A New Reduction from 3SAT to n-Partite Graphs

Daniel J Hulme
Dept. of Computer Science
University College London
Gower St, London
WC1E 6BT
T: +44 (0)20 7608 6840
F: +44 (0)20 7608 4064
d.hulme@cs.ucl.ac.uk

Robin Hirsch
Dept. of Computer Science
University College London
Gower St, London
WC1E 6BT
T: +44 (0)20 7679 1379
F: +44 (0)20 7387 1397
r.hirsch@cs.ucl.ac.uk

Bernard F Buxton
Dept. of Computer Science
University College London
Gower St, London
WC1E 6BT
T: +44 (0)20 7679 7294
F: +44 (0)20 7387 1397
b.buxton@cs.ucl.ac.uk

R.Beau Lotto
Institute of Ophthalmology
University College London
11-43 Bath St, London
EC1V 9EL
T: +44 (0)20 7608 4052
F: +44 (0)20 7608 4064
lotto@ucl.ac.uk

***Abstract*— The Constraint Satisfaction Problem (CSP) is one of the most prominent problems in artificial intelligence, logic, theoretical computer science, engineering and many other areas in science and industry. One instance of a CSP, the satisfiability problem in propositional logic (SAT), has become increasingly popular and has illuminated important insights into our understanding of the fundamentals of computation.**

Though the concept of representing propositional formulae as n-partite graphs is certainly not novel, in this paper we introduce a new polynomial reduction from 3SAT to G_7^n graphs and demonstrate that this framework has advantages over the standard representation. More specifically, after presenting the reduction we show that many hard 3SAT instances represented in this framework can be solved using a basic arc-consistency algorithm, and finally we discuss the potential advantages and implications of using such a representation.

I. Introduction

The Constraint Satisfaction Problem (CSP) describes a general framework for problems in which values must be assigned to a set of variables subject to specific constraints [1], [2], and is one of the most prominent problems in artificial intelligence (AI), logic, theoretical computer science, engineering and many other areas in science and industry.

One instance of a CSP, the satisfiability problem in propositional logic (SAT), has become increasingly popular and has lead to important insights into the nature of *satisfiability* and dramatic improvements in CSP algorithms, which can solve hard instances with thousands of variables [3] as well as many restricted instances in polynomial-time [4].

Fundamental to the study of CSP, the P versus NP problem, formulated independently by Stephen Cook [5] and Leonid Levin [6], has been one of the most important scientific questions posed to date. Indeed, over the past several decades researchers have been trying to determine whether or not there is a polynomial solution to any of the problems that have been shown (using polynomial reduction) to be NP-complete [7], many of which are described in Computers and Intractability: A Guide to the Theory of NP-completeness [8].

In this paper we introduce a new framework to represent SAT problems and demonstrate that this framework has advantages over the standard representation. More specifically, after introducing some elementary concepts in complexity and graph theory we present a new polynomial reduction from 3SAT to G_7. We demonstrate that a basic polynomial-time algorithm can solve a number of hard SAT benchmark instances represented using this framework, yet fails when applied to the standard G_3^n graph representation. Finally we briefly discuss the potential advantages, practical benefits and possible implications of using such a framework.

II. Background

A. Satisfiability and the Conjunctive Normal Form

The satisfiability problem in conjunctive normal form (CNF) consists of the conjunction ($\wedge$ representing the Boolean *and* connective) of a number of *clauses*, where a clause is a disjunction ($\vee$ representing the Boolean *or* connective) of a number of propositions or their negations.

If x_i represents propositions that can assume only the values *True* or *False*, then an example formula in CNF would be

$$(x_0 \vee x_2 \vee \bar{x}_3) \wedge (x_3) \wedge (x_1 \vee \bar{x}_2) \tag{1}$$

where $\bar{x}_i$ is the negation of x_i.

Given a set of clauses $C_0, C_1, \ldots, C_{n-1}$ on the propositions $x_0, x_1, \ldots, x_{m-1}$, the satisfiability problem is to determine if the formula $F = \bigwedge_{j<n} C_j$ has an assignment of values to the propositions such that it evaluates to *True*.

B. 3SAT

One of the original problems shown by Cook [5] to be NP-complete, 3SAT is considered the 'mother' of all SAT problems. Instances of 3SAT are restricted to Boolean formulae in CNF with three literals per clause. For example, the formula

$$(x_0 \vee x_1 \vee x_2) \wedge (\bar{x}_0 \vee x_1 \vee \bar{x}_2) \wedge (x_0 \vee \bar{x}_1 \vee x_3) \wedge (\bar{x}_0 \vee \bar{x}_2 \vee \bar{x}_3) \tag{2}$$

is a 3CNF formula with four clauses and is a YES instance to 3SAT since the truth assignment θ satisfies the formula, where one of the nine satisfying assignments is $\theta(x_0) = \theta(x_1) = True$ and $\theta(x_2) = \theta(x_3) = False$.

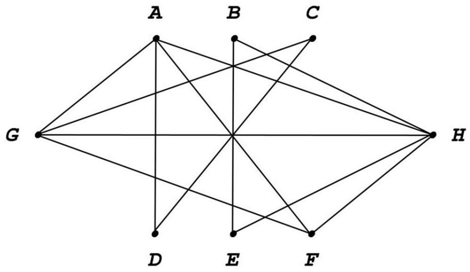

Fig. 1. An example of a 4-partite graph.

C. Satisfiability as a Constraint Satisfaction Problem

Representing a SAT instance as a CSP is specified by giving a formula in propositional logic (such as CNF) and asking whether there is an assignment to the set of propositions which makes the formula *True* [9].

EXAMPLE 1: For instance, finding a satisfying truth assignment for Formula 1 can be formulated as a CSP instance. Perhaps the most straightforward way is to construct the instance with a set of:

- Variables V = $\{x_0, x_1, x_2, x_3\}$.
- Values D = $\{0, 1\}$, corresponding to $False$ and $True$.
- Constraints C = $\{C_0, C_1, C_2\}$, where
  - $C_0 = \langle\langle x_0, x_2, x_3\rangle, D^3 \setminus \langle 0, 0, 1\rangle\rangle$
  - $C_1 = \langle\langle x_3\rangle, D^1 \setminus \langle 0\rangle\rangle$
  - $C_2 = \langle\langle x_1, x_2\rangle, D^2 \setminus \langle 0, 1\rangle\rangle$

D. Partite Graphs

A graph is n-partite when its vertices are partitioned into n independent subsets. More formally:

DEFINITION 1: A graph is n-partite iff:

1) Each of the n subsets is *independent* — i.e., no two vertices within the same set are adjacent (connect by an edge).
2) There is no partition of the vertices with fewer than n subsets where condition 1 holds.

Fig. 1 is an example of a 4-partite graph, which in this case has four independent sets of vertices: $\{A, B, C\}$, $\{D, E, F\}$, $\{G\}$, $\{H\}$.

E. Cliques

In an undirected graph, a clique is simply a set of vertices that are fully connected. For instance, in Fig. 1 there are several cliques of size three (3-clique), including $\{F, G, H\}, \{A, F, G\}$ and $\{B, E, H\}$, with only one 4-clique $\{A, F, G, H\}$.

F. The G_m Graph Problem

A G^n_m graph is n-partite, and each partition contains exactly m vertices. An instance of G_m is a G^n_m graph (for some n), and is a *YES* instance if it contains an n-clique and a NO instance otherwise.

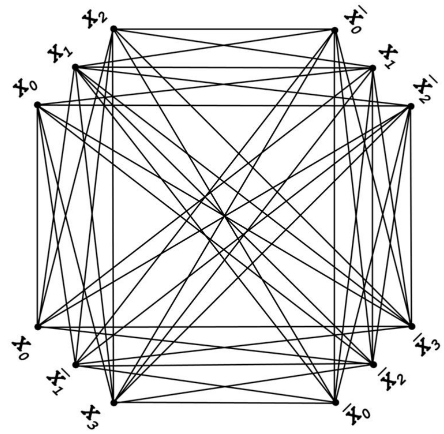

Fig. 2. Formula 2 reduced to a G^4_3 graph.

G. Polynomial Reduction

The method of showing that a problem is NP-complete by polynomial reduction is one of the most elegant and productive in computational complexity [10]. It is a means of providing compelling evidence that a problem in NP is not in P. Cook [11] defines the following:

DEFINITION 2: Suppose that L_i is a language over $\Sigma_i, i = 1, 2$. Then $L_1 \leq_p L_2$ (L_1 is polynomially reducible to L_2) iff there is a polynomial-time computable function $f : \Sigma_1 \to \Sigma_2$ such that $x \in L_1 \Leftrightarrow f(x) \in L_2$, for all $x \in \Sigma_1$.

DEFINITION 3: A language L is NP-complete iff L is in NP, and $L' \leq_p L$ for every language L' in NP.

PROPOSITION 1: Given any two languages, L_1 and L_2:

1) If $L_1 \leq_p L_2$ and $L_2 \in P$ then $L_1 \in P$.
2) If L_1 is NP-complete, $L_2 \in NP$, and $L_1 \leq_p L_2$ then L_2 is NP-complete.
3) If $L \in P$ and L is NP-complete, then $P = NP$.

H. 3*SAT* $\leq_p G_3$

The standard reduction [5, Theorem 2] of a 3CNF formula F (that has a set of n clauses C) to a G^n_3 graph $G = (V, E)$ such that G has an n-clique iff F is *satisfiable* is as follows:

1) For each clause C_k in F (where $k < n$), put a triple of vertices in V respectively labelled by the three literals in C_k.
2) For each pair of vertices $i, j \in V$ add an edge (i, j) to E iff the vertices are in different triples, and their corresponding literals are not contradictory.

Fig. 2 is the graph of Formula 2 resulting from this reduction. The number of n-cliques contained in a G^n_3 graph can be greater than the number of possible satisfying assignments, since it also represents each 'partial assignment' to a subset of propositions which also make the formula *True*. For instance, in Fig. 2 there is a 4-clique between $\{x_0, \bar{x}_2\}$.

III. A New Reduction: 3SAT $\leq_p G_7$

In a similar way to the standard G_3 reduction we can reduce 3SAT to G_7. In this case, rather than constructing a graph using the literals as vertices, we use the seven possible satisfying assignments to each clause. Two vertices are adjacent iff there are no contradictory assignments to the literals represented in each vertex.

THEOREM 2: There is a quadratic time reduction from 3SAT to G_7.

Proof: Let $F = \bigwedge_{i<n} C_i$ (for some n) be an instance of 3SAT, where each C_i is a disjunction of exactly three literals. We will define an instance of G_7 from F, in polynomial-time. We can assume that no clause contains a literal and its negation (else we could exclude that clause and the result would be logically equivalent). A *partial valuation* to a clause is a valuation defined on the propositional variables occurring in that clause only. Given a clause $C = (l_0 \vee l_1 \vee l_2)$ there are at most seven partial valuations v to $\{l_0, l_1, l_2\}$ such that $v(C) = \top$ (if the propositional variables in the literals are all distinct then there is one valuation making all three literals false and seven other valuations making at least one valuation true; if the propositional variables are not distinct then there will be less than seven partial valuations).

For each clause C_i $(i < n)$ and each partial valuation v making C_i true, create a node (i, v) of a new graph G. G has at most $7 \times n$ nodes — seven for each clause. To complete the definition of our reduction we must define the edges of G. Let $((i, v), (j, w))$ be an edge of G if $i \neq j$ and v and w do not contradict each other, i.e. we allow this edge so long as there is no propositional variable p such that v and w are both defined on p but one makes p true and the other makes it false. It is easy to see that this graph G is a G_7^n graph, hence an instance of G_7.

Now we must check that the reduction is correct. If F is a *YES* instance of 3SAT, let v be a valuation such that $v(F) = \top$. We must show that there is a n-clique of the graph G. For each clause C_i, let v_i be the restriction of v to the propositional variables in C_i. Since $v(F) = \top$ we must have $v_i(C_i) = v(C_i) = \top$, so (i, v_i) is a node of G. Let $S = \{(i, v_i) : i < n\}$. Since all the v_i's are restrictions of the same global valuation v, none of them can contradict each other. Hence, for any $i, j < n$, $((i, v_i), (j, v_j))$ is an edge of G. Therefore S is a n-clique, so G is a *YES* instance of G_7. Conversely, suppose G is a *YES* instance of G_7, so let S be a n-clique of G. For each $i < k$ there must be one vertex $(i, v) \in S$. We have $(v(C_i) = \top$. Let w be the valuation, defined on all propositions p in F by $w(p) = v(p)$ if there is $i < n$ such that v is defined on p and $(i, v) \in S$. Since S is a clique, no two partial valuations $(i, v), (j, v')$ contradict each other, so this is well-defined. Now, for each $(i, v) \in S$, v is a restriction of w, hence $w(C_i) = v(C_i) = \top$ and therefore $w(F) = \top$, as required.

Creating upto $7n$ nodes takes $O(n)$ time. Checking if v contradicts v' and adding an edge from (i, v) to (j, v') takes constant time. Adding all the edges takes $O(n^2)$ time. □

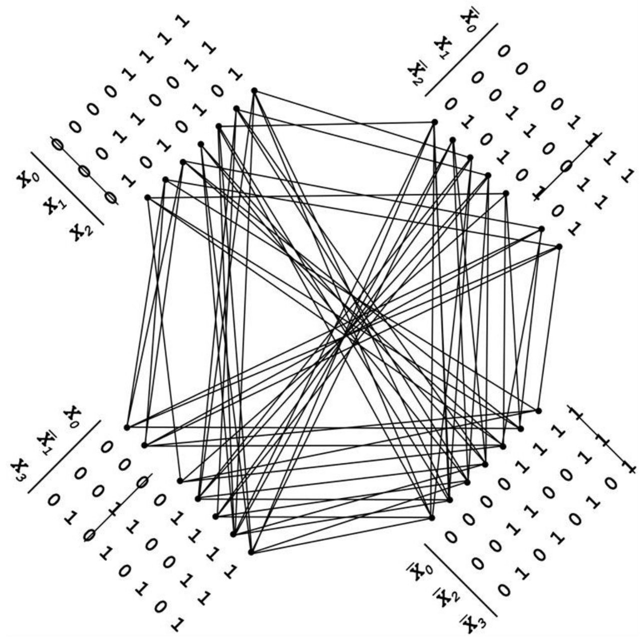

Fig. 3. Formula 2 reduced to a G_7^4 graph.

To illustrate this using an example, Fig.3 represents the graph generated from the 3CNF Formula 2:

$$(x_0 \vee x_1 \vee x_2) \wedge (\bar{x}_0 \vee x_1 \vee \bar{x}_2) \wedge (x_0 \vee \bar{x}_1 \vee x_3) \wedge (\bar{x}_0 \vee \bar{x}_2 \vee \bar{x}_3)$$

Here, the subscript, x_i, indicates the index of the literal, with the superscript, x^a, as its Boolean assignment — i.e., x_2^0 states that the literal x_2 is assigned the Boolean value '0', representing *False*.

EXAMPLE 2: To construct the G_7^n graph from this Formula:

- Convert each partial evaluation of the clause to an 'assignment vertex' (except the unsatisfiable assignment — i.e., $\{x_0^0, x_1^0, x_2^0\}$ in the first clause).
- If necessary, also remove any 'internally' inconsistent vertcies, e.g., $\{x_0^0, x_1^0, x_0^1\}$ could not exist since the assignment x_0^0 contradicts x_0^1.
- Add edges between vertices that are not contradictory, for instance:
  - Create an edge between $\{x_0^0, x_1^0, x_2^1\}$ and $\{x_0^0, \bar{x}_1^1, x_3^1\}$.
  - Do not create an edge between $\{x_0^0, x_1^0, x_2^1\}$ and $\{x_0^0, \bar{x}_1^0, x_3^0\}$ (because the assignment x_1^0 contradicts $\bar{x}_1^0$).

Using this framework the resulting G_7^n graph only represents the n-cliques which correspond to each possible satisfying assignments, i.e., the number of n-cliques equals exactly the total number of assignments that make a formula *True*. This means that since a G_3^n graph can represent 'partial' solutions, it can contain more n-cliques than the total number of n-cliques in the corresponding G_7^n graph.

IV. Discussion

The primary focus of this paper is to introduce a new reduction from 3SAT to G_7, however, to illustrate one benefit of using such a representation, we ran a very basic complete arc-consistency algorithm [4] on a number of 3CNF unsatisfiable SATLIB benchmarks reduced to G_7 and G_3 graph problems.

SATLIB [3] is an online resource for SAT-related research with its core component, a freely distributed benchmark suite of SAT instances and a collection of SAT solvers, aimed to facilitate empirical research on SAT by providing a uniform test-bed for SAT solvers.

Generally, it tends to be 'hard' for polynomial algorithms to correctly solve unsatisfiable instances, so these algorithms are usually used to preliminarily reduce the search-space for backtracking algorithms [12]. To clarify what we mean by 'solved', the arc-consistency algorithm simply prunes edges that cannot be part of a tri-clique and it runs in polynomial-time, $O(n^2)$. Since both the G_7^n and standard G_3^n representations contain an n-clique iff there is a *satisfiable* solution, an *empty* graph (containing no edges) means that there are *no* cliques and hence *no* satisfiable solutions. Hence, initially for our purpose it is interesting to focus primarily on *unsatisfiable* benchmarks.

Table I lists some of the unsatisfiable benchmark instances that this algorithm successfully solves when represented as a G_7^n graph (including the time taken). Moreover, if we reduce these instances to the standard G_3^n representation, the simple arc-consistency algorithm fails to solve a single case correctly.

Uniform Random-3SAT (*UF* and *UUF*) is a family of SAT problems obtained by generating 3CNF formulae, randomly drawing from the $2n$ possible literals with uniform probability. The *AIM* and *DUBOIS* instance are constructed with Random-3SAT instance generators and run in a randomized fashion. Although some SAT-solvers find it difficult to solve the *AIM* instances, it should be noted that these instances can be solved with polynomial preprocessing.

The results are encouraging, and it is speculated that this new framework is advantageous over the standard representation since the G_7^n graphs represent less information than the G_3^n graphs (i.e., the number of n-cliques equals the number of satisfying assignments), resulting in graphs which tend to be significantly less dense with proportionally up to 10 times fewer edges.

Indeed, we intend to further this research by attempting to understand why these polynomial-time algorithms fail to solve some unsatisfiable formulae (graphs with no n-cliques) as well as applying more robust complete polynomial-time algorithms [4] to many more of the SATLIB benchmarks and other pertinent scientific problems.

Though still unproven, the general consensus is that $P \neq NP$ [13]. From our preliminary findings for 3SAT, it would appear that even the most basic of polynomial-time algorithms may work for some of the inputs likely to be encountered in practice. In this sense, using more robust algorithms thus might yield many of the stunning practical benefits to be expected in a world in which $P = NP$ [11].

TABLE I

Some solved unsatisfiable benchmarks represented as G_7^n graphs using a basic arc-consistency algorithm, which fails to solve the G_3^n graphs correctly. Time is in seconds.

benchmark set	**instances**	**props**	**clauses**	**solved**	**avg time**
uuf50-218	1000	50	218	1000	1.16
uuf75-325	100	75	325	55	25.05
uuf100-430	1000	100	430	4	120.45
dubois#	11	60-90	160-240	11	0.24-0.75
dubois100	1	300	800	1	30.2
aim-50-1_6-no	4	50	80	4	0.032
aim-50-2_0-no	4	50	100	4	0.062
aim-100-1_6-no	4	100	160	4	0.27
aim-100-2_0-no	4	100	200	4	0.40
aim-200-1_6-no	4	200	320	4	1.85
aim-200-2_0-no	4	200	400	4	3.00

References

[1] V. Kumar, "Algorithms for constraint-satisfaction problems: A survey," *AI Magazine*, vol. 13, no. 1, pp. 32–44, 1992. [Online]. Available: citeseer.ist.psu.edu/article/kumar92algorithms.html

[2] E. Tsang, *Foundations of Constraint Satisfaction*. London: Academic Press, 1993. [Online]. Available: http://cswww.essex.ac.uk/CSP/papers/Tsang-Fcs1993.pdf/

[3] H. H. Hoos and T. Stützle, "Satlib: An online resource for research on SAT," in *SAT'2000*. IOS Press, 2000, pp. 283–292. [Online]. Available: citeseer.ist.psu.edu/hoos00satlib.html

[4] J. K. Pearson and P. G. Jeavons, "A survey of tractable constraint satisfaction problems," Royal Holloway University of London, Tech. Rep. CSD-TR-97-15, 1997. [Online]. Available: citeseer.ist.psu.edu/pearson97survey.html

[5] S. A. Cook, "The complexity of theorem-proving procedures," in *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press, 1971, pp. 151–158.

[6] B. A. Trakhtenbrot, "A survey of Russian approaches to perebor (brute-force search) algorithms," *Annals of the History of Computing*, vol. 6, no. 4, pp. 384–400, Oct./Dec. 1984, partial English translation of L. Levin, *Universal Search Problems*, 9(3), pp. 265–266, (1973). [Online]. Available: http://dlib.computer.org/an/books/an1984/pdf/a4384.pdf

[7] M. Sipser, "The history and status of the P versus NP question," in *In Proceedings of the 24th ACM Symposium on Theory of Computing*, 1992, pp. 603–618. [Online]. Available: citeseer.ist.psu.edu/sipser92history.html

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[9] P. Jeavons, D. Cohen, and J. Pearson, "Constraints and universal algebra," *Annals of Mathematics and Artificial Intelligence*, vol. 24, no. 1-4, pp. 51–67, 1998. [Online]. Available: citeseer.ist.psu.edu/106745.html

[10] L. Adleman and K. Manders, "Reducibility, randomness, and intractibility (abstract)," in *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press, 1977, pp. 151–163.

[11] S. A. Cook, "The P versus NP problem," 2000, computer Science Department, University of Toronto. Available at http://www.cs.toronto.edu/sacook/homepage/PvsNP.ps. [Online]. Available: citeseer.ist.psu.edu/302888.html

[12] J. Gu, P. Purdom, J. Franco, and B. Wah, "Algorithms for the satisfiability (sat) problem: a survey," 1996. [Online]. Available: citeseer.ist.psu.edu/56722.html

[13] L. A. Hemaspaandra, "Sigact news complexity theory column 36," *SIGACT News*, vol. 33, no. 2, pp. 34–47, 2002.