Easier Ways to Prove Counting Hard: A Dichotomy for Generalized #SAT, Applied to Constraint Graphs

MIT Hardness Group[1]
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA

Josh Brunner
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA

Erik D. Demaine
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA

Jenny Diomidova
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA

Timothy Gomez
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA

Markus Hecher
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA

Frederick Stock
University of Massachusetts, Lowell, MA 01854, USA

Zixiang Zhou
MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA

Abstract

To prove #P-hardness, a single-call reduction from #2SAT needs a clause gadget to have exactly the same number of solutions for all satisfying assignments — no matter how many and which literals satisfy the clause. In this paper, we relax this condition, making it easier to find #P-hardness reductions. Specifically, we introduce a framework called Generalized #SAT where each clause contributes a term to the total count of solutions based on a given function of the literals. For two-variable clauses (a natural generalization of #2SAT), we prove a dichotomy theorem characterizing when Generalized #SAT is in FP versus #P-complete.

Equipped with these tools, we analyze the complexity of counting solutions to Constraint Graph Satisfiability (CGS), a framework previously used to prove NP-hardness (and PSPACE-hardness) of many puzzles and games. We prove CGS ASP-hard, meaning that there is a parsimonious reduction (with algorithmic bijection on solutions) from every NP search problem, which implies #P-completeness. Then we analyze CGS restricted to various subsets of features (vertex and edge types), and prove most of them either easy (in FP) or hard (#P-complete). Most of our results also apply to planar constraint graphs. CGS is thus a second powerful framework for proving problems #P-hard, with reductions requiring very few gadgets.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Complexity theory and logic

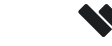
Keywords and phrases Counting, Computational Complexity, Sharp-P, Dichotomy, Constraint Graph Satisfiability

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.56

Funding *Markus Hecher*: The author was funded by the Austrian Science Fund (FWF), grants J4656, the Society for Research Funding in Lower Austria (GFF NOE) grant ExzF-0004, as well

[*] Artificial first author to highlight that the other authors (in alphabetical order) worked as an equal group. Please include all authors (including this one) in your bibliography, and refer to the authors as "MIT Hardness Group" (without "et al.").

35th International Symposium on Algorithms and Computation (ISAAC 2024).
Editors: Julián Mestre and Anthony Wirth; Article No. 56; pp. 56:1–56:14



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

as the Vienna Science and Technology Fund (WWTF) grant ICT19-065. Part of the research was carried out while visiting Simons Inst./UC Berkeley.

1 Introduction

Counting solutions to NP search problems (i.e., problems in the complexity class #P) is an algorithmic analog of the field of combinatorics. Counting (#P) is also provably harder than deciding (NP): by a consequence of Toda's Theorem [10], any problem in the polynomial hierarchy has a deterministic polynomial-time reduction to a single counting problem in #P. Already from the foundational work by Valiant in the late 1970s [13, 14], we have problems where decision is in P yet counting is #P-hard. One notable example is 2SAT, where finding one solution is easy, but counting the solutions (#2SAT) is #P-hard.

How do we prove a problem #P-hard? Ideally, we would find a ***parsimonious*** (single-call) reduction that preserves the number of solutions (and provides a polynomial-time bijection on the solutions). In addition to preserving #P-hardness, parsimonious reductions preserve a stronger property called ***ASP-hardness*** [15], meaning that all NP search problems have a parsimonious reduction to the problem. In addition to #P-hardness, ASP-hardness implies NP-hardness of the decision problem as well as the k-ASP problem: given k solutions to an instance, find another solution. The weaker notion of ***c-monious*** reduction increases the number of solutions by exactly a factor of c. Such a reduction implies both NP-hardness and #P-hardness (but not ASP-hardness), as zero-solution instances are preserved and we can recover the answer to our initial counting problem by dividing by c.

But even a c-monious reduction from #2SAT (say) can be difficult, compared to a standard NP-hardness reduction. A c-monious reduction built from standard variable and clause gadgets must have exactly the same number of solutions to every gadget; then c is the product of these counts. In particular, a clause gadget must have exactly the same number of solutions *no matter how it is satisfied* — no matter whether it is satisfied by the first clause, the second clause, or both clauses. This property often does not come without substantial effort. For example, in Lichtenstein's reductions from (planar) 3SAT to Hamiltonicity and vertex cover [6], the number of solutions to each clause is equal to the number of true literals that satisfy the clause. In this paper, we prove that such a reduction still establishes #P-hardness.

1.1 Our Results: Generalized #SAT

Specifically, we define a framework called ***Generalized #SAT***, where a clause can contribute a count of not just 0 or 1. A ***clause type*** is defined by a function from literal truth values to nonnegative integers, indicating the number of ways the clause is satisfied by that assignment. The number of solutions to the formula is then defined to be the product of the clause function outputs, summed over all possible assignments. In particular, if a clause function outputs zero, then that assignment still does not contribute to the number of solutions (similar to #SAT).

In Section 3, we present a complete complexity dichotomy of Generalized #SAT for the case of 2 variables per clause, precisely characterizing the computational hardness of every clause type as either in FP or #P-hard. In particular, Generalized #SAT is #P-complete for the function $f(x, y) = x + y$ counting the number of satisfying literals. Thus Lichtenstein's reductions [6], adapted to reduce from 2SAT instead of 3SAT, prove #P-hardness of Hamiltonicity and vertex cover. While these results have been proved in other

ways since [9, 7],[2] it is comforting to know that existing NP-hardness reductions can be more easily extended to #P-hardness, potentially saving time in the future. In some cases, a c-monious clause gadget may be very difficult or even impossible to construct, while Generalized #SAT provides the flexibility necessary for a reduction from #2SAT.

1.2 Our Results: Constraint Graph Satisfiability

We show the applicability of our Generalized #SAT framework by using it to solve another open problem: analyzing the complexity of counting solutions to the Constraint Graph Satisfiability (CGS) problem [3]. A ***constraint graph*** is a graph, usually 3-regular, together with edge weights of 1 and 2, also referred to as edge colors red and blue respectively. Given such a constraint graph, the goal in CGS is to find an orientation of the graph (direction of the edges) such that every vertex has a total incoming weight of at least 2. Constraint graphs are the foundation of Nondeterministic Constraint Logic (the reconfiguration version of CGS according to edge reversals), which is a popular framework for proving puzzles and games PSPACE-hard [3]. CGS was shown NP-complete over 15 years ago [3, Section 5.1.3], but the complexity of its counting problem remained unsolved.

In Section 4, we prove that CGS is ASP-complete in general, implying #P-completeness of #CGS. Then we analyze subproblems of #CGS, where the graph is restricted to only certain vertex and edge types, providing an almost-complete complexity characterization for these various subproblems, as summarized in Table 1. Specifically, there are three interesting degree-3 vertices in CGS:

1. **maj** (majority) vertices have three incident red edges (at least two of which must point in to reach an incoming weight of 2);
2. **or** vertices have three incident blue edges (at least one of which must point in); and
3. **and** vertices have two incident red edges and one incident blue edge (where the blue edge can point out only if both red edges point in).

The original NP-completeness proof of CGS [3, Theorem 5.4] uses just AND and OR vertices (while other Constraint Logic proofs in [3] use MAJ vertices, under the name "CHOICE"). In addition to restricting to an arbitrary subset of these vertex types, we can consider two types of edges. In ***matching edge weights***, each edge is uniformly red or blue, so both endpoints view the edge as having the same weight. In ***arbitrary edge weights***, we allow an edge to have red/weight 1 at one endpoint and blue/weight 2 at the other endpoint. In [3], the latter type of edge is called a "red-blue conversion". While red-blue conversion can be simulated with matching edge weights [3, Figure 2.4] this simulation is not parsimonious, so for our analysis the problems differ.

▶ **Conjecture 1.** *There exists no c-monious* RED-BLUE *conversion gadget unless* $\#P = FP$.

Fortunately, most reductions from Constraint Logic or CGS do not care whether edge weights are matching. Only vertex gadgets depend on their incident edge colors, while there is only one distinct edge gadget independent of color. Thus, arbitrary edge weights are of primary interest, and in this case, we provide a complete characterization of complexity. We start by showing that CGS with only AND vertices can be reduced from Generalized #SAT to show #P-hardness, while the decision problem is easy. If we have only MAJ vertices, then even counting is easy. (We leave open the case of only OR vertices.) Next, we show that when allowing all three vertex types, we can design a parsimonious reduction from Exactly

[2] The first proof, by Valiant [12], seems to have never been published.

AND	OR	MAJ	Edge Weights	CGS	#CGS
✓	✓	✓	Arbitrary	NP-c [3]	**ASP-c** (Thm. 15)
✓	✓	✓	Matching	NP-c [3]	(open)
✓	✓	×	Arbitrary	NP-c [3]	**#P-c** (Thm. 11)
✓	✓	×	Matching	NP-c [3]	(open)
×	✓	✓	Arbitrary	**P** (Thm. 17)	**#P-c** (Thm. 19)
×	✓	✓	Matching	**P** (Thm. 17)	(open)
✓	×	✓	Arbitrary	**P** (Thm. 18)	**#P-c** (Thm. 11)
✓	×	✓	Matching	**P** (Thm. 18)	**FP** (Thm. 21)
✓	×	×	Arbitrary	**P** (Thm. 18)	**#P-c** (Thm. 11)
✓	×	×	Matching	**P** (Thm. 18)	**FP** (Thm. 12)
×	✓	×	(Matching)	**P** (Thm. 17)	#P-c [1]
×	×	✓	(Matching)	**P** (Thm. 17)	**FP** (Thm. 13)

Table 1 Known and new results for Constraint Graph Satisfiability (CGS). The first three columns indicate allowed gadgets (✓). The fourth column specifies whether edge weights must be matching at either endpoint, or whether they can be arbitrary on either end (equivalently, allowing red-blue conversions). ASP-c(omplete) means that the reduction is parsimonious, which implies #P-c(ompleteness) as well. All hardness results except Theorem 19 can be encoded as planar graphs.

1-in-3 SAT, proving ASP- and #P-completeness. By a slight modification, the reduction can be made c-monious with only ANDs and ORs, achieving #P-completeness in this case. Finally, for all pairs of vertex types, we show that counting is easy.

Most of our hardness results hold when restricted to *planar* constraint graphs, making them particularly amenable for reductions to games and puzzles. The only exception is the case of OR and MAJ vertices with arbitrary edge weights, for which we have been unable to build a crossover gadget.

2 Preliminaries

2.1 Generalized #SAT

We define ***Generalized #SAT*** as follows. Each version of Generalized #SAT is specified by a ***clause type***, a nonnegative integer function $f\colon \{0,1\}^k \to \mathbb{Z}_{\geq 0}$, which describes the number of ways a given assignment of k literals satisfies the clause. We allow negations of variables, denoted with a bar (like $\bar{x}$), to be used freely in clauses. An input to Generalized #SAT consists of a number n, the number of variables (denoted $x = (x_1, \ldots, x_n)$), and a set of m clauses $C = \{\phi_1, \ldots, \phi_m\}$. Each ϕ_i is a k-tuple of literals, like $(x_{i_1}, \bar{x}_{i_2}, \ldots, x_{i_k})$. Let x_ϕ denote the restriction of a variable assignment x to the variables in a clause ϕ. The goal is to compute the number of ways to satisfy all clauses:

$$\sum_{(x_1,\ldots,x_n)\in\{0,1\}^n} \prod_{\phi\in C} f(x_\phi). \tag{1}$$

Previous work has proved that #2SAT [13] and #Max Cut [8] are #P-complete.

2.2 Constraint Graph Satisfiability

A ***constraint graph node*** (E, W, c) consists of a set E of incident edges, an assignment W of nonnegative weights to the edges, and a lower bound c on the total incoming weight.

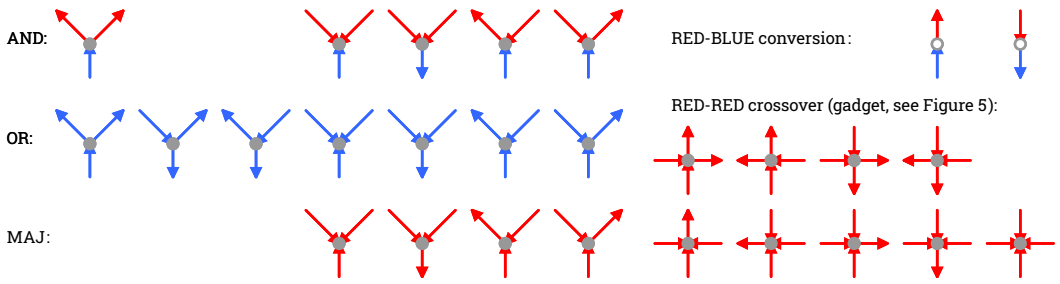


Figure 1 Overview of the Constraint Graph Satisfiability node types AND, OR, MAJ, and RED-BLUE conversion, as well as their allowed edge orientations. Thereby the total in-weight has to be at least 2 (except for RED-BLUE conversion), achievable by a single blue in-edge (weight 2) or two red in-edges (two times weight 1). The crossover gadget internally uses AND, OR, MAJ, and RED-BLUE conversion, as depicted in Figure 5.

Usually we restrict edge weights to either 1, which we call ***red***, or 2, which we call ***blue***. A ***constraint graph*** G is a set of constraint graph nodes and edges, where each edge appears in exactly two nodes. A ***configuration*** of G is an assignment of directions to the edges such that, for each node (E, W, c), the total weight of incoming edges among E is at least c.

▶ **Problem 1** (Constraint Graph Satisfiability (CGS)). *Given a constraint graph G does there exists a legal configuration?*

▶ **Problem 2** (Counting Constraint Graph Satisfiability (#CGS)). *Given a constraint graph G, how many legal configurations of G exist?*

The definitions above are more general than the standard definitions [3]. In particular, they allow the notion of edge weight to be local to a vertex, instead of being specified at the graph level. If an edge can have different weights at each end, we call the problem ***arbitrary edge weights***, while if an edge is required to have the same weight at both ends, we call the problem ***matching edge weights***. Equivalently, we can think of arbitrary edge weights as equivalently allowing for a **red-blue *conversion*** gadget — an edge that is red on one end and blue on the other.

In the standard formulation of constraint graph satisfiability, three types of degree-3 vertices: AND, OR, and MAJ. Figure 1 gives an overview of these types and all allowed edge orientations. An **and** vertex has two red edges and one blue edge, so it is satisfied only when both red edges are in-edges or the blue edge is an in-edge. Therefore, it mimics a boolean AND, where if both red edges are inward, then the vertex is "true" (the blue edge can be outward) and otherwise, it is "false" (the blue edge must be inward). An **or** vertex has three blue edges, so it is satisfied as along at least one edge is inward, similar to a boolean OR. A **maj** vertex has only red edges, so it is only satisfied if at least two (the majority) of its three edges are inward.

CGS is also a special case of Graph Orientation [4] where the valid configurations are shown in Figure 1.

In many cases, we would like the constraint graph to be planar. A useful tool for this is a CROSSING vertex allows us to build non-planar graph out of planar graphs. If two edges cross, we put a CROSSING vertex at their intersection point. This vertex mimics the standard crossover gadget used in graph reductions.

3 Generalized #SAT Dichotomy

In this section we outline our size-2 dichotomy results. We begin with some definitions that help describe the easy cases. Throughout this section, we let $f\colon \{0,1\}^2 \to \mathbb{Z}_{\geq 0}$ be a 2-variable clause type.

▶ **Definition 2.** *f is **factorable** if there exist functions g and h such that $f(x,y) = g(x)h(y)$ for all $(x,y) \in \{0,1\}^2$.*

▶ **Definition 3.** *f is **2-color** if either $f(0,0) = f(1,1) = 0$ or $f(0,1) = f(1,0) = 0$.*

Our main theorem of this section is the following dichotomy result.

▶ **Theorem 4** (Dichotomy Theorem)**.** *Generalized #SAT with a single 2-variable clause type f is in FP if f is either factorable or 2-color. Otherwise, it is #P-complete.*

3.1 Easy Cases

First, we describe polynomial-time algorithms for the factorable and 2-color cases.

▶ **Lemma 5.** *If f is factorable, Generalized #SAT can be solved in polynomial time.*

Proof. Let $f(x,y) = g(x)h(y)$. The product $\prod_{\phi \in C} f(x_\phi)$ can be expanded as a product of gs and hs, $\prod_{\phi \in C} f(x_\phi) = \prod_{i=1}^{n} g(x_i)^{a_i} g(\bar{x}_i)^{b_i} h(x_i)^{c_i} h(\bar{x}_i)^{d_i}$, for some exponents a_i, b_i, c_i, d_i. The sum of this expression over all $(x_1, \ldots, x_n) \in \{0,1\}^n$ is equal to $\prod_{i=1}^{n} \sum_{x_i \in \{0,1\}} g(x_i)^{a_i} g(\bar{x}_i)^{b_i} h(x_i)^{c_i} h(\bar{x}_i)^{d_i}$, which can be evaluated in polynomial time. ◀

▶ **Lemma 6.** *If f is 2-color, Generalized #SAT can be solved in polynomial time.*

Proof. Each clause $\phi = f(x,y) \in C$ forces x and y to be either equal to each or not equal to each other. Consider a graph on n nodes one for each variable, and add an edge between x and y for all $\phi \in C$. Within each connected component of this graph, fixing an assignment to any one variable forces all the others. There are at most 2 ways to satisfy each connected component, and the answer is the product of the answers for each component independently. ◀

3.2 Hardness

Next, we prove that all remaining cases are #P-complete, we will show that all hard clause types reduce to one of the two following cases.

▶ **Definition 7** (2SAT-like)**.** *f is **2SAT-like** if $f(0,0) = 0$, $f(0,1) = f(1,0) = a > 0$, $f(1,1) = b > 0$.*

▶ **Definition 8** (Max-Cut-like)**.** *f is **Max-Cut-like** if $f(0,0) = f(1,1) = a > 0, f(0,1) = f(1,0) = b > 0, a \neq b$*

▶ **Lemma 9.** *If f is 2SAT-like, then Generalized #SAT is #P-complete.*

Proof. We reduce from #2SAT. For each clause $\varphi = x \vee y$ in the #2SAT instance (where x and y are literals), add a unique new variable z and three clauses $\phi_1, \phi_2, \phi_3 = f(x,y), f(\bar{x},z), f(\bar{y},z)$ to the generalized #SAT formula. The number of ways to satisfy the clause is 0 if $x \vee y$ is false and $a^2 b$ otherwise:

- If $(x,y) = (0,0)$, $f(0,0) = 0$, so $f(x,y)f(\bar{x},z)f(\bar{y},z) = 0$.

- If $(x, y) = (0, 1)$ or $(1, 0)$, then $\sum_{z\in\{0,1\}} f(x, y)f(\bar{x}, z)f(\bar{y}, z) = f(x, y)\sum_{z\in\{0,1\}} f(0, z)f(1, z) = a^2b$.
- If $(x, y) = (1, 1)$, then $\sum_{z\in\{0,1\}} f(x, y)f(\bar{x}, z)f(\bar{y}, z) = f(x, y)\sum_{z\in\{0,1\}} f(0, z)f(0, z) = a^2b$.

Therefore, this reduction is $(a^2b)^m$-monious, where there are m clauses in the #2SAT instance, so Generalized #SAT is #P-complete. ◀

▶ **Lemma 10** (Max-Cut-like). *If f is Max-Cut-like, Then Generalized #SAT is #P-complete.*

Proof. We reduce from #Max Cut. Let the input of a #Max Cut instance be a graph $G = (V, E)$, and calculate the number $M := 1 + \lceil \log_{\max(a/b,b/a)}(2^{|V|})\rceil \in O(|V|)$. Associate a boolean variable to each vertex in V. For each edge $(x, y) \in E$, add M clauses of the form $f(x, y)f(\bar{x}, \bar{y})$. The answer to this Generalized #SAT instance is

$$N := \sum_{S\sqcup T=V} b^{M\mathrm{cut}(S,T)}a^{M(|E|-\mathrm{cut}(S,T))} = \sum_{c=0}^{|E|} k_c b^{Mc}a^{M(|E|-c)},$$

where $\mathrm{cut}(S, T) := \#\{(u, v) \in E \mid u \in S, v \in T\}$ and $k_c := \#\{(S, T) \mid S \sqcup T = V, \mathrm{cut}(S, T) = c\}$. Note that $0 \le k_c \le 2^{|V|}$ and the ratios of adjacent coefficients $\frac{b^{Mc}a^{M(|E|-c)}}{b^{M(c+1)}a^{M(|E|-(c+1))}} = \left(\frac{a}{b}\right)^M$ differ by more than $2^{|V|}$. Therefore, it is possible to exactly extract all the numbers $\{k_c\}_{0\le c\le|E|}$ from N, and the answer to #Max Cut is $k_{\max(c:\, k_c>0)}$. ◀

3.3 Main Dichotomy Result

We now present the complete proof of our size-2 dichotomy, based on our four clause types each defined in relevant theorems, factorable, 2-colorable for easy cases, and #2SAT-like and Max-Cut-like for the hard cases.

Proof of Theorem 4. If f is factorable or 2-colorable, Generalized #SAT is in FP by Lemma 5 or Lemma 6. Suppose f is not one of these cases. Then at most one of the values $f(x, y)$ for $(x, y) \in \{0, 1\}^2$ can be 0.

If one of these values is 0, we reduce from Lemma 9. By negating one or both arguments of f, without loss of generality we let $f(0, 0) = 0$ and $f(0, 1), f(1, 0), f(1, 1) > 0$. Replace each 2SAT-like clause $f_2(x, y)$ with two clauses, $f(x, y)f(y, x)$.

If none of these values is 0, we reduce from Lemma 10. Replace each Max-Cut-like clause with two clauses, $f(x, y)f(\bar{x}, \bar{y})$. ◀

4 Counting Constraint Graph Configurations (#CGS)

Having established the generalized #SAT dichotomy, we now use these results as a tool to prove hardness of #CGS. Specifically, we establish the many results of Table 1, which we hope will enable further use of CGS for many puzzles and games. First, we establish easy cases with single vertex types and also prove that this problem is already #P-complete on graphs using AND nodes with general edge weights. Then, in Section 4.2 we prove ASP-hardness when allowed all three vertex types — AND, OR, and MAJ— and general edge weights. Finally we restrict the set of allowed vertex types, and present bounds and complexity characterizations when counting solutions is hard.

Figure 2 Variable gadget (left) showing an equal number of positive and negative variable occurrences. Using RED-BLUE conversion vertex types, we can construct an odd number of out-going edges (right). Note that at most one red-blue conversion is needed (in case of different parity of x and $\neg x$ occurrences), as x's and $\neg x$'s can be linked by a red edge (depicted above).

4.1 Constraint Graphs with a Single Vertex Type

We show that #CGS on graphs with just AND vertices in #P-complete with general edge weights. But if we enforce matching edge weights, then #CGS is in FP. We also show that for just MAJ vertices, #CGS is in FP. (Because MAJ vertices have just one edge type, the notion of matching or general edge weights is irrelevant.)

To show that AND vertices with general edge weights is #P-complete, we reduce from #SAT. In the following, we design a (parsimonious) variable gadget for every variable x, as depicted in Figure 2. The gadget uses $O(k)$ nodes where k is the number of times x appears, thus each vertex functions as a literal, with the edge orientation propagating the value. Every variable occurrence is thereby connected to an AND node such that if the edge is directed towards x ($\neg x$) the literal x ($\neg x$) evaluates to true. By construction, the variable gadget has only two legal edge orientations. One of which is shown in Figure 2 (x is true), the other configuration is its complement, where all edges are inverted (x is false).

This gadget and Section 3 are sufficient ingredients to establish the following tight result.

▶ **Theorem 11** (#P-Hardness). *Counting with just AND is #P-complete if we do not enforce matching edge weights.*

Proof. We reduce from 2SAT-like Generalized #SAT with clause type $f(x, y) = x + y$ (see Lemma 9) and construct a #CGS instance as follows. We reuse the variable gadget of Figure 2. However, the clause gadget for a binary clause $c = (\ell \vee \ell')$ (ℓ, ℓ' are two distinct literals) is simply a red edge connecting literal ℓ to literal ℓ'. Indeed, by construction, the clause gadget requires that the edge is directed either to ℓ (i.e. ℓ is true) or to ℓ' (i.e. ℓ' is true). Consequently, if both ℓ and ℓ' are true, clause c is considered with weight 2, as we can always direct the edge from ℓ to ℓ' or from ℓ' to ℓ. This is as required by Equation (1), which coincides with the number of CGS solutions of the constructed instance and establishes the result. ◀

This result immediately works for planar graphs as well, as in Lemma 9 we can alternatively reduce from planar (monotone) #2SAT, which is #P-complete [11]. This result is tight, as the counting problem #CGS is easy for just AND nodes if we enforce matching edge weights.

▶ **Theorem 12.** *#CGS with just AND with matching edge weights is in FP.*

Proof. Note that since matching edge weights is enforced, each AND vertex must pair off with exactly one other AND vertex. They connect to each other via their blue input and form a "super-vertex" with four red inputs/outputs (Figure 3). For any such pair of AND vertices, A and B, clearly, their shared blue edge can only be directed into one of them. Without loss of generality, assume it is directed into A. Then, for B to be satisfied, the red edges

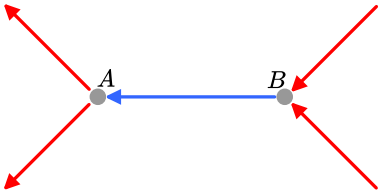


Figure 3 AND vertices must pair along blue edges if matching edge weights are enforced.

of B must point into itself. Note that this implies each degree 4 super-vertex must have in-degree at least 2. Our assumption of the blue edge pointing into A fixes the direction of the red edges of B. It also forces that the red edges of A must point outwards. Assume one of A's red edges points inwards; this super vertex has in-degree 3. This implies there must be some other corresponding super vertex with an in-degree of exactly 1, which is unsatisfied. Therefore, for any pair of AND vertices, if we set the direction of their shared blue edge, the direction of their red edges is forced, and those edges then force the direction of the edges of other pairs of ANDs, and so forth. Therefore, a connected graph of AND vertices with matching edge weights has either no solutions or two solutions (we can reverse the edges of one solution to get a second). Hence, for a graph of AND vertices with matching edge weights with k connected components, the number of solutions is either 0 or 2^k. ◀

Additionally, #CGS on a graph of just MAJ vertices is also in P, by a relatively simple proof that such a graph is never satisfied.

▶ **Theorem 13.** *#CGS with just MAJ is in FP.*

Proof. MAJ vertices require an in-degree of 2 to be satisfied. However, a graph of only MAJ vertices is 3-regular. Therefore, the average in-degree is exactly 1.5 so at least one vertex must have in-degree less than two. Therefore, there are always 0 solutions. ◀

For the case of only ORs, an equivalent version of #3SAT was shown to be #P-complete [1]. We can view each OR vertex as a 3CNF clause and each edge as a variable that appears once with each sign, i.e., for each variable x, there exists exactly one x literal and exactly one $\overline{x}$ literal in the formula.[3]

▶ **Theorem 14.** [1] *#CGS with just OR is #P-complete even when restricted to planar graphs.*

4.2 Parsimonious Reduction from #1-in-3SAT to #CGS

First, we discuss a parsimonious reduction that uses AND, OR, MAJ, and RED-BLUE conversion vertex types. This reduction then yields ASP-hardness for CGS, which makes this formalism a perfect tool to prove that for puzzles and games, finding a second solution is still hard. To this end, we directly reuse the variable gadget of Figure 2. Figure 4 (left) depicts the clause gadget for the clause $x \vee \neg y \vee z$, which works via the three 1-in-3SAT cases. Both gadgets are then used in the reduction below.

▶ **Theorem 15** (ASP-hardness). *The another-solution problem for CGS is ASP-hard, even if restricted to the vertex types AND, OR, and MAJ.*

[3] This version of #3SAT is named #Pl-Rtw-Opp-3CNF.

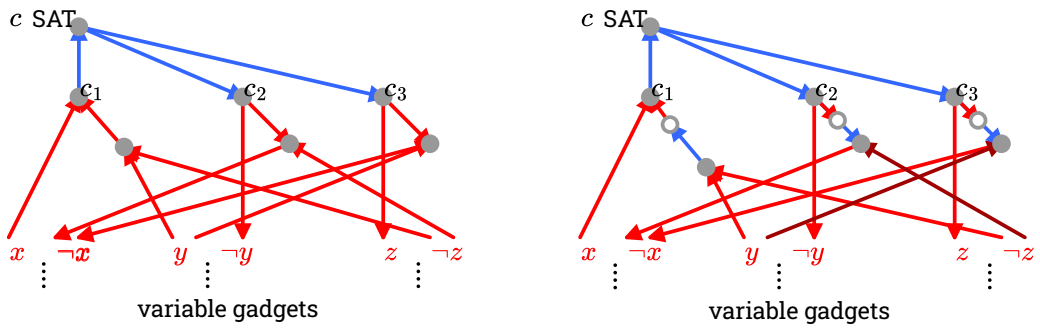


Figure 4 Clause gadget (left), where for a 1-in-3SAT clause of the form $c = x \vee \neg y \vee z$ we parsimoniously preserve solutions by expressing three cases (terms) $c_1 = (x \wedge (y \wedge \neg z))$, $c_2 = (\neg y \wedge (\neg x \wedge \neg z))$, $c_3 = (z \wedge (\neg x \wedge y))$, where the first literal of each c_i is the one from c being true (and the two remaining literals occur negated in c). Indeed these three cases allow us to preserve a bijective relationship between 1-in-3SAT solutions and satisfying edge orientations. However, it is crucial that the brackets are precisely as above, as this pins down edge orientations for MAJ vertex types connected to c_i. Roughly, in c_i the literals of c that are supposed to be false are connected by a MAJ vertex. If we replace the MAJ vertices by AND vertices (right), the orientation of the dark red edges is free. While this does not preserve parsimony, for m clauses the reduction is still 4^m-monious.

Proof. The reduction consisting of variables gadgets (Figure 2) and clause gadgets (Figure 4) is correct. By construction, the variable gadget for a variable x in Figure 2, which is attached at the bottom, prevents that there are both outgoing x and $\neg x$ edges (simultaneously). The clause gadget for a clause c shown in Figure 4 ensures that precisely one of the cases c_1, c_2, c_3 holds. Indeed, each pairwise combination of the three cases is by construction contradicting. However, in a solution every edge direction is pinned down as either the case c_i holds (outgoing blue edge, which requires all outgoing red edges towards c_i), or there is precisely one outgoing edge of the MAJ vertex. Indeed, by construction the pairwise intersections of literals in c_i, c_j for $i \neq j$ are of size 1. Hence, if c_j does not hold then exactly one of the two literals of c_j that are negated in c are true. This literal therefore is a predecessor of the MAJ vertex attached to c_j.

Since this reduction is parsimonious and 1-in-3SAT is ASP-hard [15], we conclude the result. ◀

Observe from the variable gadget in Figure 2 that if for every variable x, the number of occurrences of x and $\neg x$ are identical, RED-BLUE conversion vertex types are not needed. However, if we do not use MAJ vertex types, we need RED-BLUE conversion in the clause gadget, as depicted in Figure 4 (right). Further, this leaves 2 free edges per clause, resulting in a reduction that is $2^{2m} = 4^m$-monious.

Crossover Gadget for Planarity. Theorem 15 immediately works for planar graphs, since we can construct a parsimonious crossover gadget as depicted in Figure 5 and eliminate all crossings with a gadget.

▶ **Corollary 16** (ASP-hardness for Planar CGS). *The another-solution problem for CGS is ASP-hard, even if restricted to planar graphs over the vertex types* AND, OR, *and* MAJ.

Proof. Figure 5 depicts a planar red-red crossing gadget. Indeed we only need to eliminate red-red crossings as there is no face completely built out of blue edges (see Figures 2 and 4). Observe further that the gadget in Figure 5 is parsimonious, i.e., there is no free edge,

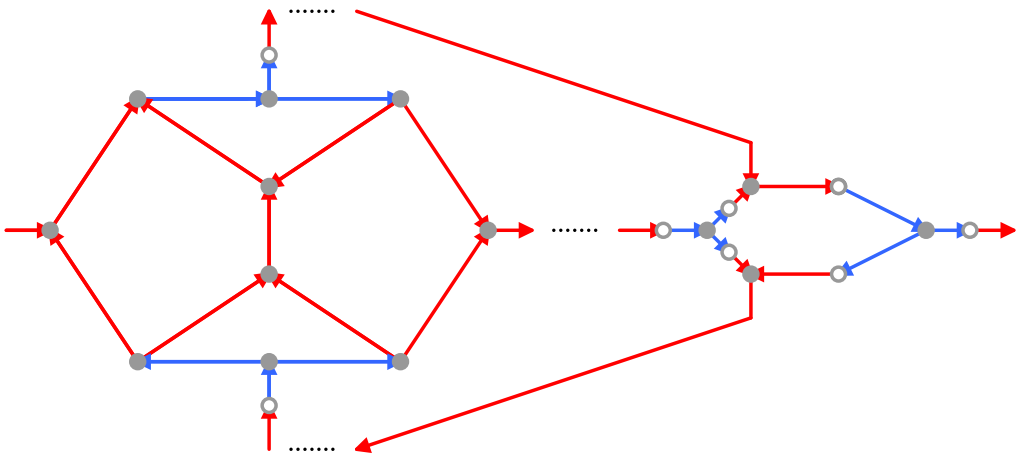


Figure 5 Parsimonious red-red crossover gadget that consists of a leaky main part (left) that is leaky in the sense that it still allows the case where both north/south vertices are directed inward and east/west are directed outward. In order to fix this, one can add a degree-4-vertex type simulation gadget (right), as shown. This gadget simulates a degree-4-vertex requiring a total in-flow of weight at least 2 if there shall be out-flow (as depicted).

assuming the original north/south and east/west edges are fixed as well. Therefore the result follows from Theorem 15. ◀

4.3 Constraint Graphs with Two Vertex Types

We now discuss what happens when our graph consists of exactly two vertex types. First, we establish easiness for the decision problems. Then, we consider counting.

4.3.1 Decision Easiness

For the decision problem CGS we obtain the following easiness results.

▶ **Theorem 17.** *CGS with* OR *and* MAJ *is in P.*

Proof. We may reduce from CGS over OR and MAJ vertices to the Max Flow problem, which is known to be in P. Each OR vertex will be replaced by sink of weight 1 and each MAJ will be replaced by a sink of weight 2. At the center of each edge add a source of weight 1.

A flow which satisfies all the sinks can be used to assign orientations on the edges of the constraint graph. The direction taken by edges out of the source is in the direction of the edge in the CGS solution. Since each sink has a weight equal to the number of incoming edges needed by the constraint node each node will be satisfied. A set of edge orientation which satisfy each edge can be used to assign the flow of each edge in the same way. ◀

▶ **Theorem 18.** *CGS for* AND *and* MAJ *is in P.*

Proof. We reduce to 2-SAT, which is in P [5]. Note that an AND vertex with inputs labeled b, r_1, r_2, where b is its blue input, and r_1, r_2 are its red inputs, can be represented by the boolean equation $(b) \vee (r_1 \wedge r_2) = (r_1 \vee b) \wedge (r_2 \vee b)$, where a variable is true if its corresponding edge is directed into the vertex. Additionally, a MAJ vertex with inputs a, b, c can be represented by $(a \wedge b) \vee (a \wedge c) \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \wedge (b \vee c)$. So each vertex can be represented with a set of 2-SAT clauses. Note that each variable represents an edge

that connects two vertices. Therefore, each variable will only appear in sets of clauses corresponding to the two vertices it is incident to. We negate a set of these literals so that only one may be true to simulate the edge only pointing in one direction. Therefore we can write any constraint graph of AND and MAJ vertices as a 2-SAT formula. ◀

4.3.2 #CGS Hardness

In this section we show that, if we have exactly two vertex types (AND/OR, AND/MAJ, or OR/MAJ) and do not enforce matching edge weights, then #CGS is hard.

▶ **Theorem 19** (#P-Hardness for Two Vertex Types). *#CGS is #P-hard if we do not enforce matching edge weights and are restricted to AND/OR vertices, AND/MAJ vertices, or OR/MAJ vertices, with at least one vertex of each of the two types.*

Note that #P-hardness for the cases including AND vertices follows already from Theorem 11. Consequently, it suffices to establish the following lemma.

▶ **Lemma 20.** *#CGS with OR and MAJ is #P-complete, even when there exists a connected component which contains both vertex types.*

Proof. We reduce from counting the number of perfect matchings in a 3-regular bipartite graph $(V = \{O \bigcup M\}, E)$, which has been shown to be #P-hard [2]. We will replace one partition of vertices O with OR vertices and the other M with MAJ.

The set of edges in the matching correspond to edges directed from a node in M to a node in O. Each O node requires edge pointed in to satisfy its inflow. Each M node can only be used in a single matching as it requires 2 edges pointed in. ◀

4.3.3 #CGS Easiness

However, with matching edge weights #CGS is in FP if we have two vertex types, one of which must be MAJ. This further strengthens Theorems 17 and 18.

▶ **Theorem 21** (Counting is easy MAJ). *#CGS can be solved in polynomial time if we enforce matching edge weights and are restricted to two vertex types, one of which is MAJ (either MAJ/OR or MAJ/AND).*

Each pair of vertices has a separate proof, hence we prove this theorem via the following lemmas.

▶ **Lemma 22.** *#CGS can be done in polynomial time if we enforce matching edge weights and are restricted to only MAJ and OR vertices, and there is a nonzero number of MAJ vertices.*

Proof. Since we enforce matching edge weights, OR and MAJ vertices cannot connect to each other as OR vertices only take blue inputs, and MAJ vertices only take red inputs. Therefore, any constraint graph with OR and MAJ will have multiple components, ones made up of only OR vertices and ones made up of only MAJ vertices. By Theorem 13, we know each MAJ component can never be satisfied; hence, the number of solutions to CGS with only MAJ and OR vertices with matching edge weights is always 0. ◀

Note that in Lemma 22 we require the number of MAJ vertices to be nonzero as otherwise the graph has only OR vertices, which we leave an open problem, as discussed in section 4.1.

▶ **Lemma 23.** *#CGS can be done in polynomial time if we enforce matching edge weights and are restricted to only MAJ and AND vertices.*

Proof. By enforcing matching edge weights, an argument similar to Theorem 12 applies, as MAJ vertices have no blue inputs; therefore, each AND vertex must pair with another AND through a blue edge. Note that the average in-degree in each of these AND pairs is ≥ 1.5. Further, each MAJ vertex requires in-degree > 2 to be satisfied. Therefore, if a constraint graph contains AND and MAJ vertices, the average in-degree must be > 1.5, or else it is not satisfied. But this is impossible; if the graph is 3-regular, the average in-degree is 1.5. Hence, if there is a nonzero number of MAJ vertices, the number of solutions is 0. Otherwise Theorem 12 applies and else the number of solutions is either again 0 or 2^k where k is the number of components of the graph. ◀

5 Conclusion and Future Work

In this work we presented a novel generalized #SAT framework as well as a dichotomy for two variables. These results then serve as the basis for novel insights into the counting complexity of graph orientation problems (constraint graph satisfiability), where we discuss an almost-complete classification (see also Table 1). We expect that counting solutions to constraint graph satisfiability (#CGS) is an interesting source to support the development and characterization of challenging puzzles, riddles, and games. Indeed, given our insights we expect many further insights into counting solutions and solving another-solution problems.

Based on our dichotomy result for 2-variable clauses, we conjecture that Generalized #SAT is in FP if f factors into a product of single-variable and 2-color functions, or it is a multiple of an affine function, and is #P-complete in all other cases. Our understanding is that for any given f, it is probably not difficult to prove this via taking "slices" with fewer variables, but we lack a systematic method to prove the dichotomy for all f that does not rely on ad-hoc casework.

References

1 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Holographic reduction, interpolation and hardness. *Computational Complexity*, 21:573–604, 2012.

2 Paul Dagum and Michael Luby. Approximating the permanent of graphs with large factors. *Theoretical Computer Science*, 102(2):283–305, 1992.

3 Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. CRC Press, 2009.

4 Takashi Horiyama, Takehiro Ito, Keita Nakatsuka, Akira Suzuki, and Ryuhei Uehara. Complexity of tiling a polygon with trominoes or bars. *Discrete & Computational Geometry*, 58(3):686–704, October 2017. doi:10.1007/s00454-017-9884-9.

5 M. R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2):15–20, 1967. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/malq.19670130104, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/malq.19670130104, doi:10.1002/malq.19670130104.

6 David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982. doi:10.1137/0211025.

7 MIT Hardness Group, Josh Brunner, Lily Chung, Erik D. Demaine, Della Hendrickson, and Andy Tockman. ASP-completeness of Hamiltonicity in grid graphs, with applications to loop puzzles. In *Proceedings of the 12th International Conference on Fun with Algorithms*, pages 30:1–30:20, 2024.

8 J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983. arXiv:https://doi.org/10.1137/0212053, doi:10.1137/0212053.

9 Takahiro Seta. The complexities of puzzles, Cross Sum, and their Another Solution Problems (ASP). Senior thesis, University of Tokyo, 2002. URL: https://web.archive.org/web/20221007013910/www-imai.is.s.u-tokyo.ac.jp/~seta/paper/senior_thesis/seniorthesis.pdf.

10 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi:10.1137/0220053.

11 Salil P. Vadhan. The Complexity of Counting in Sparse, Regular, and Planar Graphs. *SIAM J. Comput.*, 31(2):398–427, 2001. doi:10.1137/S0097539797321602.

12 Leslie G. Valiant. A polynomial reduction from satisfiability to Hamiltonian circuits that preserves the number of solutions. Manuscript, 1974.

13 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

14 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979. doi:10.1137/0208032.

15 Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 86(5):1052–1060, 2003.