Parametrizing Above Guaranteed Values: MaxSat and MaxCut

Meena Mahajan and Venkatesh Raman
The Institute of Mathematical Sciences,
C. I. T. Campus, Chennai, India 600 113.
{meena,vraman}@imsc.ernet.in

Abstract

In this paper we investigate the parametrized complexity of the problems MaxSat and MaxCut using the framework developed by Downey and Fellows[7].

Let G be an arbitrary graph having n vertices and m edges, and let f be an arbitrary CNF formula with m clauses on n variables. We improve Cai and Chen's $O(2^{2k}m)$ time algorithm for determining if at least k clauses of of a c-CNF formula f can be satisfied[4]; our algorithm runs in $O(|f|+k^2\phi^k)$ time for arbitrary formulae and in $O(m+k\phi^k)$ time for c-CNF formulae. We also give an algorithm for finding a cut of size at least k; our algorithm runs in $O(m+n+k4^k)$ time.

Since it is known that G has a cut of size at least $\lceil \frac{m}{2} \rceil$ and that there exists an assignment to the variables of f that satisfies at least $\lceil \frac{m}{2} \rceil$ clauses of f, we argue that the standard parametrization of these problems is unsuitable. Non-trivial situations arise only for large parameter values, in which range the fixed-parameter tractable algorithms are infeasible. A more meaningful question in the parametrized setting is to ask whether $\lceil \frac{m}{2} \rceil + k$ clauses can be satisfied, or $\lceil \frac{m}{2} \rceil + k$ edges can be placed in a cut. We show that these problems remain fixed-parameter tractable even under this parametrization. Furthermore, for upto logarithmic values of the parameter, our algorithms run in polynomial time.

We also discuss the complexity of the parametrized versions of these problems where all but k clauses have to satisfied or all but k edges have to be in the cut.

1 Introduction

Given a boolean formula in conjunctive normal form (CNF) f with m clauses on n variables, the MaxSat problem asks for the maximum number of clauses that can be satisfied by any assignment. The decision version of the problem which asks whether at least k of the clauses can be satisfied is NP-Complete when k is an integer given as part of the input[11].

Given a graph $G = (V, E)$ on n vertices and m edges, the MaxCut Problem asks for the maximum number of edges that can be placed in a cut $(S, \bar{S})$ for some $S \subset V$. (An edge is in cut $(S, \bar{S})$ if it has exactly one end point in S.) Again, the decision version which asks whether there is a set $S \subset V$ such that $|(S, \bar{S})| \geq k$ is NP-complete when k is an integer given as part of the input[11]. (This is equivalent to asking whether $\exists E' \subseteq E : |E'| \geq k$ and $G' = (V, E')$ is bipartite.)

We investigate the parametrized complexity of MaxSat and MaxCut using the framework developed by Downey and Fellows[7]. In this framework, a parametrized problem is (uniformly) fixed-parameter tractable (in the class FPT), if there is an $O(g(k)N^{\alpha})$ algorithm for the problem, where g is an arbitrary (could be exponential or worse) function of k, N is the size of the input, and α is a constant independent of k. Standard polynomial time reductions do not necessarily preserve parametrized complexity, so to study problems in this framework we need FPT reductions which preserve independence of the parameter from the input size. FPT reductions refine and expose a rich structure within the NP-complete degree. For a formal exposition of parametrized complexity, see [7, 8].

We first show that MaxSat and MaxCut are both fixed-parameter tractable in the sense described above. In particular, our results mean that for any fixed k, MaxSat and MaxCut have $O(m + n)$ algorithms.

However, these results are not very exciting because both these problems share an important property: for $k \leq \lceil \frac{m}{2} \rceil$ the answer is always yes. In other words, $\lceil \frac{m}{2} \rceil$ is a kind of "guaranteed value" (see Propositions 5 and 13). Thus non-trivial situations arise only for large parameter values, in which range the FPT algorithms described above are infeasible.

Consequently, in a parametrized setting, a more meaningful question to ask is: How much larger than the guaranteed value is the maximum value? In this paper, we show that both MaxSat and MaxCut are FPT in this sense as well; given integer k, checking satisfiability of $(\lceil \frac{m}{2} \rceil + k)$ clauses or existence of an $(\lceil \frac{m}{2} \rceil + k)$ size cut is in FPT.

Note that even for constant k, polynomial-time algorithms for these versions are not obvious. Our FPT results imply polynomial-time algorithms for constant k, in fact for values of k upto logarithmic in input size. For more on logarithmically bounded parametrization, see [17].

There is another way in which these problems can be parametrized. Consider MaxSat. Checking if all clauses can be satisfied is NP-complete. What about all but at most one? All but at most two? All but at most k? The parameter now specifies the distance from the value that is "guaranteed to be hard". However, for any fixed k, we are still looking at NP-complete problems. (The formula f is in SAT iff all but one clause of $f \wedge x \wedge \bar{x}$ can be satisfied, where x is a new variable not occurring in f. For any k, k new variables can be added to f in this fashion.) Not surprisingly, we show that this kind of parametrization gives a problem hard for the class $W[2]$ in the W hierarchy defined by Downey and Fellows[7].

This is evidence that it is unlikely to be in FPT.

For MaxCut, parametrizing in this direction has unexpected implications. At one extreme, checking if all but at most k edges can be put in a cut is also easy for constant k; just cycle through all the $\binom{|E|}{k}$ subsets of E of size $m-k$ testing bipartiteness. At the other extreme, checking if $0 \leq i \leq \lceil \frac{m}{2} \rceil$ edges can be put into the cut is also easy (the answer is always Yes). Can the problem really become hard in the middle range? We show that this "all but at most k" parametrized version of MaxCut is no harder than the corresponding version of even 2-CNF-MaxSat.

Defining the following parametrized problems, the results of this paper are tabulated below:

$$
\begin{aligned}
L_1 &= \{ \langle f, k \rangle \mid \exists \text{ an assignment satisfying at least } k \text{ clauses of the formula } f \} \\
L_2 &= \{ \langle f, k \rangle \mid \exists \text{ an assignment satisfying at least } k + \lceil \tfrac{m}{2} \rceil \text{ clauses of the formula } f \} \\
L_3 &= \{ \langle f, k \rangle \mid \exists \text{ an assignment satisfying at least } m - k \text{ clauses of the formula } f \} \\
C_1 &= \{ \langle G, k \rangle \mid \exists \text{ a cut of size at least } k \text{ in the graph } G \} \\
C_2 &= \{ \langle G, k \rangle \mid \exists \text{ a cut of size at least } k + \lceil \tfrac{m}{2} \rceil \text{ in the graph } G \} \\
C_3 &= \{ \langle G, k \rangle \mid \exists \text{ a cut of size at least } m - k \text{ in the graph } G \}
\end{aligned}
$$

L_1, C_1, L_2, C_2	FPT, in P for logarithmically bound parameters
L_3	W[2] hard
C_3	reducible to the 2-CNF restriction of L_3

Cai and Chen [4] show that the (standard) parametrized versions of all problems in MaxSNP are fixed-parameter tractable; however MaxSat is not in MaxSNP. Our algorithm for L_1 provides the first proof that L_1 is fixed-parameter tractable. Since C_1 is in MaxSNP, it has been known that C_1 is in FPT through the MaxSNP reduction. We give a simpler direct algorithm for C_1.

It has been brought to our notice recently that the hardness of L_3, not just for W[2] but for the entire W hierarchy W[P], is already known [10].

2 Parametrizing MaxSat

We first examine special cases of the maximum satisfiability problem. These are used as building blocks in subsequent algorithms. In the subsequent three subsections, we investigate the parametrized complexity of the languages L_1, L_2 and L_3.

2.1 Handling Special Cases

For the MaxSat problem, some assumptions routinely made in Sat are not reasonable. One such assumption is that to begin with, all clauses are distinct; another, that there are no unit clauses. In our algorithms, we will not make any such assumptions. However, we do assume, without loss of generality, that within a clause, all literals are distinct. We also assume, without loss of generality, that no clause contains both a variable and its negation (no clause is trivially true).

Proposition 1: *If f is a Boolean CNF formula with only unit clauses, then an assignment that satisfies the maximum number of clauses can be found in $O(|f|)$ time.*

Proof: The greedy algorithm of assigning true to each variable with more positive occurrences than negated, and false to all the remaining variables, clearly gives the optimum solution. Occurrences of each variable can be counted in a single scan of f, giving a linear time algorithm. $\Box$

Lemma 2: *Given a Boolean CNF formula f, if there are k clauses in it that each contain at least k literals, then there is an assignment satisfying all these k clauses. The assignment can be found in $O(|f|)$ time.*

Proof: Let $C_1, C_2, \ldots, C_k$ be a set of "long" clauses, each containing at least k literals. A satisfying assignment can be found in k stages as follows: At stage i, pick the first unassigned literal in clause C_i, and set it to true. (There will be one such literal, since only $i-1$ variables have already been assigned values.) After k stages, the remaining variables can be set arbitrarily.

The assignment can be found in linear time by maintaining a flag for each variable indicating whether it has been assigned a value or not. Now, in a single scan, we can identify the long clauses and satisfy them as described above. $\Box$

Corollary 3: *Given a Boolean CNF formula f with m clauses, if each clause contains at least m literals, then f is satisfiable. A satisfying assignment can be found in $O(|f|)$ time.*

Lemma 4: [19] *Let f be a Boolean CNF formula such that each variable occurs at most twice in f (f is an instance of Twice-Sat). Then an assignment satisfying the maximum number of clauses of f can be found in $O(|f|)$ time.*

2.2 The Complexity of L_1

Proposition 5: *Let f be a Boolean CNF formula with m clauses. An assignment satisfying at least $\lceil \frac{m}{2} \rceil$ clauses exists, and can be found in $O(|f|)$ time.*

Proof: Pick any assignment A. If A satisifes $\lceil \frac{m}{2} \rceil$ clauses, we are through. Otherwise, the bitwise complement assignment $\bar{A}$ satisfies all the clauses left unsatisfied by A (and possibly some more too); thus it satisfies $\lceil \frac{m}{2} \rceil$ clauses.

In fact, this argument shows that at least half of all possible assignments satisfy at least $\lceil \frac{m}{2} \rceil$ clauses. $\Box$

Cai and Chen[4] used this observation to prove that the parametrized Max-c-Sat, where every clause has at most c variables, is FPT (and hence every problem in MaxSNP is FPT). Their algorithm relies on the boundedness of the clauses.

We present in Figure 1 a simple algorithm for L_1, the Branching-SAT algorithm. It has FPT complexity for any CNF, with no bound on the number of variables per clause. In particular, for c-CNF, it gives a better time complexity than the algorithm of [4].

Theorem 6: *Given a boolean CNF formula f with n variables and m clauses and an integer k, in $O(k^2 2^k + |f|)$ time, we can find an assignment to the variables that satisfies at*

Branching SAT Algorithm B-S-A-1 (f, k)

input CNF formula f with m clauses, parameter k.

output Yes, if there is an assignment which satisfies at least k clauses of f,
No otherwise.

begin

1 If $k > m$ then output No and halt.

2 If $k \leq \lceil \frac{m}{2} \rceil$ then output Yes and halt.

3 Separate the clauses of f into two sets: long clauses (k or more literals per clause) and short clauses. Let f_l and f_s be the conjunctions of the long clauses and the short clauses respectively; $f = f_l \wedge f_s$. Let f_l have b clauses. If $b \geq k$ then output Yes and halt.

4 Construct any binary tree T of the following type: The pair $\langle f_s, k - b \rangle$ is at the root. In general, each node in the tree is labelled by some formula g and some non-negative integer j.

If j exceeds the number of clauses in g, then $\langle g, j \rangle$ is a leaf node labelled No.

If $j = 0$, then $\langle g, j \rangle$ is a leaf node labelled Yes.

If no variable occurs in g both positively and negatively, then $\langle g, j \rangle$ is a leaf node labelled Yes.

Otherwise, pick a variable v which has a positive and a negative occurrence in g. Let v_t be the number of clauses of g that contain the literal v, and let g_v be the simplified formula obtained by setting v to true in g. Similarly, let v_f be the number of clauses of g that contain the literal $\bar{v}$, and let $g_{\bar{v}}$ be the simplified formula obtained by setting v to false in g. Then $\langle g, j \rangle$ has two children, labelled $\langle g_v, j - v_t \rangle$ and $\langle g_{\bar{v}}, j - v_f \rangle$.

If T has a leaf node labelled Yes then output Yes, else output No.

end

Figure 1: FPT algorithm for satisfying k clauses (L_1)

least k clauses of the formula or discover that no such assignment exists.

For instances of c-Sat, the running time is $O(ck2^k + |f|)$.

Proof: Consider first the algorithm B-S-A-1 from Figure 1 for the decision version.

Correctness Steps 1,2,3 tackle simple cases. In step 4, at a node $\langle g, j \rangle$, the algorithm tries to find if at least j clauses of g are satisfiable. If the answer is not trivially known (for trivial cases, the node is a leaf labelled Yes or No appropriately), then any assignment to g must set each variable to either true or false. Instead of exploring all assignments, the algorithm explores only carefully chosen partial assignments. It is easy to see that if there is an assignment satisfying j or more clauses of g, then some partial version of this assignment is constructed along some path from $\langle g, j \rangle$ to a leaf labelled Yes. And if there is no such assignment, then all leaves of the subtree rooted at $\langle g, j \rangle$ are labelled No.

Time Analysis From Proposition 5 and Lemma 2, it follows that the first three steps run in linear time.

In the fourth step, the binary tree T is constructed. At a node $\langle g, j \rangle$, the processing required to find the descriptions of its children, or to discover that it is a leaf node, requires $O(|g|)$ time. If this is not a leaf node, then its children are labelled by integers strictly smaller than j and by formulae of length less than $|g|$. Since the root is labelled with $\langle f_s, k - b \rangle$, the tree is of depth at most $k - b$ and hence size at most 2^{k-b}. So it can be constructed in $O(|f_s| 2^{k-b})$ time. But step 4 is executed only if $m < 2k$, and f_s has only short clauses of length at most k. So $|f_s| < 2k^2$, and T can be constructed in $O(k^2 2^k)$ time.

The modification to tackle the search version is as follows: Search versions of steps 2 and 3 follow from Proposition 5 and Lemma 2 respectively. Step 4 is constructive and gives a partial assignment satisfying $k - b$ short clauses. Under this partial assignment (which assigns values to at most $k - b$ variables), some clauses of f_l may already be satisfied. So there are at most b clauses left in f_l, each of reduced length at least $k - (k - b) = b$, and Lemma 2 is again applicable.

Further, if the input is a c-CNF formula, then at step 4 we know that $m < 2k$ and hence $|f_s| \leq cm < 2ck$. So the running time is only $O(ck2^k + |f|)$. □

In step 4, if the node being processed is not a leaf node, we have to choose a variable v for branching. Apart from checking that v occurs both positively and negatively, we can be clever and pick such a variable that appears in the maximum number of clauses. The idea is to push the integer part of the node label $\langle g, j \rangle$ to 0 faster. This idea is used in the algorithm described in Figure 2, which establishes the following theorem.

Theorem 7: *Given a boolean CNF formula f with n variables and m clauses and an integer k, in $O(k^2 \phi^k + |f|)$ time, we can find an assignment to the variables that satisfies at least k clauses of the formula or discover that no such assignment exists. Here ϕ is the golden ratio $(1 + \sqrt{5})/2$.*

For instances of c-Sat, the running time is $O(ck\phi^k + |f|)$.

Proof: The algorithm B-S-A-2 solves the decision version, and can be modified to solve the search version as described in the proof of Theorem 6. Its correctness follows from that of Theorem 6 and Lemma 4.

Modified Branching SAT Algorithm B-S-A-2 (f, k)

input CNF formula f with m clauses, parameter k.

output Yes, if there is an assignment which satisfies at least k clauses of f,
No otherwise.

begin Steps 1,2,3 are as in B-S-A-1.

Step 4: Construct the following binary tree T: The pair $\langle f_s, k-b\rangle$ is at the root. In general, each node in the tree is labelled by some formula g and some non-negative integer j.

If j exceeds the number of clauses in g, then $\langle g, j\rangle$ is a leaf node labelled No.

If $j = 0$, then $\langle g, j\rangle$ is a leaf node labelled Yes.

If no variable occurs in g both positively and negatively, then $\langle g, j\rangle$ is a leaf node labelled Yes.

Otherwise, of all the variables that occur in g both positively and negatively, choose that variable v which occurs the maximum number of times.

If v occurs only twice in g, then $\langle g, j\rangle$ is a leaf node whose label Yes/No is determined as follows: First, set all variables that occur only positively or only negatively in g to true and false respectively. This satisfies say p_1 clauses, and simplifies g to a formula g' where each variable occurs at most twice. Using the method of Lemma 4, find p_2, the maximum number of clauses satisfiable in g'. If $p_1 + p_2 \geq j$, then $\langle g, j\rangle$ is a leaf node labelled Yes, otherwise $\langle g, j\rangle$ is a leaf node labelled No.

If v occurs more than twice, then $\langle g, j\rangle$ has two children $\langle g_v, j - v_t\rangle$ and $\langle g_{\bar{v}}, j - v_f\rangle$, as in B-S-A-1.

Figure 2: Modified Branching Sat algorithm

To see its time complexity, note that steps 1,2,3 are the same as in B-S-A-1 and hence run in linear time.

At step 4, the processing at each node $\langle g, j\rangle$ is of two types: (1) find the labels of the two children, or (2) label the leaf. The first type of processing is easily seen to be possible in $O(|g|)$ time, as in B-S-A-1. The second type can also be done in $O(|g|)$ time, using an appropriate linked list representation for the variables and clauses of g and then using the result of Lemma 4. So either way, a node can be processed in $O(|g|)$ time.

If node $\langle g, j\rangle$ is not a leaf, both its children have labels with integers strictly less than j. Furthermore, since $v_t + v_f \geq 3$, at least one of v_t and v_f exceeds one, and so at least one child has a label with integer strictly less than $j - 1$. So the size of the tree constructed satisfies the Fibonacci recurrence: $T(j) \leq 1 + T(j-1) + T(j-2)$. Since its root is labelled $k - b$, the number of nodes in the binary tree is bounded by the solution to this recurrence at $k - b$, which is ϕ^{k-b} where ϕ is the golden ratio $(1 + \sqrt{5})/2$.

Thus the entire tree can be constructed and labelled in time $O(\phi^{k-b}|f_s|)$. As argued earlier, $|f_s| \in O(k^2)$ and so the algorithm runs in time $O(k^2\phi^k + |f|)$. For instances of c-SAT, $|f_s| = O(ck)$, and the run time is $O(ck\phi^k + |f|)$.

$\square$

2.3 The Complexity of L_2

Recall that if f is a CNF formula with m clauses, then the pair $\langle f, k\rangle$ belongs to L_2 iff at least $\lceil \frac{m}{2} \rceil + k$ clauses of f can be satisfied.

Proposition 8: *Let f be a Boolean CNF formula with m clauses, of which p clauses are not unit clauses. An assignment satisfying at least $\lceil \frac{m}{2} \rceil + \frac{p}{4} - 1$ clauses exists, and can be found in $O(|f|)$ time.*

Proof: If we set each variable to 0 or 1 uniformly and independently, then the expected number of clauses satisfied by this random assignment will be at least $\frac{m}{2}$. Something stronger is true. Since the random assignment satisfies a non-unit clause with probability at least $\frac{3}{4}$, it follows that if f has p non-unit clauses, then the expected number of clauses satisfied by the random assignment is at least $\frac{m-p}{2} + \frac{3p}{4} = \frac{m}{2} + \frac{p}{4}$. Thus there is some assignment satisfying at least $\lceil \frac{m}{2} + \frac{p}{4} \rceil \geq \lceil \frac{m}{2} \rceil + \frac{p}{4} - 1$ clauses. $\square$

This argument gives us a randomized algorithm for finding a good assignment, one which satisfies at least $\lceil \frac{m}{2} \rceil + \frac{p}{4} - 1$ clauses. The algorithm can be derandomized using the method of conditional expectations, to construct a good assignment in $O(|f|)$ time. For details, see [21].

If f has more than $4k+3$ non-unit clauses, the above argument guarantees an assignment satisfying at least $\lceil \frac{m}{2} \rceil + k$. So now we can assume without loss of generality that f has at most $4k + 3$ non-unit clauses. An algorithm for the decision version is shown in Figure 3. (The search version of finding such an assignment is obtained by an easy modification and by including the derandomization described above.)

Theorem 9: *Given a boolean CNF formula f with n variables and m clauses, and an integer k, we can in $O(|f| + k^2\phi^{6k})$ time find an assignment to the variables that satisfies at least $\lceil \frac{m}{2} \rceil + k$ clauses or discover that such an assignment does not exist.*

Large Sat Algorithm L-S-A (f, k)

Input formula f with m clauses on n variables, integer k.

Output Yes, if at least $T = \lceil \frac{m}{2} \rceil + k$ clauses of f can be satisfied, otherwise No.

begin

Let U be the set of unit clauses and N the set of non-unit clauses, $|U| + |N| = m$.

If $N = \emptyset$, solve the pure-unit-clauses instance using the proof of Proposition 1.

Elseif $|N| \geq 4k + 4$, report Yes.

Else While there are unit clauses v and $\overline{v}$ in U, modify f by removing both clauses.
This decreases m by 2 and T by 1.
Now, if $|U| \geq \lceil \frac{m}{2} \rceil + k$, output Yes.
Otherwise, report B-S-A-2(f, T).

End

Figure 3: FPT algorithm for satisfying $\lceil \frac{m}{2} \rceil + k$ clauses (L_2)

Proof:

Correctness If $|N|$ is 0, or exceeds $4k + 4$, the correct decision is arrived at following Propositions 1 and 8. Otherwise, the while loop is executed. This removes clauses of the form v and $\bar{v}$ in pairs, since exactly one of these is satisifed by any assignment. Once the while loop is completed, there are no conflicting clauses within U. So all the clauses in U can be satisfied by setting the variables occurring in U appropriately. If this number itself is large enough ($|U| \geq \lceil \frac{m}{2} \rceil + k$), then there is nothing left to be done. Otherwise, the algorithm B-S-A-2, whose correctness we have already seen, is invoked.

Time Analysis The first three steps are trivial. Note that when B-S-A-2 is called, $|U| < \lceil \frac{m}{2} \rceil + k$, and $|N| \leq 4k+3$, so $m = |U| + |N| \leq (\lceil \frac{m}{2} \rceil + k - 1) + (4k+3)$, hence $\lfloor \frac{m}{2} \rfloor \leq 5k+2$. So the number of clauses to be satisfied, $T = \lceil \frac{m}{2} \rceil + k$, is bounded by $6k + 3$. The time bound now follows from Theorem 7.

$\square$

Corollary 10: *Given a boolean CNF formula f with n variables and m clauses and an integer $k \in O(\log mn)$, we can find an assignment to the variables that satisfies at least $\lceil \frac{m}{2} \rceil + k$ clauses of the formula, or discover that no such assignment exists, in polynomial time.*

To decide membership of $\langle f, k \rangle$ in L_1 (where $k \geq \lceil \frac{m}{2} \rceil$), one could use L-S-A($f$,$k - \lceil \frac{m}{2} \rceil$) instead of using B-S-A-2(f, k). This turns out to be a better option provided $k < 3m/5$.

2.4 The Complexity of L_3

If f is a CNF formula with m clauses, then the pair $\langle f, k\rangle$ belongs to L_3 iff the deletion of at most k clauses from the formula f makes it satisfiable. That is, at least $m-k$ clauses of f can be satisfied.

We show that L_3 is hard for the complexity class $W[2]$ (see [7] for the definition of the W-hierarchy). This essentially means that L_3 is unlikely to be in FPT. We show the $W[2]$ hardness of L_3 by reducing the $W[2]$-hard dominating set problem to it. The parametrized dominating set problem is the following: given a graph $G=(V,E)$ and an integer k, is there a subset $V' \subseteq V$, $|V'| \leq k$, such that $\forall i \in V$, $\exists j \in N[i]$: $j \in V'$? Here $N[i]$ for a vertex i is the closed neighbourhood of i (containing i and all its neighbours). The reduction is as follows: Given $G=(V,E)$, introduce a variable x_i for vertex i of G, and consider the formula

$$F = \bar{x}_1 \wedge \bar{x}_2 \wedge \ldots \bar{x}_n \wedge \bigwedge_{i=1}^{n} \left(\vee_{j\in N[i]}\ (x_j)\right)$$

It is easy to see that there is a dominating set of size at most k in G if and only if there is an assignment to F that satisfies all but at most k clauses. Thus we have

Theorem 11: *Given a boolean CNF formula and an integer k, the problem of deciding whether there is an assignment to the formula that satisfies all but at most k of the clauses is $W[2]$-hard.*

It has been brought to our notice recently that the hardness of L_3, not just for W[2] but for the entire W hierarchy W[P], is already known [10].

We observe that even if the clause sizes are bounded by three, L_3 remains hard; the standard polynomial-time reduction (see for example [11]) from an unrestricted SAT formula f to a 3-SAT formula f' has the property that $(f,k) \in L_3$ if and only if $(f',k) \in L_3$. Thus we have

Theorem 12: *Given a boolean CNF formula in which each clause has at most three variables, and an integer k, the problem of deciding whether there is an assignment to the formula that satisfies all but at most k of the clauses, is $W[2]$-hard.*

However, we do not know the parametrized complexity of the 2-CNF restriction of L_3.

If $L_3(k) = \{\langle f,k\rangle \mid \langle f,k\rangle \in L_3\}$ is fixed-paramater tractable for any fixed k, then P = NP. (This is because Sat can be framed as an $L_3(k)$ instance for any k by adding some clauses.) Therefore, if $L_3(k)$ is in $W[i]$, then $W[i]$ in FPT would imply P = NP. Such a strong consequence of the W-hierarchy collapse appears unlikely; the best known consequence of a $W[i]$-complete problem being FPT is that the W hierarchy up to the class $W[i]$ collapses to FPT. Therefore, even for a fixed k, it appears unlikely that $L_3(k)$ is in any level of the W hierarchy.

This version of MaxSat, L_3, is somewhat different from most other parametrized problems in that it remains NP-complete even for a constant k. (It has no known algorithm with run time $|f|^{b(k)}$, where b can be an arbitrary function of k. An algorithm with such a run time would in fact imply P=FPT=W[P]=NP.) The parametrized complexity framework is not really geared to address such problems.

3 Parametrizing MaxCut

We start with a well known proposition that every graph has a cut of size $\lceil \frac{m}{2} \rceil$ and such a cut can be found in linear time.

Proposition 13: *Given a graph G with n vertices and m edges, a cut in G of size at least $\lceil \frac{m}{2} \rceil$ can be found in $O(m+n)$ time.*

Proof: Pick a random partition $(S, \bar{S})$ of V by placing each vertex of V in S uniformly and independently with probability $\frac{1}{2}$. Each edge (u, v) is in the cut $(S, \bar{S})$ with probability exactly half. So the expected cut size on a random partition is $\frac{m}{2}$. Furthermore, by a derandomization technique based on conditional expectations, similar to the one used for MaxSat, we can find a cut of size at least $\lceil \frac{m}{2} \rceil$ in $O(m+n)$ time. □

We now consider the parametrized versions of Maxcut as defined in Section 1.

3.1 The Complexity of C_1 and C_3

The language C_1 is the direct decision version of the maximization problem. As the MaxCut problem is in the class MaxSNP, C_1 is fixed-parameter tractable by the result of [4]. We propose some simple algorithms which directly place C_1 in FPT without recourse to the MaxSNP reduction. We also comment on the complexity of C_3.

First, reduce MaxCut to Max2Sat using the following reduction from [21]: create a variable v for every vertex v in the graph. For every edge (i, j) in the graph, create two 2-literal clauses $\{\bar{i} \vee \bar{j}\}$ and $\{i \vee j\}$. The formula f_G is the conjunction of these $2m$ clauses where m is the number of edges in the graph.

The following Proposition follows easily by identifying variables set to true with vertices in S.

Proposition 14: *Given a graph G with m edges, let f_G be the corresponding formula described above. G has a cut of size exactly p if and only if there is an assignment satisfying exactly $m+p$ of the $2m$ clauses of f_G.*

Corollary 15: *The parametrized language C_3 fixed-parameter reduces to the subset of L_3 where the input formula f is in 2-CNF (it has at most two literals per clause).*

Algorithms for C_1

Input: A graph G with n vertices and m edges, an integer k.

If $k \leq \lceil \frac{m}{2} \rceil$, then the required cut can be found in $O(m+n)$ time using Proposition 13. Otherwise, any one of the following methods may be used.

Method 1 Construct f_G as described in Propostion 14 and call the algorithm L-S-A(f_G,k). From Theorem 9, this algorithm requires $O(m+n+k^2\phi^{6k})$ time.

Method 2 Instead of calling L-S-A as in Method 1, note that at this stage $m < 2k$ and so $m+k < 3k$. So construct f_G and call the algorithm B-S-A-2(f_G, $m+k$) given in the proof of Theorem 7. The running time is $O(m+n+k^2\phi^{3k})$.

Method 3 At this stage $m < 2k$. Without loss of generality, we can remove isolated vertices, since they do not contribute any edges to any cut. So $n \leq 2m$, i.e. $n < 4k$. Find the maximum cut in G as follows: cycle through all subsets $S \subseteq V$, for each S, find the size of the cut $(S, \bar{S})$, and keep track of the maximum cut. If this is at least k, report Yes, otherwise report No. There are at most 2^n subsets, and processing each subset requires $O(m)$ time, so this method requires $O(m2^n)$ time. Since $n \leq 4k$ and $m \leq 2k$, this algorithm has a run time of $O(m + n + k2^{4k})$.

Method 4 Cycle through all subsets $H \subseteq E$ of size exactly k. For each H, check if the graph (V, H) is bipartite. If any such subset is found, report Yes, otherwise report No. There are at most $\binom{m}{k}$ subsets, and processing each subset requires $O(m) = O(2k)$ time. Since $\binom{m}{k} \leq 2^m < 2^{2k}$, this algorithm has a run time of $O(m + n + k2^{2k})$.

The theorem below follows from Method 4.

Theorem 16: *Given a graph G with n vertices and m edges, and an integer k, we can in $O(m + n + k2^{2k})$ time find a cut of size at least k in G or discover that such a cut does not exist.*

3.2 The Complexity of C_2

For C_2, we develop two FPT algorithms: one has the feature that it runs in polynomial time for logarithmically bounded parameter values, and the other runs in linear ($O(m+n)$) time when k is a constant.

For the first algorithm, we use the following result [1, 2].

Theorem 17: (Edwards[1, 2]) *Given a connected graph G with no self-loops, there exists a bipartite subgraph of G with at least $\frac{m}{2} + \frac{1}{2}\left\lceil \frac{n-1}{2} \right\rceil$ edges.*

We use the simpler version: every connected graph without self-loops has a cut of size at least $\frac{m}{2} + \frac{n-1}{4}$. Consequently, if a graph has c components (and no self-loops), then it has a cut of size at least $\frac{m}{2} + \frac{n-c}{4}$.

Figure 4 shows our FPT algorithm for the decision version of C_2. Step 1 runs in $O(m+n)$ time, using any standard connected components algorithm. In step 2, in each G_i, the time spent is $O(m_i 2^{n_i})$, which is at most $O(m_i 2^{4k})$. So the total time required is $O(m2^{4k} + m + n)$.

For the search version, step 2 already constructs the cut as required. Step 1 (when $k \leq \frac{n-c}{4}$) can be made constructive by using an $O(n^3)$ time implementation of Edward's proof, due to Poljak and Turzik [18]. Thus we have the following theorem.

Theorem 18: *Given a graph G and a parameter k, in $O(n^3 + m2^{4k})$ time we can find a cut of size at least $\lceil \frac{m}{2} \rceil + k$ if one exists in G. The decision version runs in $O(n + m + m2^{4k})$ time.*

Corollary 19: *Given a graph G and a parameter $k \in O(\log mn)$, we can find a cut of size at least $\lceil \frac{m}{2} \rceil + k$ in G, if one exists, in polynomial time.*

A different approach, which we describe below, gives an FPT algorithm with run time $O(m + n + e(k))$, where e is a function of k alone. For small values of k, this is better,

Large Cut Algorithm L-C-A-1 (G, k)

Input A graph $G = (V, E)$ with n vertices and m edges, an integer k. Without loss of generality, assume that G has no isolated vertices and no self loops.

Output Yes if there is a cut of size $\lceil \frac{m}{2} \rceil + k$ in G; otherwise No.

begin

Step 1 Find the connected components $G_1, G_2, \ldots, G_c$ of G. Let G_i have n_i vertices and m_i edges. If $k \leq \frac{n-c}{4}$, then output Yes and halt.

Step 2 Otherwise, $n - c < 4k$. For each i, $n_i \leq n - (c - 1) \leq 4k$. For each G_i, find the maximum cut size s_i using the Method 3 described in the previous subsection. If $\sum_{i=1}^{c} s_i \geq \lceil \frac{m}{2} \rceil + k$, then output Yes and halt, otherwise output No and halt.

end

Figure 4: Algorithm for MaxCut C_2

since the $O(m)$ and $O(n)$ terms are additive. However, it does not give polynomial time algorithms for logarithmic values of k.

For this algorithm, we first note the following stengthening of the proposition that every graph with m edges has a cut of size at least $\lceil \frac{m}{2} \rceil$.

Theorem 20: [12] *Given a graph G, and an arbitrary matching M in the graph, there exists a cut in the graph of size at least $\lceil \frac{m}{2} \rceil + \lfloor \frac{|M|}{2} \rfloor$.*

We exploit this result in our second FPT algorithm for C_2. Figure 5 shows the algorithm for the decision version.

To see why this is an FPT algorithm, note that Steps 1 and 2 run in linear time. If Step 3 is reached, then we already know that $|M| < 2k$, so $|S| \leq 4k - 2$. Also, since M is a maximal matching, S forms a vertex cover. And the cut $(S, \bar{S})$ is not large enough. Thus the following inequalities hold:

$$\begin{aligned}
\lceil \tfrac{m}{2} \rceil + k &> \text{size of the cut } (S, \bar{S}) \\
&= m - \text{ the number of edges with both end points in } S \\
&\qquad \text{(no edge has both endpoints in } \bar{S} \text{ because } M \text{ is maximal)} \\
&\geq m - \binom{|S|}{2} \\
&\geq m - (2k - 1)(4k - 3) \\
\text{Hence } \lceil \tfrac{m}{2} \rceil + k &\leq (2k - 1)(4k - 3) + 2k
\end{aligned}$$

Thus if Step 3 is reached, we know that m, and hence $\lceil \frac{m}{2} \rceil + k$, are $O(k^2)$. So is n, since there are no isolated vertices. Using the algorithm for C_1 at this stage yields a running time of $O(k^2 2^{dk^2} + m + n)$ for some constant d.

For the search version, steps 2 and 3 already construct cuts as required. To make Step 1

Large Cut Algorithm L-C-A-2 (G, k)

Input A graph $G = (V, E)$ with n vertices and m edges, an integer k. Without loss of generality, assume that G has no isolated vertices.

Output Yes if there is a cut of size $\lceil \frac{m}{2} \rceil + k$ in G; otherwise No.

begin

Step 1 Find a maximal matching M in the graph. If $|M| \geq 2k$, then output Yes and halt.

Step 2 Let S be the set of vertices of the matching M. If the cut $(S, \bar{S})$ is of size at least $\lceil \frac{m}{2} \rceil + k$ then ouput Yes and halt.

Step 3 Call the algorithm for C_1 (described in the preceding subsection, Theorem 16), with parameter $(\lceil \frac{m}{2} \rceil + k)$.

end

Figure 5: Algorithm for MaxCut C_2

constructive, one approach is to use a recursive implementation of the proof of Theorem 20. We give in Figure 6 a bottomup approach of the same construction that takes $O(m)$ time. For a proof that this produces a cut of the required size, see [12].

Thus we have the following theorem.

Theorem 21: *Given a graph G and a parameter k, in $O(k^2 2^{dk^2} + m + n)$ time where d is a constant, we can find a cut of size at least $\lceil \frac{m}{2} \rceil + k$ if one exists in G.*

In keeping with the philosophy of looking at parameter values over and above the guaranteed values, Theorem 17 suggests that the parametrized complexity of C_4 defined below is also of interest. We do not know of any FPT algorithm for this problem.

$C_4 = \{\langle G, k \rangle \mid \exists$ a cut of size at least $k + \frac{m}{2} + \frac{1}{2} \left\lceil \frac{n-1}{2} \right\rceil$ in the graph G $\}$

4 Conclusions

We have investigated the fixed-parameter complexity of the MaxSat and MaxCut problems. One could also consider the corresponding MinSat and MinCut problems. The MinCut problem has several polynomial time algorithms[15, 16]. For a recent algorithm not based on network flows, see [20]. In the MinSat problem the aim is to find the minimum number of clauses of a given formula that must be satisfied by any truth assignment. Kohli, Krishnamurti and Mirchandani[13] have shown that the decision version of this problem is NP-complete. We can easily show (by an appropriate "UnSat" version of the B-S-A-1 algorithm of Section 2.2, or by using the FPT reduction[14] to Vertex Cover) that the parametrized MinSat language $L_4 = \{\langle f, k \rangle \mid \exists$ an assignment satisfying at most k clauses

Cut Construction including Matching (G, M)

input A graph $G = (V, E)$, a matching in G given as a list of edges
$M = \langle (u_1, v_1), (u_2, v_2) \ldots, (u_{|M|}, v_{|M|}) \rangle$.

Output A set $S \subseteq V$ such that the cut $(S, \bar{S})$ contains all the matched edges and at least half of the non-matched edges.

begin

Initialise W (the set of vertices already examined) and $(S, W - S)$ (the cut constructed so far).
$W = \{u_1, v_1\}$, $S = \{u_1\}$.

For $i = 2$ to $|M|$ do
    $W = W \cup \{u_i, v_i\}$, $S_1 = S \cup \{u_i\}$, $S_2 = S \cup \{v_i\}$.
    If $(S_1, W - S_1)$ gives a larger cut than $(S_2, W - S_2)$
    in the subgraph induced by W
        then $S = S_1$
        else $S = S_2$.
EndFor
While there is a vertex $v \in V - W$ do
    $W = W \cup \{v\}$, $S' = S \cup \{v\}$
    If $(S', W - S')$ gives a larger cut than $(S, W - S)$
    in the subgraph induced by W
        then $S = S'$.
EndWhile

end

Figure 6: Algorithm for Cut Construction

of the CNF formula f } is fixed-parameter tractable.

We make some observations based on our investigation which suggests possible directions to pursue in the area of parametrized complexity.

- In the parametrized versions, $\leq k$ (or $\geq k$) questions and $= k$ questions can have different parametrized complexity. For instance, consider the MinWtSat problem, on bounded CNF formulae, which asks for a satisfying assignment with the minimum weight (the number of "true" variables). The parametrized language

  $$L_5(c) = \{\langle f, k\rangle \mid f \text{ is a } c\text{-Sat formula with a satisfying assignment of weight } k\}$$

  is NP-complete even for $c = 2$, as witnessed by the following reduction from Independent Set. Given a graph $G = (V, E)$ and an integer k, construct the formula $f = \wedge_{(i,j)\in E}(\bar{x}_i \vee \bar{x}_j)$. Then there is a weight k assignment satisfying f if and only if G has an independent set of size k. (What is more, in f every variable appears only negatively.) Since the Independent Set problem is $W[1]$-complete[8], it follows that $L_5(c)$ is $W[1]$-hard even for $c = 2$.

  However, the related parametrized language

  $$L_6(c) = \{\langle f, k\rangle \mid f \text{ is a } c\text{-Sat formula with a satisfying assignment of weight at most } k\}$$

  is fixed-parameter tractable. To see this, consider an algorithm which scans the input for a positive clause (a clause with no negated literals). (If no such clauses exist, setting all variables to false satisfies all clauses.) On finding such a clause, it branches along c paths, setting one of the variables of this clause to true along each path. The tree so constructed is not explored beyond depth k, so the algorithm is an FPT algorithm. When there is no bound on the number of variables per clause, Downey and Fellows[7], and Cai and Chen[4, 5, 6] have independently shown that the problem is $W[2]$-complete.

  This difference between the complexities of $L_5(c)$ and $L_6(c)$ is in sharp contrast to the observation by Cai and Chen[4] that $Q_{=k}$, $Q_{\leq k}$ and $Q_{\geq k}$ versions, are all FPT equivalent for any optimization problem Q if the parametrized questions are of the form "Is the size of the optimum solution $=, \leq$ or $\geq k$". Querying the existence of solutions of a particular size is, in many settings, a more natural decision version than bounding the optimum value, and in this setting, the $Q_{=k}$, $Q_{\leq k}$ and $Q_{\geq k}$ versions require independent analysis.

  The natural direction to pursue is whether there are other examples of this type. We conjecture that the languages L_1, L_2, C_1, C_2 we investigated in this paper fall in this category — i.e. their corresponding "=k" versions are hard for the W hierarchy. In fact, even if the instance of L_1 consists of unit clauses alone, we do not know whether the "=k" version is in FPT.

- Study of parametrized complexity is useful even to design polynomial algorithms when the parameter values are bounded. For example, the only way we know that the languages $L_1(k), L_2(k)$ and $C_2(k)$ are polynomial time recognizable if k is a constant, is by observing that they are fixed-parameter tractable.

- Even if a problem is fixed-parameter tractable, designing simpler and more efficient algorithms is worth pursuing. As the dependence of run time on k improves, this will give us feasible algorithms for larger ranges of the parameter. Our $O(m + \phi^k k)$ algorithm compared to Cai and Chen's $O(2^{6k} m)$ for the standard parameterized Max3Sat, and the $O((1.324718)^k k^2 + nk)$ algorithm[3] for vertex cover improving over the straightforward $O(2^k n)$ algorithm are clear examples of this direction.

  Even more dramatically, an FPT algorithm where dependence on the parameter is of the form 2^{ck} for some constant c gives polynomial-time algorithms for values of k logarithmic in the input size. Our algorithms for L_1, L_2, C_1 and the first algorithm for C_2 are of this form.

- Invariably, the "at least (at most) k" parametrized version and the "all but k" versions have complementary parametrized complexity as witnessed by the following examples.

  - For Vertex Cover, the "at most k" version is fixed-parameter tractable whereas the "all but at most k" version is the same as the "at least k" version of clique and hence is $W[1]$ complete.
  - For the irredundant set problem, the "at most k" version is $W[1]$-Complete whereas the "all but k" version (called the co-irredundant set problem) is in FPT[9].
  - For the MaxSat problem, the "at least k" version is fixed-parameter tractable (Section 2.1) whereas the "all but at most k" version is $W[2]$-hard (Section 2.3).

  It would be interesting to explore this property in other parametrized problems.

We end with some specific open problems: What is the parametrized complexity of the following?

1. *Parametrized (all but k) Max2Sat*
   Input: A boolean CNF formula in which each clause has at most 2 variables.
   Parameter: $k > 0$.
   Question: Is there an assignment that satisfies all but at most k of the clauses?

2. *Parametrized (all but k) MaxCut*
   Input: A graph G.
   Parameter: $k > 0$.
   Question: Is there a set of at most k edges whose removal makes the graph bipartite?

3. *Parametrized (beyond the large guarantee) MaxCut*
   Input: A graph G.
   Parameter: $k > 0$.
   Question: Does G have a cut of size at least $k + \frac{m}{2} + \frac{1}{2}\left\lceil\frac{(n-1)}{2}\right\rceil$?

In Section 3.1, we have shown that problem 2 above is no harder than the problem 1.

Acknowledgments

We thank Mike Fellows for his comments on an earlier draft, S. Srinivasa Rao for pointing out references [12] and [18] and for suggesting the version of the algorithm B-S-A-1 for L_1 in this paper, and V. Arvind for suggesting an investigation of the parametrized complexity of MaxCut.

References

[1] C. S. Edwards, "Some Extremal Properties of Bipartite Subgraphs", *Can. J. Math.*, **25** (1973) 475–483.

[2] C. S. Edwards, "An Improved Lower Bound for the Number of Edges in a Largest Bipartite Subgraph", in *Recent Advances in Graph Theory* (Academia, Prague, 1975), 167–181.

[3] R. Balasubramanian, M.R. Fellows and Venkatesh Raman, "An Improved Fixed Parameter Algorithm for Vertex Cover,", to appear in *Information Processing Letters.*

[4] L. Cai and J. Chen, "On Fixed-Parameter Tractability and Approximation of NP Optimization Problems," *Proceedings of the 2nd Israel Symposium on Theory and Computing Systems (1993)* 118–126; to appear in *J. Computer and System Sciences.*

[5] L. Cai and J. Chen, "On the amount of nondeterminism and the power of verifying", *Proceedings of the Mathematical Foundations of Computer Science Conference*, Springer-Verlag LNCS **711** (1993) 311–320.

[6] L. Cai and J. Chen, "On Input Read Modes of Alternating Turing Machines" Technical Report of the Department of Computer Science, Texas A&M University (1993) TR93-046.

[7] R. G. Downey and M. R. Fellows, "Fixed Parameter Tractability and Completeness I: Basic Theory," *SIAM Journal of Computing* **24** (1995) 873–921.

[8] R. G. Downey and M. R. Fellows, "Fixed Parameter tractibility and completeness II: Completeness for W[1]", *Theoretical Computer Science* **141** (1-2) (1995) 109–131.

[9] R. G. Downey, M. R. Fellows and Venkatesh Raman, "The Complexity of Irredundant Sets Parametrized by Size", Technical Report of the Institute of Mathematical Sciences IMSc-TR-97/06/24. submitted.

[10] M. R. Fellows, personal communication.

[11] M. R. Garey and D. S. Johnson, "Computers and Intractability, A Guide to the Theory of NP-Completeness", Freeman and Company (1979).

[12] D. J. Haglin and Shankar M. Venkatesan, "Approximation and Intractibility Results for the Maximum Cut Problem and its Variants", *IEEE Transactions on Computers* **40** (1) (1991) 110–113

[13] R. Kohli, R. Krishnamurti and P. Mirchandani, "The Minimum Satisfiability Problem", *SIAM J. Discrete Mathematics* **7** (2) (1984) 275–283.

[14] M. V. Marathe and S. S. Ravi, "On approximation algorithms for the minimum satisfiability problem", *Information Processing Letters* **58** (1996) 23–29.

[15] H. Nagamochi and T. Ibaraki, "A linear time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph", *Algorithmica* **7** (1992) 583–596.

[16] H. Nagamochi and T. Ibaraki, "Computing edge-connectivity in multigraphs and capacitated graphs", *SIAM Journal of Discrete Mathematics* **5** (1992) 54–66.

[17] C. H. Papadimitriou and M. Yannakakis, "On Limited Nondeterminism and the Complexity of the V-C Dimension", *Journal of Computer and System Sciences* **53**(2) (1996) 161-170. Preliminary version in *Proceedings of the 8th Structure in Complexity Theory Conference (1993)* 12–18.

[18] S. Poljak and D. Turzik, "A Polynomial Algorithm for Constructing a Large Bipartite Subgraph with an Application to a Satisfiability Problem", *Can. J. Math.*, Vol XXXIV No 3, (1982) 519–524.

[19] V. Raman, B. Ravikumar and S. Srinivasa Rao, "A Simplified NP-Complete MAXSAT problem", Technical Report of the Institute of Mathematical Sciences IMSc-TR-97/06/23. submitted.

[20] M. Stoer and F. Wagner, "A Simple Min Cut Algorithm", *Proceedings of the European Symposium on Algorithms*, Springer-Verlag LNCS **855** (1994) 141–147.

[21] M. Yannakakis, "On the Approximation of Maximum Satisfiability," *Journal of Algorithms* **17** (1994) 475–502.