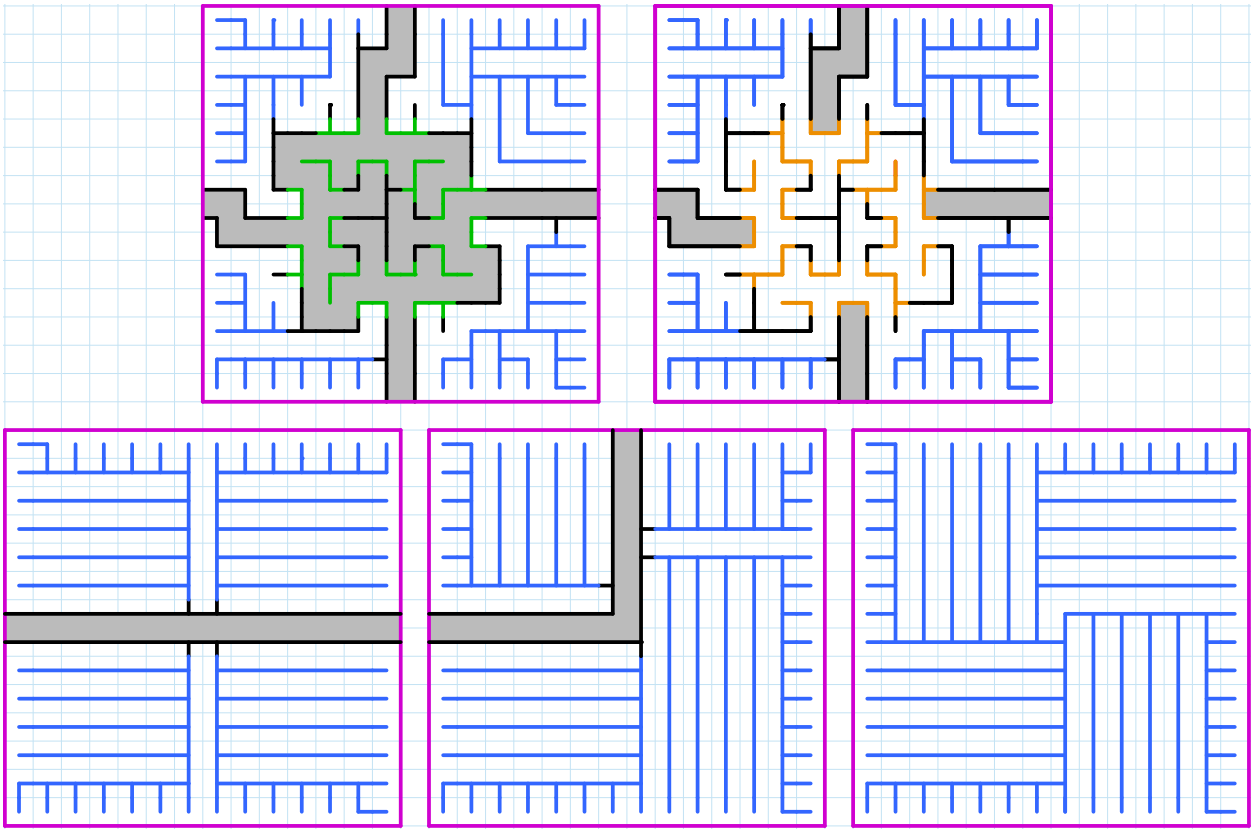

Figure 3.5. Metacells for the *Net* NP-hardness reduction from Planar Graph $(\{4\}, \{\})$-TRVB (Tree-Residue Vertex-Breaking) problem.[a] Top (left-to-right): The solutions to the X tile metacell (vertex gadget), representing an unbroken and broken vertex. Bottom (left-to-right): The I, L, and O tile metacells (edge gadgets and space filling gadget). Metacells are joined together by composing the pink squares together. Tiles part of the embedding of G which have fixed orientation are shown in black. Tiles unique to the vertex gadget solution are shown in green or red. Tiles used to fill space are shown in blue. In all gadgets, the space corresponding to the interior of G is shaded.

[a] Credit to Andy Tockman for designing these space-filling metacells based on the gadget design in Figure 3.8.

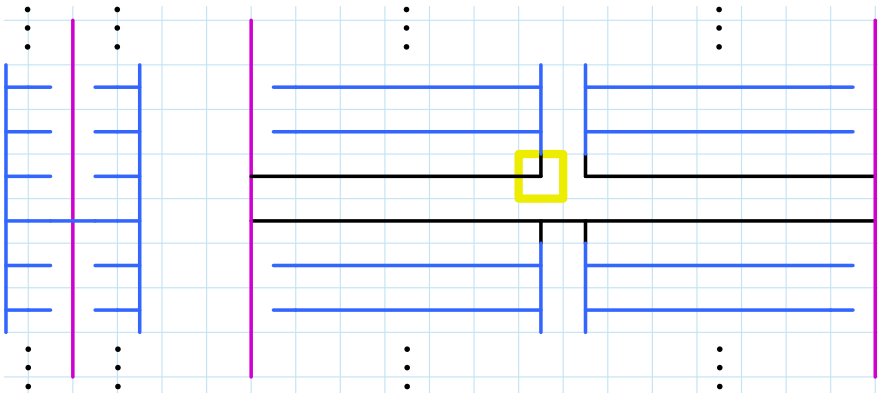

Figure 3.6. Modifications necessary to create a valid *Net* input. Left: Connecting two adjacent metacells, which is done when necessary to ensure that the grid forms a tree when connected. Right: Modification of an I tile metacell to include a source tile, boxed in yellow. This metacell also serves to break the cycle exactly once.

connected components, and were even able to eliminate cycles from the gadget, making it valid for the *Net* ruleset.

The designed gadget is shown in Figure 3.8, and is the basis of the reduction presented in Section 3.3.2.

3.4 *Puzzle Pirates* Optimization Variant

In this section, we analyze the *Puzzle Pirates* optimization variant of *Net*, called *Patching*, background for which is discussed in Section 3.2.3. We choose to analyze this variant in particular because it is a fairly natural extension of *Net* to a non-trivial optimization problem, particularly the version without tile movement.

The variant's pieces are described on the *Puzzle Pirates* wiki, *YPPedia* [5], however, there is no full rules explanation. Here, we briefly explain the rules, to the extent necessary to analyze the complexity. The following must hold for the variant without tile movement:

- As in *Net*, there is a single source (termed the "spool" in this variant) which is a Q piece.
- The tree input restriction does not apply, and cycles are allowed.
- The player is not required to connect all the pieces, but the goal is to maximize the size of the source's connected component, or the network.
- A tile is not allowed to have loose ends if it is part of the network; however, other tiles may have loose ends.

For the variant with tile movement, we require the following to hold in addition:

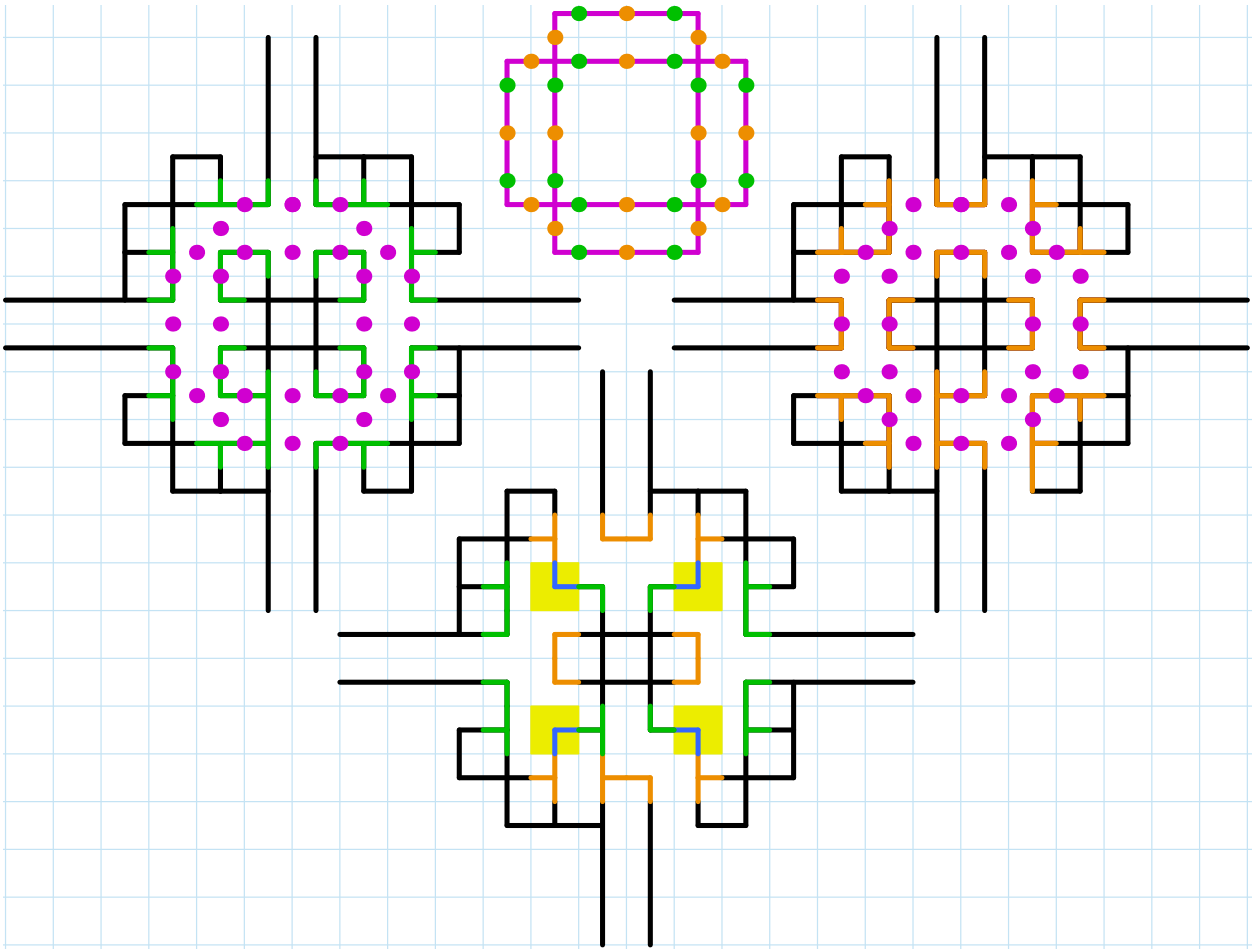

Figure 3.7. Example of a previous attempt at a TRVB vertex gadget (allowing cycles), along with a mixed solution which combines the unbroken and broken solutions into a new solution with only one connected component. At locations where one solution has an edge where the other does not, large pink dots are placed on the midpoint of the edge, or corresponding location. The top diagram shows the two logical implication loops (where loops must go straight at crossings) formed by taking the union of these edges. Colored dots on the top diagram indicate which solution an edge belongs to (where edges begin and end at centers of tiles); a valid solution must have an edge at every other point in the loop. The mixed solution picks the left (green) solution for the wide loop, and the right (orange) solution for the tall loop. Tiles corresponding to loop crossings are highlighted in the mixed solution; the orientations of these tiles are not present in the original solutions, but solution mixing nevertheless causes these to be valid L tiles.

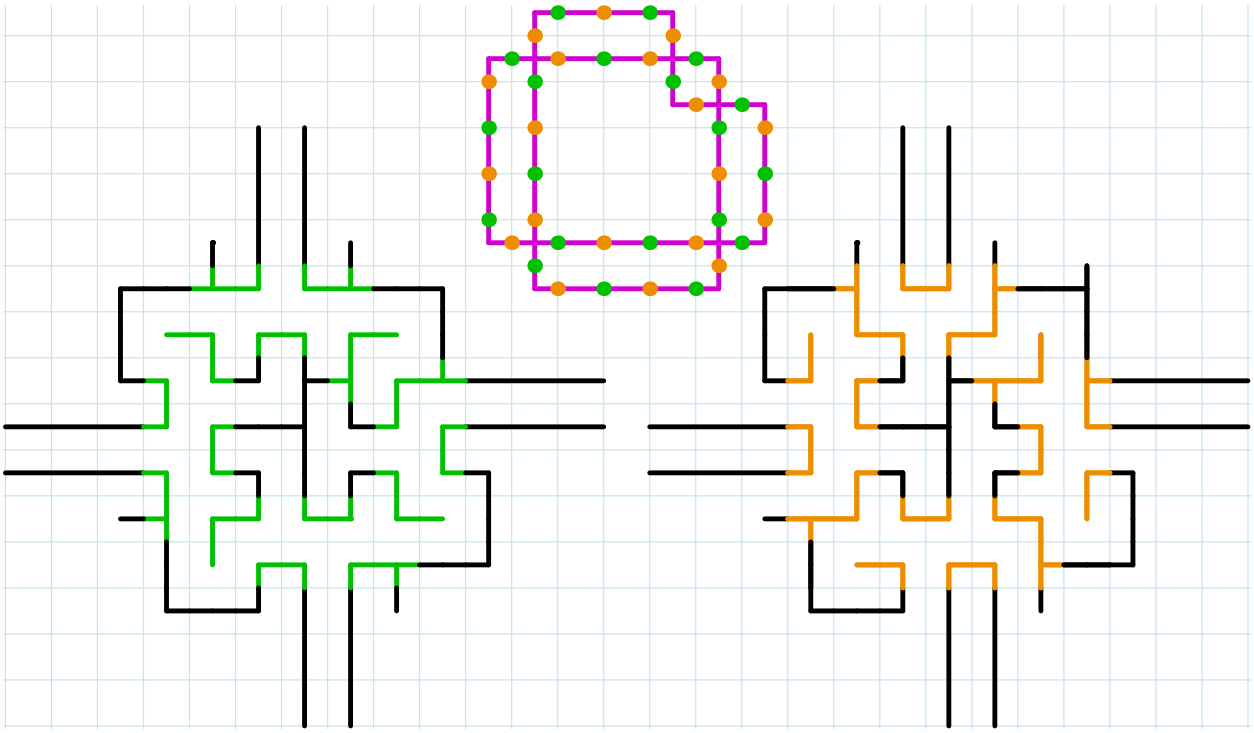

Figure 3.8. The TRVB vertex gadget that our solution is based on. The top diagram is shown in the same format as Figure 3.7, but modified to only have one implication loop (where loops must go straight at crossings). The gadget was designed from this diagram, and its two solutions are shown below. Locations where one solution has an edge where the other does not are omitted on the solutions for clarity; however, these locations correspond in position to what is shown in the top diagram.

- There may be any number of required sinks (termed "tie-offs") which are Q pieces.
- There may be any number of *blockers*, which are special blanks which cannot be moved by the following rule.
- The player may rearrange the positions of any tiles besides the source, required sinks, and blockers, within the bounds of the grid.

We will begin in Section 3.4.1 by using a gap-producing reduction to show hardness of approximation in the case where all tiles are immobile. In Section 3.4.2, we will then show the weaker result of decision NP-hardness in the case where tiles may be moved by the player.

3.4.1 $\Theta(n)$-gap NP-Hardness Without Tile Movement

In this section, we show that for the *Patching* variant without tile movement, the gap problem of distinguishing between OPT = 4 tiles and OPT = $\Theta(n)$ tiles is NP-hard. We will do so via a *gap-producing* reduction from Planar Monotone Rectilinear 3SAT.

Gap-producing reductions are similar to the *gap-preserving* reduction, which was discussed in Chapter 2 as a weaker form of reduction than the L-reduction. The difference is that a gap-producing reduction starts with any NP-hard problem, not necessarily a gap decision problem, and *produces* a gap of b/a by showing that a YES instance to the original problem leads to an OPT of at least b, while a NO instance leads to an OPT of at most a.

Planar Monotone Rectilinear 3SAT [7] is an input restriction to 3SAT. Rather than the restrictions on variable occurrences analyzed in Chapter 2, Planar Monotone Rectilinear 3SAT requires the following:

- The graph formed by connecting all variables to clauses they appear in is planar.
- Each clause is either all-positive ($x_i \vee x_j \vee x_k$) or all-negative ($\neg x_i \vee \neg x_j \vee \neg x_k$) (monotone).
- The above graph can be drawn with the following conditions (rectilinear):
  - Variable nodes are disjoint vertical segments on the y-axis.
  - Clauses are vertical segments off the y-axis, with positive clauses to the right of the y-axis and negative clauses to the left.[10]
  - Edges, or *wires* between variables and clauses are horizontal segments connecting the corresponding vertical segments. No other intersections or overlap between segments is allowed, besides edge segments meeting their variable segment and clause segment.

Now, we present the reduction.

[10]This problem is canonically formulated with variables on the x-axis, positive clauses above, and negative clauses below; however, we alter the description for clarity to match our gadget diagrams.

Theorem 3.4.1. *The $\Theta(n)$-gap problem for Patching without tile movement is NP-hard.*

Proof. We reduce from Planar Monotone Rectilinear 3SAT.[11]

The central idea of this reduction is to leverage the rule against loose ends specifically only within the network. We create a situation where the player can either trivially create a network of size 4, or make the choice to try solving the 3SAT instance.[12] If we choose the latter, then all the wires connected to the network are in danger of creating loose ends, and the player is forced to choose the least-bad options to manage this.

Our variable gadget is shown in Figure 3.9. Here, the player is logically forced to set exactly one of x_i and $\neg x_i$, where False means in the network, and a True means disconnected from the network. Although the gadget is depicted as compactly as possible, it is clear that we can vertically spread out variable initialization and fanouts as much as needed to approximate the y values of horizontal wires in the Planar Monotone Rectilinear 3SAT instance.

Our clause gadget is shown in Figure 3.10. All the positive clauses will appear to the right of the variable gadget, and all negative clauses will appear to the left. As with the variable gadget, the input wires can have vertical spacing adjusted to match the incoming y values of horizontal wires. The clause is satisfied if there are no loose ends on any tile in the network, so the game rules require all clauses to be satisfied.

This clause takes inputs x_i, x_j, x_k, and is designed with a *key decision tile* (an L piece marked with a red circle in Figure 3.10). Two inputs to the key decision tile are x_i and x_j directly. The other two are x_k or a safe dead end in some order—there is a separate choice to allow x_k to choose its input, but if x_k is in the network, it must connect to the key decision tile. Many examples of solutions to the clause gadget are shown in the figure, but we can analyze the cases exhaustively:

- If all of x_i, x_j, x_k are True, the entire gadget is disconnected from the source, so there's no risk of loose ends in the network. The clause is trivially satisfied.
- When only x_k is True, we simply route x_i and x_j to each other through the key decision tile.
- When only x_i or only x_j is True, we route x_k to the key decision tile in the position that allows it to join with the other of x_i or x_j.
- When x_k and one of x_i or x_j is True, we route the other into a dead end through the key decision tile.
- When x_i and x_j are True, we can route x_k into the key decision tile through one of its positions, then routing a dead end to the other position.
- Finally, when all of x_i, x_j, x_k are 0, the piece circled in red will have 3 half-edges adjacent to it, each part of the network. However, it cannot handle all of them because it only has degree 2. Thus, this is the only case where the clause is not satisfiable.

[11] Or just Planar Monotone 3SAT, with slightly more effort.

[12] We can even eliminate the choice of creating a small network, which makes the decision problem of creating any size network NP-hard.

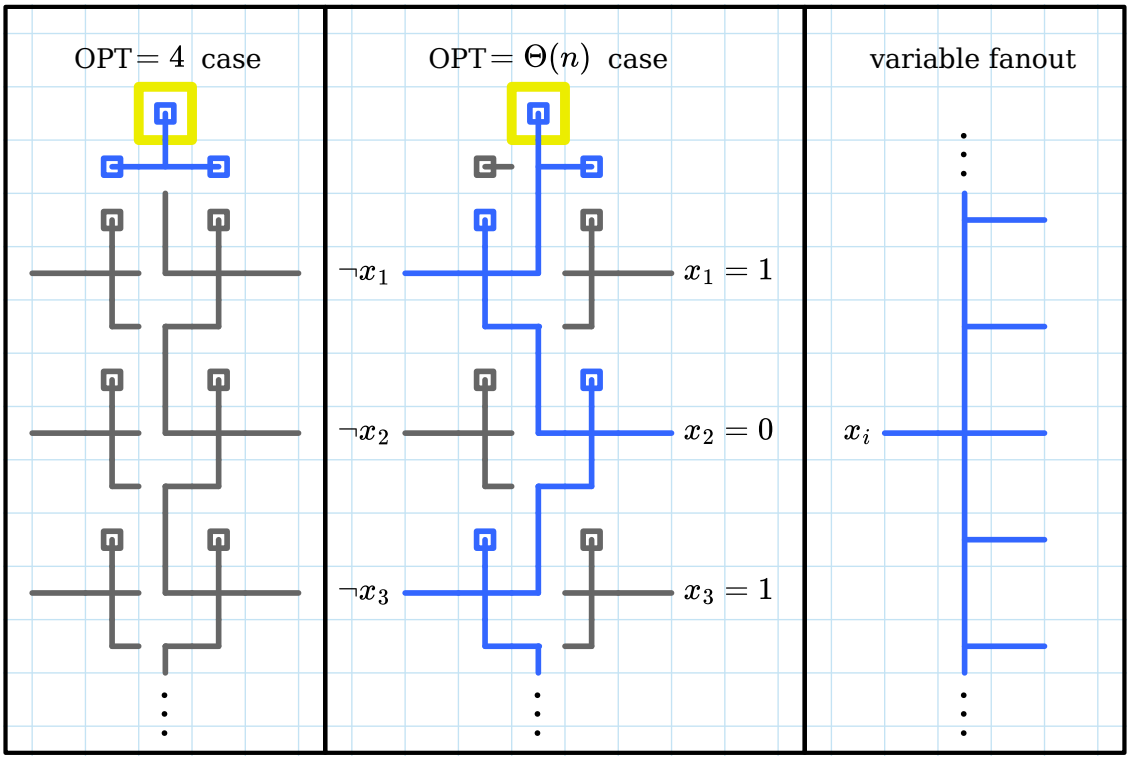

Figure 3.9. Variable and fanout gadgets for the gap reduction from Planar Monotone Rectilinear 3SAT, for the *Patching* puzzle in *Puzzle Pirates* without tile movement. Fanout is simple because the reduction is based on connectivity. When faced with the variable gadget, the player has a choice to either give up (shown left), or try to solve the Planar Monotone Rectilinear 3SAT instance (shown center). Tiles in the network (corresponding to False values) are shown in blue, and other tiles (corresponding to True values) are shown in gray. The source is boxed in yellow, and all Q pieces are marked with a small box as a visual aid to distinguish them from loose ends, which appear often in this construction.

Now, we analyze how many tiles are part of the network in the case where the 3SAT instance is solved. Note that for any given instance, we can place clauses arbitrarily far in the x-direction from variables and each other in the construction. This allows us to analyze the network size asymptotically as the proportion of literals in the 3CNF formula which evaluate to False under the assignment. Although False literals are not required to satisfy the formula, the proportion of False literals is still $\Theta(1)$ for Planar Monotone Rectilinear 3SAT instances, under some minor restrictions that do not interfere with NP-hardness.[13] □

[13]This aligns with intuition that the ratio of False literals clearly does not approach zero for virtually all useful 3SAT instances. The proof sketch: if we remove all size-1 clauses and any duplicate size-2 clauses, one can show by induction that (# clauses ≤ 6(# vars $- 2$)). Then, if we guarantee that each variable has at least one positive and one negative use by removing dummy variables, we can show that one wire for each variable must be False, which accounts for asymptotically $\geq 1/18$ of wires.

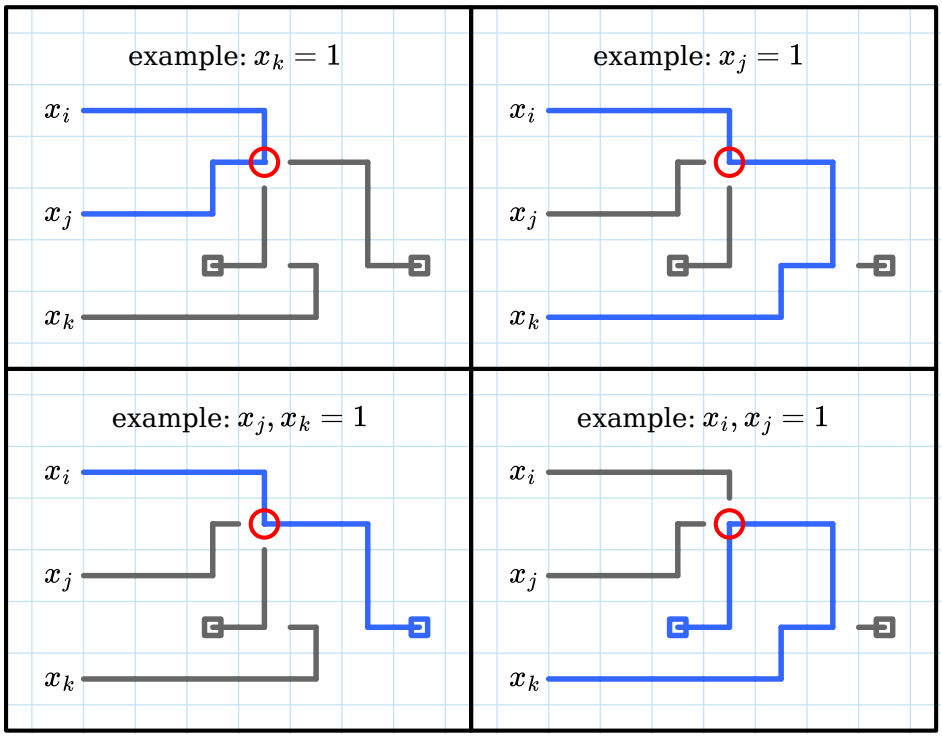

Figure 3.10. Example solutions to the clause gadget for the gap reduction from Planar Monotone Rectilinear 3SAT, for the *Patching* puzzle in *Puzzle Pirates* without tile movement. The clause takes as input three wires which are either in the network (False) or disconnected (True), and can be configured to leave no loose ends iff at least one wire is True. Tiles are colored as in Figure 3.9. The key decision tile is marked with a red circle.

3.4.2 Decision NP-Hardness With Tile Movement

In this section, we show that for the *Patching* variant with tile movement, blockers, and required sinks,[14] the decision problem of distinguishing between OPT $< k$ tiles and OPT $\geq k$ tiles is NP-hard, where k is taken as input.

An immediate fact of note for this problem is that the amount of information encoded in the movable *Net* pieces themselves is very low. If the player can rearrange tiles arbitrarily, then all that matters is how many of each piece type there are, so the amount of information encoded becomes logarithmic in the board size. Thus, to get NP-hardness, we must essentially encode all information in the immobile pieces.

We choose to reduce from Hamiltonian cycle on grid graphs of max degree 3, a problem discussed in Section 3.3.1 as it was used by [3] to show NP-hardness of *Net*.[15] This problem has the advantage that the reduction is very compact.

Theorem 3.4.2. *The decision problem for Patching with tile movement is NP-hard.*

Proof. We reduce from Hamiltonian cycle on grid graphs.

An example of a reduced instance is shown in Figure 3.11. Given an arbitrary grid graph $G = (V, E)$, we may embed it as follows:

- First, identify a *top-right corner* of G, that is, a vertex $(x, y) \in V$ such that $(x + 1, y), (x, y + 1) \notin V$.
- Define a grid large enough to comfortably fit G.
- Place a blank at each vertex coordinate of G.
- Place a blocker at each non-vertex coordinate of G, at least so that G is surrounded by one layer of blockers.
- At the selected top-right corner (x, y) of G, replace the blank with an X piece.
- Place the source at $(x, y + 1)$, and a required sink at $(x + 1, y)$, extending the grid and adding blockers if necessary.
- Add at least $|V| - 1$ L pieces and $|V| - 1$ I pieces to the grid, outside of G.

It is clear that if the Hamiltonian cycle problem has a solution, then we can use the L and I pieces to transcribe the solution onto the grid, achieving a maximum of $|V| + 2$ tiles if we are able to do so.

Now, suppose we have a solution producing a network of $|V| + 2$ tiles. The X piece must always be adjacent to the source and required sink, unless they are connected directly with an L piece, which gives a network of 3 tiles. Then, any tile we place within the bounds of G

[14] Use of required sinks (tie-offs) is only a convenience for this reduction, but may be useful for future work.

[15] Use of max degree 3 is also only a convenience, for ensuring the X piece placement in edge cases.

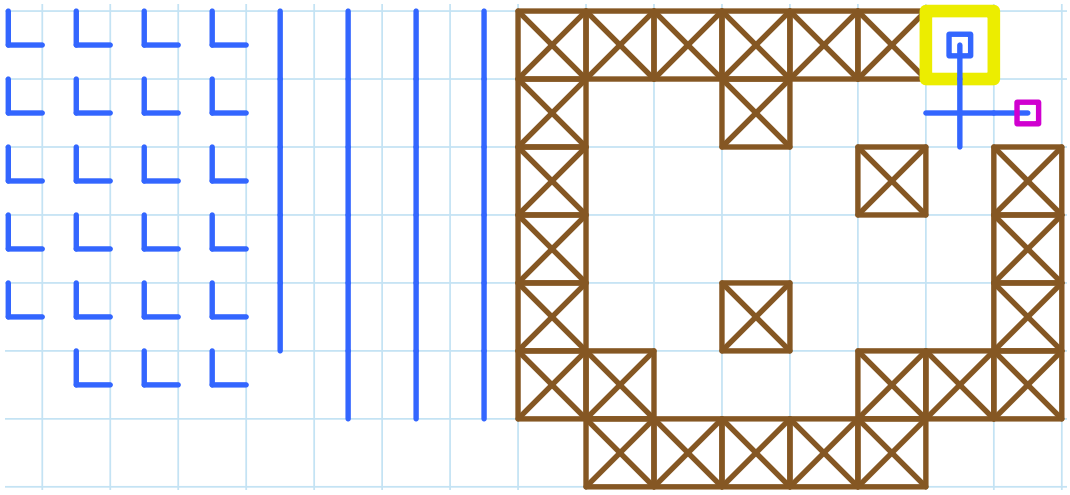

Figure 3.11. An example reduced instance for the reduction from Hamiltonian cycle on grid graphs of max degree 3, for the *Patching* puzzle in *Puzzle Pirates* with tile movement. Blockers are shown as brown boxes with Xs, the source is boxed in yellow, and the required sink is the small pink square. Other pieces are drawn in blue, with most pieces in a "piece bank" to the left.

must have degree 2, as the only pieces left are L and I pieces. Since loose ends are disallowed, we can directly extract a cycle in G starting and ending at the X piece. If the network has $|V| + 2$ tiles, then this cycle must be a Hamiltonian cycle. □

References

[1] Simon Tatham, *Simon Tatham's Portable Puzzle Collection*, 2020. URL: https://www.chiark.greenend.org.uk/~jharvey/puzzles/puzzles.html.

[2] D. Král', V. Majerech, J. Sgall, T. Tichý, and G. Woeginger, "It is tough to be a plumber," *Theoretical Computer Science*, vol. 313, no. 3, pp. 473–484, Feb. 2004, ISSN: 03043975. DOI: 10.1016/j.tcs.2002.12.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S0304397503005954.

[3] M. D. Biasi, "The complexity of the puzzle game Net: rotating wires can drive you crazy," Dec. 2012. URL: https://www.nearly42.org/vdisk/cstheory/netnpc.pdf.

[4] Three Rings Design, *Puzzle Pirates*, PC, [Online; accessed 2024-04-27], 2003. URL: https://www.puzzlepirates.com/.

[5] YPPedia contributors, *Patching — YPPedia*, https://yppedia.puzzlepirates.com/index.php?title=Patching&oldid=773695, [Online; accessed 2024-04-27], 2020.

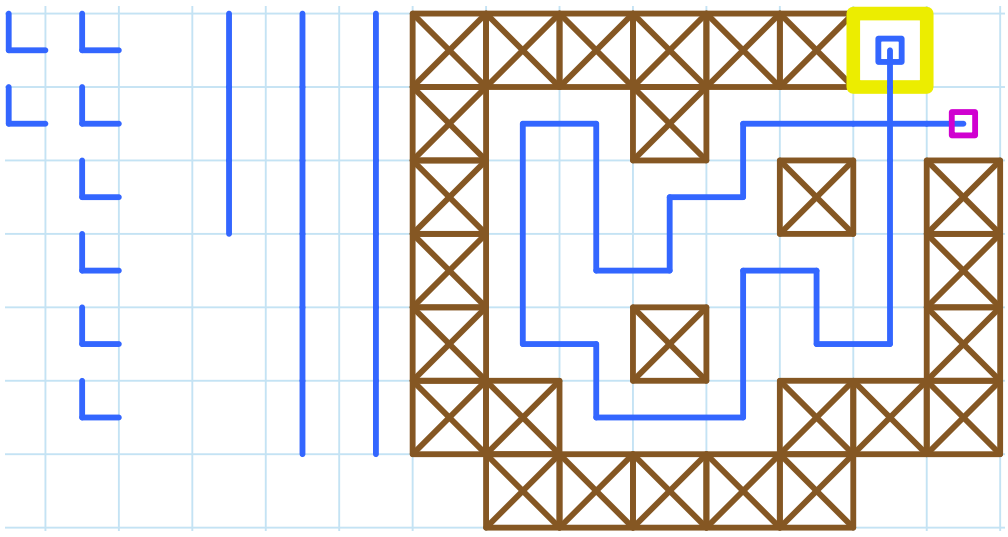

Figure 3.12. The same instance shown in Figure 3.11, with the solution in place. Although there are pieces remaining to the left, this is the maximum network size that one can hope to achieve.

[6] E. D. Demaine and M. Rudoy, "Tree-Residue Vertex-Breaking: a new tool for proving hardness," in *Proceedings of the 20th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, Malmö, Sweden, Jun. 2018, 32:1–32:14. DOI: 10.48550/arXiv.1706.07900. arXiv: 1706.07900 [cs.CC].

[7] M. de Berg and A. Khosravi, "Optimal binary space partitions in the plane," in *Proceedings of the 16th annual international conference on Computing and combinatorics*, ser. COCOON'10, Berlin, Heidelberg: Springer-Verlag, Jul. 2010, pp. 216–225, ISBN: 9783642140303.

Chapter 4

Euclidea

Euclidea [1] is a popular mobile puzzle game revolving around *Euclidean constructions*, constructions of various geometric objects using only an unmarked straightedge and compass. The game currently features 156 levels, with challenges ranging from finding the midpoint of a line segment, to finding the regular octagon inscribed in a given square, or constructing an angle of 3°.

In this chapter, we analyze the computational complexity of *Euclidea*. In Section 4.1, we explain the game mechanics and formally define *Euclidea* as an optimization problem, with two natural optimization metrics given by the game design. In Section 4.2, we show that both of the natural optimization problems are APX-hard, and by extension, the decision versions of the problems are NP-hard. In Section 4.3, we define solution-checking more rigorously, and discuss challenges that arise when trying to upper bound the complexity of *Euclidea*.

This chapter represents primarily individual work, with contributions from Erik Demaine, Josh Brunner, Lily Chung, and Jenny Diomidova, the instructor and TAs of the MIT class Algorithmic Lower Bounds: Fun with Hardness Proofs (6.5440), taught by Erik Demaine in Fall 2023.

4.1 Overview

In each level of *Euclidea*, the player begins with a *base configuration* of points, lines, and circles. The base configuration also permits line segments, but for ease of formalizing the problem, we will only consider infinite lines. From there, the *straightedge tool* allows the player to pick any two points A, B and draw the (infinite) line AB. The *compass tool* similarly takes two points A, B, and draws the circle centered at A passing through B. The goal of *Euclidea* is to construct one or more *target objects*, which have some desired geometric relation to the base configuration; for example, the circumcenter of a given triangle.

Certain constructions, such as perpendicular bisector and angle bisector, are useful enough that once the player completes the level constructing them, *Euclidea* provides these constructions to the player as tools. These tools are used similarly to the basic straightedge and compass tools, and function as a shortcut to make constructions simpler and visually

cleaner. Any construction using these tools is still a straightedge-and-compass construction, because each application of a tool can be simulated with the steps used to construct the tool.

The computational difficulty of *Euclidea* lies in finding the *shortest* solution to a level; this is presented as a main objective of the game, but is not a requirement to progress. The notion of a shortest solution is measured by two different metrics. A construction's length in *L-moves* is simply the number of tool applications, where any tool that *Euclidea* provides counts as a single move. A construction's length in *E-moves* also scores straightedge/compass moves as 1, but other tools have a cost equal to the number of moves required for that tool's construction. In other words, E-moves represent the length if only straightedge/compass moves were allowed.[1]

As an example, the orthocenter of a triangle ABC can be constructed by using the perpendicular bisector tool twice, once on AB and once on BC. This solution has a length of 2 in L-moves, or 6 in E-moves (denoted "2L 6E"), since one can use three straightedge/compass moves to construct each perpendicular bisector.

Players can earn up to four "stars" for each level, which represent objectives for the level and aid in progression. These stars are:

- A completion star for any construction of the target object or objects.
- The E-star for the shortest straightedge-and-compass-only construction (or shortest construction as measured by E-moves).
- The L-star for a shortest construction using any tools (as measured by L-moves).
- The V-star for constructing all possible target objects—for example, "construct an equilateral triangle with given edge" has two solutions.

The objectives for the E-star and L-star correspond directly to the player finding the optimal solution under the respective construction length metrics. In these terms, we show in this chapter that not only is it NP-hard to achieve an E-star or L-star, it is NP-hard to even approximate these solutions to better than a constant factor.

4.1.1 Problem Definition

In this section, we first introduce terms that will be used throughout the entire chapter, building up to the definitions of a construction, configuration, and level. We will then formally define the *Euclidea* optimization problem.

Although there are some simplifying assumptions made along the way, the definitions in this section are written in a very technical manner. The remainder of this chapter is written to be fairly understandable without the definitions, especially if the reader is familiar with

[1]Since any construction can be transformed into a pure straightedge-and-compass construction without changing the E-move value, this means that it's always optimal to only use the straightedge and compass, since using tools may cause the player to miss out on useful intersection points.

Euclidea. As such, we ask that this section be treated as a reference of terms, and as a basis for formalizing *Euclidea*.

Throughout the following definitions and the rest of the chapter, there will be many cases where we define a new point based on existing objects. In this case, we let *determined point* denote a new point that is the intersection of two or more existing lines or circles, so that its position is determined up to a constant number of possible locations. We then distinguish a *generic point on a line/circle* from just a *generic point*, where the former is constrained to lie on some object but is otherwise free, and the latter is entirely unconstrained in placement. We will use *non-determined point* to refer to a point which is either generic or generic on a line/circle.[2]

Definition 4.1.1. An *elementary move*, or *E-move*, is a single use of the straightedge or compass, where inputs may either be generic points in the plane, or constrained to lie on one or two previously constructed objects. An *L-move* is a single use of any tool provided in the game, including E-moves.

The additional tools provided by L-moves are:

- Perpendicular bisector, which takes as input two points A, B and draws their perpendicular bisector,
- Perpendicular line, which takes as input a point A and line ℓ and draws the line passing through A perpendicular to ℓ,
- Angle bisector, which takes as input three points A, B, C and draws the line passing through B bisecting angle $\angle ABC$,
- Parallel line, which takes as input a point A and line ℓ and draws the line passing through A parallel to ℓ, and
- Non-collapsing compass, which takes as input three points A, B, C and draws the circle centered at C with radius AB.

Definition 4.1.2. An *E- or L-construction* $\mathcal{C}$ is a sequence of point or line/circle definitions. Point definitions take in a list of 0, 1, or 2 existing line/circle objects, and define a (generic, generic on an object, or determined) point that lies on those objects. Line/circle definitions take in the inputs to an E- or L-move, respective to the construction type, and defines the line or circle which is the output of that move.

Essentially, a construction is a sequence of moves assumed to be performed from an empty plane, starting by defining generic points, and it encodes information on how each move depends on previously-constructed objects. A construction's *cost* in E- or L-moves does not include point definitions.

[2]Since constructions use determined points, for the most part, *Euclidea* will highlight any non-determined point red, to ensure that placing such a point was an intentional choice by the player.

Now, we need to define the configuration that the player starts with, as well as the target objects that the player is trying to construct. In *Euclidea*, the base configuration may need to have certain geometric properties; for example, some levels may start the player with a square, but no circles which would have been required to construct the square. Thus, we cannot just define the base configuration as some construction, but must define it as a subset of the constructed objects.

The easiest way to define target objects is in terms of a construction involving the base configuration, especially since we would like to enforce that constructing target objects is possible. One convenient solution, then, would be to have a single construction $\mathcal{C}$, with target objects and base configuration objects marked. To do this, we must ensure that the target objects are constructible using only the base configuration. The following definitions formalize this.

Definition 4.1.3. Define the *dependency partial order* $\prec$ on a construction $\mathcal{C}$ to be the transitive completion of the following relation: if object Y is used as a direct input to construct X in $\mathcal{C}$, then $Y \prec X$.[3]

In a construction, an object is *determined by* a set of previously-constructed objects if its location depends only on those objects, and is determined uniquely by those objects.[4] More formally,

Definition 4.1.4. In a construction $\mathcal{C}$, a set of objects $\mathcal{T}$ is *determined from* another set of objects $\mathcal{V}$ if for all $X \in \mathcal{T}$ and all $Y \preceq X$, either:

1. For some $Z \in \mathcal{V}$, $Y \preceq Z \prec X$, or
2. Y was constructed after all elements of $\mathcal{V}$, and for some $Z \in \mathcal{V}$, $Z \prec Y$.

The *cost* of constructing $\mathcal{T}$ from $\mathcal{V}$ is the total number of distinct line/circle objects Y which are verified by this process and do not fall into case (1) above.

Essentially, the definition of "determined from" allows tracing the antecedents of $\mathcal{T}$ back entirely to objects in $\mathcal{V}$, to get a construction starting with objects in $\mathcal{V}$ and ending with all objects in $\mathcal{T}$. The cost is a formal way of counting the number of objects used to make this construction, in the usual way. The condition that intermediate values must be constructed after all elements of $\mathcal{V}$ is only added for ease of definition of the *Euclidea* problem, where we must consider alternate constructions of the target objects.

Definition 4.1.5. A *base configuration* is a tuple $\mathcal{B} = (\mathcal{C}, \mathcal{V})$, where $\mathcal{C}$ is an E-construction, and $\mathcal{V}$ is a subset of the objects of $\mathcal{C}$, to be marked *visible*. The objects not marked visible are called *hidden*.

An *E- or L-construction from* $\mathcal{B} = (\mathcal{C}, \mathcal{V})$ is any further sequence of E- or L-moves $\mathcal{C}_+$ such that the combination $\mathcal{C} + \mathcal{C}_+$ is a valid construction, and where the set of all objects defined in $\mathcal{C}_+$ is determined from $\mathcal{V}$.

[3] We consider dependency to be a symbolic relation for constructions, and ignore edge cases where a dependency may disappear due to geometric invariants.

[4] Up to a constant number of solutions.

Finally, we define a *Euclidea* level and the *Euclidea* optimization problem for both E-moves and L-moves:

Definition 4.1.6. Let a *Euclidea level* be a tuple $\mathcal{L} = (\mathcal{C}, \mathcal{V}, \mathcal{C}_+, \mathcal{T})$, where $\mathcal{B} = (\mathcal{C}, \mathcal{V})$ is a base configuration, $\mathcal{C}_+$ is an E-construction from $\mathcal{B}$, and $\mathcal{T}$ is a subset of the objects defined in $\mathcal{C}_+$. $\mathcal{T}$ denotes the *target* objects of the level.

With these definitions, the *Euclidea* E- or L-move optimization problem is as follows. Given a *Euclidea* level $\mathcal{L} = (\mathcal{C}, \mathcal{V}, \mathcal{C}_+, \mathcal{T})$, a solution $(\mathcal{C}'_+, \mathcal{T}')$ is valid if:

1. $\mathcal{C}_+$ is an E- or L-construction from base configuration $\mathcal{B} = (\mathcal{C}, \mathcal{V})$,
2. $\mathcal{T}'$ is a subset of the objects defined in $\mathcal{C}'_+$, and
3. There is a bijection between $\mathcal{T}$ and $\mathcal{T}'$ such that corresponding objects always have the same mathematical position, regardless of the exact positions of any non-determined point in $\mathcal{C}$, $\mathcal{C}_+$, or $\mathcal{C}'_+$.

What is the minimum possible E- or L-move cost of constructing $\mathcal{T}$ from $\mathcal{V}$?

4.2 *Euclidea* is APX-Hard

In this section, we prove that the E-move and L-move *Euclidea* optimization problems are APX-hard.

In Section 4.2.1, we define our main reduction to show APX-hardness for the E-move optimization problem. In Section 4.2.2, we then discuss issues with rigorously proving correctness of the previous reduction, and define a simpler one. We also note that this simplified reduction shows APX-hardness for the L-move optimization problem, as well. Finally, in Section 4.2.3, we briefly discuss the extent to which our assumptions generalize *Euclidea*.

4.2.1 E-Move APX-Hardness

In this section, we define an L-reduction[5] from Cubic Max Vertex Cover, which is known to be APX-complete [2]. In this reduction, we intentionally treat leniently the proof that intended solutions to the reduced *Euclidea* instance are optimal. This issue, as well as a simplified reduction that trades elegance and realism for rigor, will be presented in Section 4.2.2.

Max Vertex Cover is defined as follows. Given a graph $G = (V, E)$, what is the smallest subset $V' \subseteq V$ of vertices such that every edge $e \in E$ is incident to a vertex in V'?

Cubic Max Vertex Cover is an input restriction on this problem, requiring that G is a cubic graph.

Theorem 4.2.1. *The Euclidea E-move optimization problem is APX-hard.*

[5]Not related to L-moves in *Euclidea*; L-reductions were discussed primarily in Chapter 2.

Proof. We show an L-reduction from Cubic Max Vertex Cover.

Suppose we have an input cubic graph $G = (V, E)$ to Vertex Cover. The idea is to define a reduced *Euclidea* instance which embeds G, where we can perform some moves to select a set $V' \subseteq V$ of vertices, which will save us some moves if V' is a vertex cover.

As a high-level overview, there are a few key elements necessary for this reduction to work:

- An edge must be satisfied with being covered by either incident vertex,
- A vertex must cover all its edges in a constant number of moves,
- Bypassing Vertex Cover by covering an edge $V_iV_j \in E$ manually must cost at least as much as covering a vertex, and
- The length of the solution must increase linearly with the size of the vertex cover.

Now, we proceed with the reduction. First, we will embed G generically in the plane, possibly with crossings; let $(V = \{V_1, \ldots, V_n\}, E)$ be this embedding. Our goal will be to construct the circumcenter of a triangle with base V_iV_j, for each edge $V_iV_j \in E$. Figure 4.1 indicates how the desired properties of the reduction will be satisfied—a circumcenter can be constructed with any two perpendicular bisectors, and we will have enough freedom to place the third vertex P_{ij} of the triangles to satisfy the others.

The base configuration $\mathcal{B} = (\mathcal{C}, \mathcal{V})$ is specified as follows:

- Embed G as (V, E) generically in the plane, as stated above. This can be done by placing all vertices in V as generic points, and constructing lines V_iV_j for all $V_iV_j \in E$.
- For each vertex V_i, pick radius r_i arbitrarily such that for all i, j, $r_i + r_j > |V_iV_j|$. For all i, construct the circle C_i centered at V_i with radius r_i.
- For all $V_iV_j \in E$, let P_{ij} be one of the two intersections of C_i and C_j, chosen arbitrarily. Construct the lines V_iP_{ij} and V_jP_{ij}.
- Finally, construct the perpendicular bisector of all $V_iV_j \in E$, (denoted $L(V_i, V_j)$ in Figure 4.1).
- At this point, the construction $\mathcal{C}$ is composed of the above operations. Mark the circles C_i hidden, and let the visible objects $\mathcal{V}$ consist of all other objects.

The target object set $\mathcal{T}$ is the set of circumcenters T_{ij} of triangles $\triangle V_iV_jP_{ij}$, for all $V_iV_j \in E$. It is trivial to find a sequence of moves $\mathcal{C}_+$ to specify them, by further constructing the perpendicular bisectors of all edges V_iP_{ij}. This completes the definition of the *Euclidea* level.

Given a valid solution $(\mathcal{C}'_+, \mathcal{T}')$ to the *Euclidea* instance, we extract a vertex cover by taking the set of vertices V_i where the circle C_i was drawn in the sequence $\mathcal{C}'_+$. If any

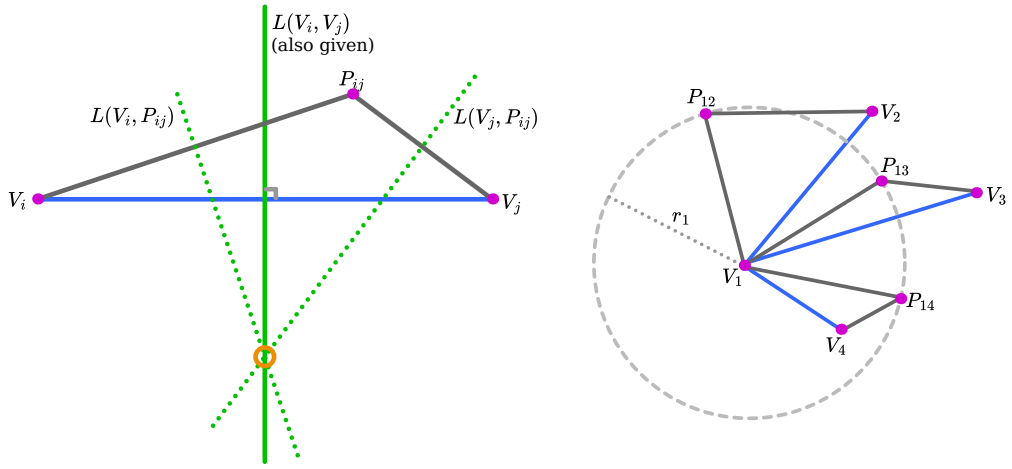

Figure 4.1. Left: An edge V_iV_j can be covered by either of its vertices, since either perpendicular bisector (dotted green) can construct the circumcenter. $L(A, B)$ denotes the perpendicular bisector of the line segment AB. Right: A single move at a vertex (C_1 centered at V_1) can cover all edges adjacent to the vertex, due to our construction.

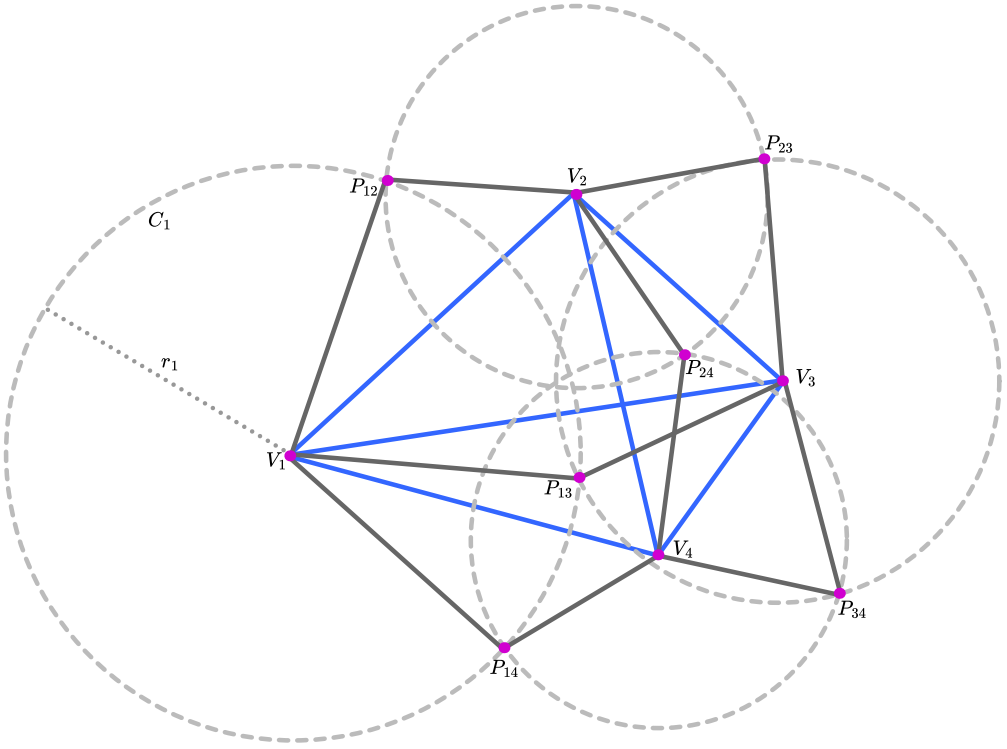

Figure 4.2. A reduced *Euclidea* instance for input $G = K_4$, in the L-reduction from Cubic Max Vertex Cover to the *Euclidea* E-move optimization problem. The circles C_i (shown dashed, light gray) are not included in the base configuration. The perpendicular bisectors $L(V_i, V_j)$ shown in Figure 4.1 are included in the base configuration, but are omitted here for clarity.

circumcenter T_{ij} was constructed without drawing the corresponding C_i or C_j, add one of V_i or V_j arbitrarily to the vertex cover.

Now, we demonstrate this construction is a valid L-reduction.

Given a valid vertex cover V' with $|V'| = k$, we can draw C_i for all $V_i \in V'$ in k moves, a subset of the circles which were hidden in the base configuration. Then, for each edge V_iV_j, assume without loss of generality that $V_i \in V'$. We use two more moves per edge to construct the perpendicular bisector $L(V_i, P_{ij})$. This constructs all circumcenters T_{ij} in $k' = 2|E| + k$ E-moves, showing that $\text{OPT}_{\text{Euclidea}} \leq 2|E| + \text{OPT}_{\text{Vertex Cover}}$. Because G is cubic, one vertex can cover at most three edges, so we have $|E| \leq 3\text{OPT}_{\text{Vertex Cover}}$, and

$$\text{OPT}_{\text{Euclidea}} \leq O(\text{OPT}_{\text{Vertex Cover}}).$$

In the other direction, suppose we have a valid k'-move construction. One of *Euclidea*'s levels constructs the circumcircle optimally in 3L 7E, which implies that two perpendicular bisectors is the optimal way to construct the circumcenter [3]. One perpendicular bisector is already given, so at least three E-moves must be used to construct any given circumcenter T_{ij}.

Now, we argue that the only way constructions of different circumcenters T_{ij} can share moves is via the circles C_i. Note that the triangle $\triangle V_iV_jP_{ij}$ only depends on the initial settings of V_i, V_j, r_i, r_j. If we fix these values and can vary all other settings, any object with a dependency on V_k for $k \neq i, j$ will vary as well.

Thus, what we wish to prove is that even if the set of dependencies of T_{ij} contains objects $X \prec T_{ij}$ which depends on some other value besides $\{V_i, V_j, r_i, r_j\}$, it still requires at least 3 objects specific to i and j, at most one of which is C_i or C_j. This would allow us to prove an upper bound on the size of a vertex cover generated from a valid solution to the *Euclidea* instance—specifically, for a solution of length k', the vertex cover would have size $k \leq k' - 2|E|$.

The potential issue, in general, is that X may be the result of some expensive computation which depends on no inputs, but assists in constructing T_{ij}. Then, even if X could potentially save just one move in constructing T_{ij}, this translates to an $O(n)$ benefit for a one-time cost.

We take as assumption that the construction of T_{ij} should only depend on inputs related to i and j, for the purposes of this proof, and defer discussion to Section 4.2.2.

In this chapter, we omit the detailed L-reduction calculations. If the above assertions hold, then we can use the bound on the vertex cover size for the mapped solution to prove that $\text{OPT}_{\text{Euclidea}} = 2|E| + \text{OPT}_{\text{Vertex Cover}}$, and

$$k - \text{OPT}_{\text{Vertex Cover}} = O(k' - \text{OPT}_{\text{Euclidea}}).$$

Under the above assumption, this completes the proof that the *Euclidea* E-move optimization problem is APX-hard.

□

4.2.2 Simplified E-Move Reduction and L-Move APX-Hardness

In this section, we mainly discuss difficulties in the reduction given in Section 4.2.1. We then present a simplified reduction that circumvents these issues, and show that this simplified reduction works for the L-move problem, as well.

The main issue with the reduction is that in our solution, we are not just constructing a single circumcenter T_{ij}. We are also constructing other circumcenters, or could even just perform an independent, expensive computation that doesn't depend on any input, and then use these results to construct T_{ij} faster. Even though we know that T_{ij} only depends on select inputs, it is difficult to even prove that the independent computation cannot assist in constructing T_{ij} in fewer moves than the optimal solution, if the independent computation's moves are free.

As alluded to earlier, we must prove this for the reduction to work, because we can effectively treat an independent computation's moves as cost 0. If a computation is independent of any inputs and saves a move in constructing a circumcenter T_{ij}, then by symmetry, it saves a move for all circumcenters. One needs only to run this computation once, and then they can reap the benefits by saving $O(n)$ moves for a fixed cost.

Although we don't have a perfect solution to this issue, we circumvent it by further simplifying the reduction, at the cost of similarity to real *Euclidea* gameplay.

Simplified proof of Theorem 4.2.1. We modify the reduction for E-moves to simplify the construction by designating P_{ij} directly to be the target objects, instead of the circumcenters.[6] Because P_{ij} was only the intersection of two objects, C_i and C_j, we must then provide an extra object passing through it, so that only one of C_i and C_j is needed. We must also provide an additional arbitrary point R_i on C_i in the base construction, so that the player can draw a circle of the correct radius.

The reduced problem is now as follows. Given an embedding $(V = \{V_1, \ldots, V_n\}, E)$ of graph G, and for each i, a point R_i of distance r_i from V_i, construct the points P_{ij} for all $V_iV_j \in E$ such that $|V_iP_{ij}| = r_i$.

Formally, the base configuration is specified as follows:

- Embed G as (V, E) generically in the plane, as before. This time, the edges V_iV_j don't need to be constructed.
- For each vertex V_i, pick radius r_i arbitrarily such that for all i, j, $r_i + r_j > |V_iV_j|$.
- For all i, construct the circle C_i centered at V_i with radius r_i. Construct R_i as a generic point on C_i.
- For all $V_iV_j \in E$, let P_{ij}, P'_{ij} be the two intersections of C_i and C_j. Construct the line $P_{ij}P'_{ij}$.
- Mark the circles C_i and points P_{ij}, P'_{ij} hidden, and all other objects visible.

[6] Another very similar option embeds $G = (V, E)$ by drawing each vertex as a generic line in the plane, and the goal is to construct all line intersections corresponding to edges in E.

The target objects are the P_{ij} (one or both) for each $V_iV_j \in E$, and are already specified by hidden objects in the base configuration.

In this case, the proof of correctness follows more easily than in the previous reduction. This is mainly because all that is required to construct the target objects is to reinstate the circles C_i corresponding to a vertex cover.

Specifically, because P_{ij} is not already constructed, the cost of constructing it, relative to the base configuration, must be at least 1 move. Consider the set of inputs this move can depend on. Because P_{ij} is sensitive to the inputs V_i, V_j, r_i, r_j, and no object in the base configuration captures the entire dependency (not even the line $P_{ij}P'_{ij}$, as r_i and r_j can change together), the move constructing P_{ij} must depend on a nonempty subset of $\{V_i, V_j, r_i, r_j\}$.

Now, given a solution to the *Euclidea* instance, we can extract a vertex cover as follows: for each constructed P_{ij}, add V_i if the move constructing it depended on V_i or r_i, and respectively for j. If the move depended on both, pick whichever of V_i or V_j has lower index.

This ensures that the vertex cover's size is upper bounded by the number of distinct moves used to construct all P_{ij}. It is a valid vertex cover, since each P_{ij} corresponds to an edge, and adds either V_i or V_j to the vertex cover. This allows us to finish the L-reduction as before. □

Finally, we address the L-move optimization problem, using the above reduction.

Theorem 4.2.2. *The Euclidea L-move optimization problem is APX-hard.*

Proof. We note that the above simplified L-reduction works equivalently if we replace E-moves with L-moves. □

4.2.3 Generalizing Assumptions

In this section, we briefly discuss the worry that we may be overgeneralizing *Euclidea*, for the sake of proving hardness.

One interesting dimension we can judge the previous reductions on is how much they feel like real geometric constructions, or how in the spirit of *Euclidea* they feel. It turns out that although our definition of *Euclidea* feels like a generalization, closer inspection reveals that the original game shares some of these idiosyncrasies as well.

One salient difference is the construction of many target objects, whereas *Euclidea* mostly provides a single object or polygon as a goal. In the way we've defined the problem, however, it's difficult to draw a distinction between a set of objects forming a polygon from other arbitrary sets of objects. In addition, some levels ask to construct multiple segments outside the context of polygons.[7]

Another major seemingly-artificial step was the hidden objects in the base configuration—without the circles C_i, it seems unnatural to promise that all segments V_iP_{ij} are equal for a given i. However, as previously discussed, *Euclidea* has many levels where the base configuration is a square, or other polygon. Under our definition, there must have been

[7]See Euclidea 3.4 "Three equal segments - 1", or Euclidea 7.5 "Heron's Problem".

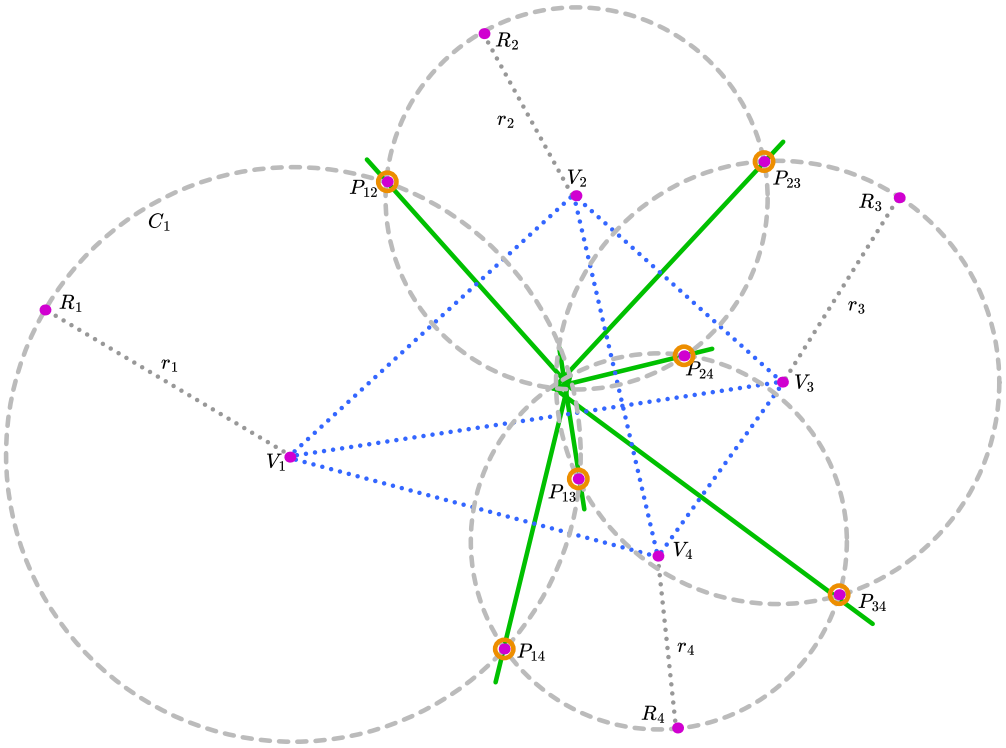

Figure 4.3. A reduced *Euclidea* instance for input $G = K_4$, in the simplified L-reduction from Cubic Max Vertex Cover to the *Euclidea* E- or L-move optimization problem. The target objects are the circled points P_{ij}. Lines passing through P_{ij} are given in the base configuration so that only one of C_i or C_j needs to be drawn to construct the point. Points R_i are included to ensure that any C_i can be drawn in a single move. In this reduction, not only the circles C_i (shown dashed, light gray) but also the edges V_iV_j (shown dotted, blue) are not included in the base configuration.

some hidden objects aiding the construction of the square, as well. Under this lens, the base configurations used in the reductions aren't actually as different from usual as one might expect.

4.3 In Search of a Complexity Upper Bound

In this section, we describe some of the challenges that plagued attempts to upper bound the complexity of *Euclidea*, and show that verifying a construction is in BPP if and only if deciding equality of constructible numbers is.

Let us begin by considering the obvious certificate to the *Euclidea* optimization problem: a valid construction of the target objects with length at most k. This certificate's size is certainly polynomial in the input size, since at least k moves were taken to construct (and thus define) the target objects. However, verifying the certificate amounts to checking that the constructed objects and target objects are identical, which is not obviously in P for two reasons:

- The equality between objects must hold regardless of the positions of generic points in the construction. This means that verification is more like comparing equality of functions than of values.
- This function is not necessarily polynomial either (see Figure 4.4)—it can be represented by an arithmetic circuit using the operations $+, -, \times, /, \sqrt{\cdot}$. In other words, the function returns *constructible numbers* as long as the inputs are themselves constructible.

Given this, it seems natural to aim for placing certificate verification in BPP, which would place *Euclidea* in Merlin-Arthur. This is a natural choice because Polynomial Identity Testing is one of the remaining problems known to be in BPP but not known to be in P. It seems intuitive, as well, that one should be able to verify constructions with bounded error probability.

However, when putting the above issues together, it becomes difficult to even place construction verification in BPP. Instead, we will present a sketch of a proof that verifying constructions lies in BPP if and only if equality of constructible numbers is in BPP. The latter is, to our knowledge, still an open problem.[8]

First, it's well-known that given a segment of length 1, it's possible to construct a segment of length x for any constructible number x (which is where the name comes from). Thus, verifying constructions in BPP trivially implies comparing constructible numbers in BPP.

Now, suppose we have an algorithm for comparing constructible numbers in BPP. Then, let the inputs $x_1, \ldots, x_m$ represent the construction's degrees of freedom, and consider the minimal polynomial describing an output value x. It has up to exponential degree, and its coefficients may be up to exponential-degree polynomials in the construction's inputs.

[8] A special case of this, sums of radicals, was shown to be in BPP by [4].

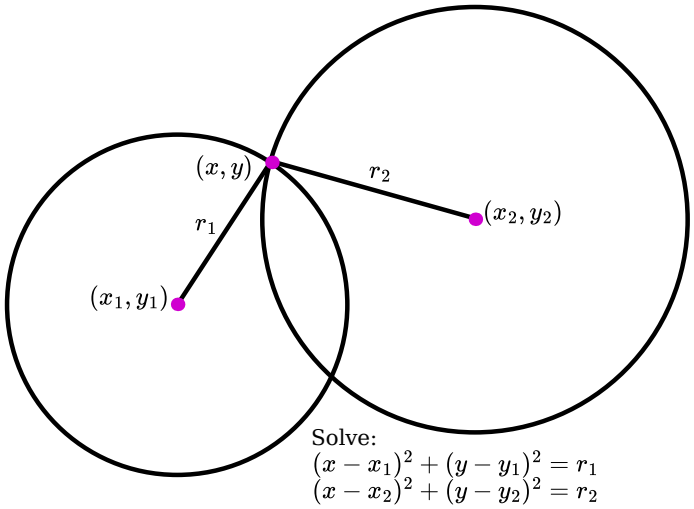

Figure 4.4. Illustration of the system of equations arising from two intersecting circles. When solving for (x, y) in terms of inputs, the expression will contain square roots. It's also possible to set x or y equal to a polynomial of exponential degree, or have them be roots of a polynomial of exponential degree.

Suppose now that we only allow the input to vary along one degree of freedom x_i, so the minimal polynomial reduces to a plane curve $p(x, x_i) = 0$ of exponential degree d. Consider the analogous plane curve for the target object corresponding to x, $p'(x, x_i) = 0$ of degree d'. By Bézout's theorem, these two curves must either be identical, or intersect in no more than dd' points. Assuming the degrees are bounded above by 2^n, the number of intersection points is $\leq 2^{2n}$.

To verify a construction, begin by fixing the input $x_1, \ldots, x_m$ except for one particular x_i. Use $O(n)$ bits of randomness to pick a random value of x_i from $|S| \gg 2^{2n}$ options. Evaluate the constructed objects and target objects as constructible numbers, and use the BPP algorithm to compare them with bounded error probability. There is an additional possibility of error if the constructed and target objects actually happen to be equal for this choice of x_i. However, this happens with bounded error as well, because it corresponds to hitting an intersection point between the plane curves, which happens with probability $\leq 2^{2n}/|S| \ll 1$.

References

[1] *Euclidea - Geometric Constructions Game with Straightedge and Compass*. URL: https://www.euclidea.xyz/.

[2] P. Alimonti and V. Kann, "Some APX-completeness results for cubic graphs," *Theoretical Computer Science*, vol. 237, no. 1, pp. 123–134, 2000, ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(98)00158-3.

[3] Euclidea Wiki contributors, *Circumscribed Circle*, https://euclidea.fandom.com/wiki/Circumscribed_Circle, [Online; accessed 2023-09-24], 2022.

[4] J. Blomer, "Computing sums of radicals in polynomial time," in *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, 1991, pp. 670–677. DOI: 10.1109/SFCS.1991.185434.