Chapter 2

Background

In this chapter, preliminary knowledge for understanding our graph is presented. Beginning with graph theory and complexity theory, this will provide basic a understanding of its form, and the problem we are trying to solve. Additionally, we will highlight key concepts from related works, which will be used in constructing our graph. Furthermore, we will briefly describe the history of related tools that we require in development.

2.1 Graph Theory

2.1.1 Graphs

A *graph* is a pair $G = (V,E)$ of sets such that $E \subseteq [V]^2$. Hence, the elements of E are 2-element subsets of V. We shall assume $V \cap E = \emptyset$. The elements of V are the *vertices* (or *nodes*, or *points*) of the graph G, the elements of E are its *edges* (or *lines*).

2.1.2 Morphisms

Given graphs $G = (V,E)$ and $G' = (V',E')$. If there exists a bijection between vertex sets $f : V \to V'$ with $(x,y) \in E \iff (f(x),f(y)) \in E'$ for all $x,y \in V$, then we call G and G' *isomorphic*, denoted $G \simeq G'$. Such a map f is called an *isomorphism*; if $G = G'$, it is called an *automorphism*. The set of all automorphisms forms a group which we call an *automorphism group* $\mathrm{Aut}(G)$. If $\mathrm{Aut}(G)$ is trivial, that is, it contains only the identity mapping, then we say that G is *rigid*.

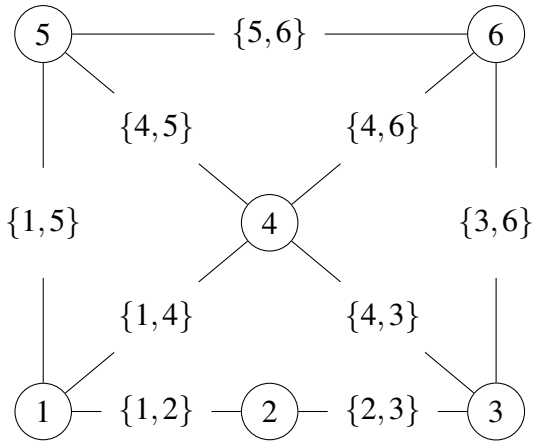

Fig. 2.1 An example graph
The graph on $V = \{1,2,3,4,5,6\}$ with edge set
$E = \{\{1,2\},\{2,3\},\{1,4\},\{4,3\},\{1,5\},\{4,5\},\{4,6\},\{3,6\},\{5,6\}\}$

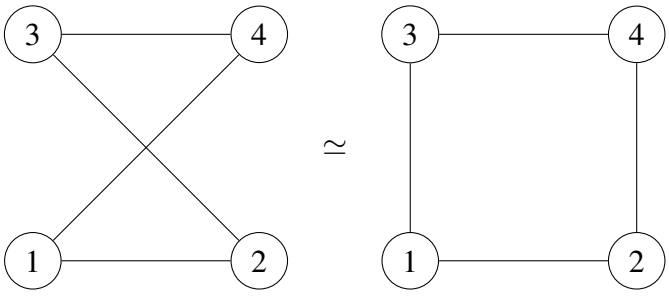

Fig. 2.2 Isomorphic graphs

2.2 Complexity Theory

A *complexity class* is a set of computational problems that can be decided within some bound specified by their performance, that is, whether input to a Turing Machine completes or runs forever. *P* is the complexity class that contains all decision problems that can be solved by a deterministic Turing Machine in polynomial time. The complexity class *NP* is the set of all problems which can be validated to be true in polynomial time, or solvable in polynomial time by a non-deterministic Turing Machine. Problems that are *NP-Hard* are at least as hard as the hardest problems in NP. *NP-Complete* problems are those which contain the hardest problems in NP.

A *quasi-polynomial* time algorithm is one that runs slower than polynomial time, yet not slow as to be exponential time. The worst case running time of a quasipolynomial time algorithm is $2^{O((\log n)^c)}$ for some fixed $c > 0$.

The graph isomorphism problem (GI) is the decision problem determining whether two graphs are isomorphic. GI has been proven to have a quasipolynomial time algorithm solution by Babai [6]. The boolean satisfiability problem (SAT) is the decision problem in determining whether there exists a satisfying argument to a boolean formula, which is NP-Complete. The XOR-SAT problem is similar to SAT, but we are limited to using $\oplus$, the exclusive or, when constructing clauses in logical statements. XOR-SAT when viewed as system of linear equations (mod 2) can be solved in polynomial-time by using Gaussian elimination [9].

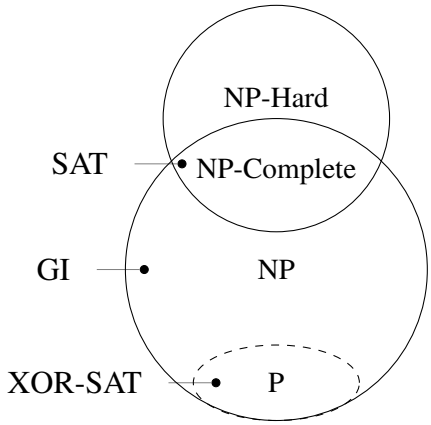

Fig. 2.3 Fundamental complexity classes and computational problems

2.3 XOR-Formulas on the Threshold of Satisfiability

XOR formulas are logical statements that consist of the exclusive $\oplus$ and $\wedge$ connectives; for example $r = (a \oplus b \oplus c) \wedge d$ is an XOR-Formula consisting of two *clauses*, namely $a \oplus b \oplus c$ and d. Each variable has the value of either False or True. If the statement equates to True, that is, it can hold true with an assignment of False and True values, then we say it is *satisfiable*, otherwise way say it is *unsatisfiable*.

XOR-Formulas are also represented in algebraic terms. Using the previous example, $r = a + b + c(mod 2)$ and $s = d(mod 2)$ is its equivalent statement, also known its *algebraic normal form*. Here, values are assigned either 0 or 1, representing False and True respectively. Therefore, we can interchange descriptions when we mention XOR-Formula: when we say logical statement, we also mean algebraic equation mod 2; when we say formula, we also mean set of equations.

Let ϕ denote an XOR-Formula. If ϕ has at most k literals in every clause, then we say it is a k-XOR-Formula; for example, $r = (a \oplus b \oplus c)$ is a 3-XOR-Formula. We let n denote the number of variables in a formula ϕ and m the total number of clauses. A k-XORSAT is the problem of determining the truth of an k-XOR-Formula. Dubois and Mandler proved that if ϕ has an equal number of variable to clauses $n = m$ and $k \geq 3$, then was say it is on the *threshold of satisfiability* [10]. That is to say, if $n < m$, then ϕ will always be $unsatisfiable$, and if $n >> m$ then ϕ is likely to be satisfiable. Therefore, $n = m$ is a sharp bound for satisfiability for any given instance of k-XORSAT, where $k \geq 3$. This is confirmed by Pittel and Sorkin [11].

2.3.1 Multipedes

Multipedes are a form of hypergraph, where there exists hyperedges. Such hyperedges have edges which connect to any number of nodes, that is, they can connect 3 or more elements $h = (x, y, z)$. Normally, we would take (x, y) to represent an edge between nodes, but now we allow z to observe that there exists an edge between x and y. We will not describe the mathematical definition of multipedes, as this is covered extensively, and can not be summarised quickly. We can say that they are proven to be rigid [12].

2.4 Construction

Here we will refer Professor Dawar's notes provided in the Appendix to describe our construction and non-standard terminology. A 3-XOR-Formula ϕ is *homogenous* if the all equations are equal to 0. A homogeneous system is always satisfied by the all-zero solution, whereby all literals equal 0. If the all-zero is the *only* solution to a 3-XOR-Formula, then we say it is *uniquely satisfiable* (See Fig. 2.4). Our definition of *k-local consistency* is more lengthy, and requires us to describe it in the form of a hypothetical game [].

Our construction will be created at random. Recall that n and m represent the number of variables and clauses respectively in a 3-XOR-Formula. From these values, we generate ϕ to be uniquely satisfiable and k-locally consistent. Once we know that it has both these properties, we generate our preliminary graph G_A, which we will check for automorphisms. If G_A has no non-trivial automorphisms, then we construct our final graph G_B. These graphs as described as follows.

G_A has nodes for each literal used and clause. Therefore, if we have n literals and m clause, G_A has $n+m$ nodes. The edges are connections between nodes, such that if a clause contains a literal, then there exists an edge. Every clause as a node, has exactly 3 edges, since every clause has 3 literals. (See Fig. 2.5)

G_B is our final construction, once we apply the multipede property to G_A. For every literal x, we add two nodes x_0 and x_1. For every clause C, we insert four clauses C_1, C_2, C_3, C_4. Since C has 3 literals, there is a total of 4 ways in which we build a homogeneous system. For example, take $0 = a+b+c$, the all-zero solution is possible $a=0, b=0, c=0$; furthermore, since our equation is *mod 2*, $a=0, b=1, c=1$ and $a=1, b=1, c=0$ and $a=1, b=0, c=1$ are also solutions. We will allocate each solution to a clause such that, if x is in the solution C and equals 0, then there is an edge between x_0 and C, and if x is in the solution C and equals 1, then there is an edge between x_1 and C. (See Fig. 2.6). G_B has a total of $2n+4m$ nodes.

$$
\begin{array}{r}
b+c+e=0 \\
b+c+f=0 \\
c+d+e=0 \\
a+b+d=0 \\
b+e+f=0 \\
b+c+d=0
\end{array}
\iff
\begin{array}{r}
(\neg b \oplus \neg c \oplus \neg e) \wedge \\
(\neg b \oplus \neg c \oplus \neg f) \wedge \\
(\neg c \oplus \neg d \oplus \neg e) \wedge \\
(\neg a \oplus \neg b \oplus \neg d) \wedge \\
(\neg b \oplus \neg e \oplus \neg f) \wedge \\
(\neg b \oplus \neg c \oplus \neg d)
\end{array}
\iff
\begin{array}{r}
False(b \oplus c \oplus e) \wedge \\
False(b \oplus c \oplus f) \wedge \\
False(c \oplus d \oplus e) \wedge \\
False(a \oplus b \oplus d) \wedge \\
False(b \oplus e \oplus f) \wedge \\
False(b \oplus c \oplus d)
\end{array}
$$

Fig. 2.4 Example 3-XOR-Formula ϕ
that is homogeneous and uniquely satisfiable.

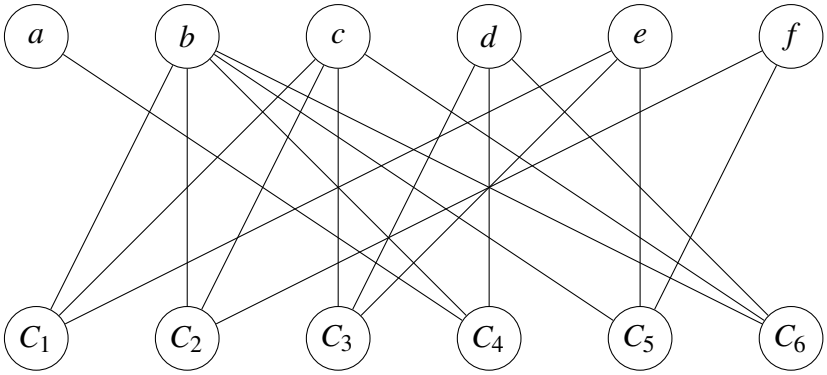

Fig. 2.5 Example preliminary graph G_A on ϕ

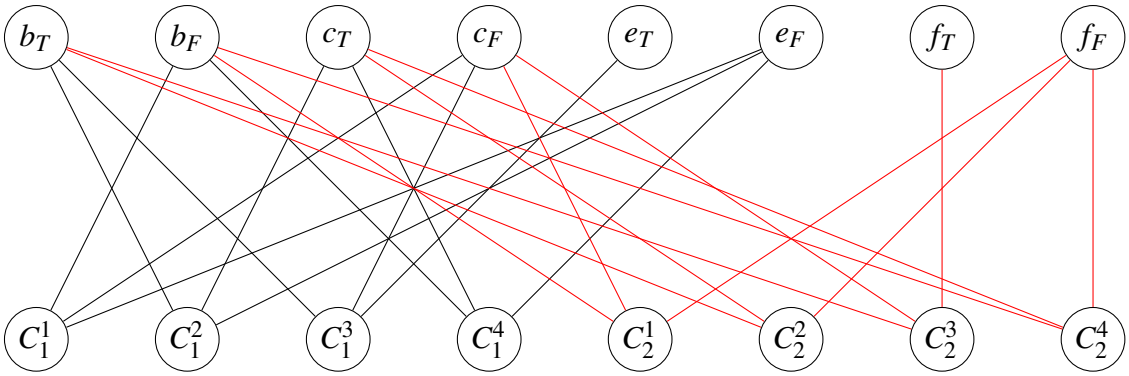

Fig. 2.6 Snippet of example final graph G_B on ϕ.
Only C_1 and C_2 is shown

2.5 Solvers

Computational problems have led scientists and mathematicians to develop efficient implementations in dealing with them. Our work will require two such programs: a SAT solver and a GI solver.

2.5.1 GI Solvers

The programs Nauty and Traces are identified as among the two fastest GI solvers for most cases of small graphs and large graphs respectively. Mauty was developed in the 1980s by B. McKay [1], which was later revised in 2013 as Traces [2] to produce greater efficiency gains. Moreover, it is known that these programs work off two key components utilised in a backtracking search: vertex colour refinement and factoring of automorphisms. However, the exact complexity of these solvers is unknown. Therefore, benchmarks are used to gauge performance. Among the standard benchmark tests are Cai-Fürer-Immerman (CFI) graphs. Interestingly, CFI graphs are resistant to vertex colour refinement, but prone to factorisation due to high levels of automorphisms. Subsequently, this knowledge provides a basis of finding graphs that attack both of these methods; providing a new standard in benchmarking, and uncovering a complex family of graphs that refers Babai refers to.

2.5.2 SAT Solvers

The algorithms behind SAT solvers typically use Gaussian Elimination in solving a system of linear equations. In some cases, this feature can be disabled, and some implementations do not use it all. One such example is the cryptominisat program, a winner of many numerous SAT solving competitions. By toggling Gaussian Elimination on and off, and recording the times, one can make a good guess at whether ϕ is k-locally consistent. Those randomly generated systems which are solved significantly faster with Gaussian Elimination on versus off are ones of interest. The prediction is that systems with greater differences in execution speeds hints at k-local consistency of a system.

2.6 Previous Attempt

Recently, a study searched for the family of graphs Babai describes. E. Arrighi, a Cambridge intern guided by my project supervisor, found that multipedes are theoretically complex for Traces.[3][4]. Arrighi concluded that the smallest examples of suitable probabilistically generated multipedes, containing significantly high probability of non-trivial automorphisms, would be too large to generate, so rendering them too large for experimentation on Traces.

However, there is a method of producing similar smaller graphs which have the essential elements of multipedes. By combining formulas of XORSAT[5], that are on the threshold of satisfiability, and expander graphs. The resulting hybrid will have the same resistance as multipedes, yet smaller and more manageable for solvers.