Electronic Colloquium on Computational Complexity, Report No. 157 (2015)

Representation-Independent Fixed Parameter Tractability for Vertex Cover and Weighted Monotone Satisfiability

T. E. O'Neil

University of North Dakota
Grand Forks, ND 58202-9015, USA
oneil@cs.und.edu

Abstract. A symmetric representation for a set of objects requires the same amount of space for the set as for its complement. Complexity classifications that are based on the length of the input can depend on whether the representation is symmetric. In this article we describe a symmetric representation scheme for graphs and show that published parameterized algorithms for Vertex Cover do not provide a representation-independent proof that Vertex Cover is Fixed Parameter Tractable. In response to this challenge, a simple specialized backtracking algorithm is given for Vertex Cover that maintains $f(|y|)\cdot|x|$ complexity even if the input x is a symmetric representation of length $O((\lg n)^2)$ for a very sparse or very dense graph with n vertices. The algorithm is then generalized to solve the Weighted Monotone q-Satisfiability problem, constituting representation-independent proof that these two problems are in *FPT*.

Keywords: symmetric representation, Vertex Cover, Weighted Monotone q-Satisfiability, Fixed Parameter Tractable

1 A Classic Trio of Problems

Vertex Cover (VC), Clique (CL), and Independent Set (IS) comprise a classic trio of closely related graph problems. They lie at the core of the class of NP-Complete problems [4]. There are well-known, direct reductions between Vertex Cover and Independent Set, and between Independent Set and Clique, both involving the complement of an edge set or a vertex set [6]. The largest independent set of vertices in a graph is the complement of the smallest vertex cover, and the largest clique in a graph is the largest independent set of vertices in the complement graph. It is immediate that we can also define a direct reduction between Clique and Vertex Cover: the largest clique in a graph is the complement of the smallest vertex cover for the complement graph. While these three problems are very closely related, there are studies in both classical and parameterized complexity that find major complexity differences among them.

ISSN 1433-8092

1.1 The Power Index Distinction in Classical Complexity

We can define the power index of a problem to be the smallest value of i for which the problem has a deterministic algorithm that operates in time $2^{O(n^i)}$. The classical complexity literature contains a study that classifies Clique at power index 1/2 while Independent Set and Vertex Cover remain at power index 1 [9]. The analysis is based on a recursive backtracking algorithm for Clique and an inductive proof that the number of activations of the code is $2^{O(\sqrt{e})}$, where e is the size of the graph's edge set. If the graph is represented as an edge list x of length $O(e \lg n)$, the resulting complexity with respect to input length is $2^{O(\sqrt{|x|})}$. Independent Set and Vertex Cover remain at power index 1 because the reduction from either problem to Clique requires construction of the complement graph, which may have quadratically more edges that the original graph.

1.2 The Tractability Distinction in Parameterized Complexity

In classical complexity, the input for a decision algorithm is just the problem instance, and the bit length of the input is the parameter for the algorithm's time function. In parameterized complexity, the parameter is paired with the problem instance as part of the input to an algorithm. Using definitions from [5], a parameterized problem L is a set of pairs of strings (x, y) where x is the representation of a problem instance and y is some semantically meaningful complexity parameter. L is categorized as Fixed Parameter Tractable (FPT) if membership of (x, y) in L is decided by some algorithm with running time $f(|y|) \cdot |x|^{O(1)}$ for arbitrary function f. The idea is apparently that if deciding the problem becomes tractable when the parameter y is fixed, the algorithm might be well suited for certain application domains where the parameter y is bounded.

When FPT becomes the basis for a general complexity theory, the choice of the parameter is obviously very important. If the parameter is the length of the problem instance, every problem is trivially in FPT. So we find that all NP-complete problems are classified as tractable when the parameter is input length. Membership of a hard problem in FPT is more interesting if the parameter does not constitute an upper bound on the length of the problem instance. In that case, an algorithm exists that will make the problem tractable with a fixed parameter, even for arbitrarily large instances.

The distinction among VC/CL/IS in the literature of parameterized complexity is much different from the one found in classical complexity: it separates Vertex Cover, rather than Clique, from the other two problems. It also postulates a larger complexity gap. The power index distinction between Clique and IS contrasts strongly sub-exponential time $2^{O(\sqrt{|x|})}$ with exponential time $2^{O(|x|)}$, where x is the representation of the input graph. In parameterized complexity, the distinction is tractable vs. intractable: the Vertex Cover problem is classified as Fixed Parameter Tractable, while CL/IS remain in a higher class of problems considered to be intractable.

Vertex Cover is classified as FPT in [5] when the parameter is the size k of the cover, while k-Clique and k-Independent Set are in the presumably higher complexity class $W[1]$, which also contains weighted q-CNF Satisfiability for all q. This distinction is based on algorithms such as those described in [1], [2], and [3], with complexities of $O(kn + 2^k k^{2k+2})$, $O(kn + 1.2745^k k^4)$, and $O(kn + 1.2738^k)$ respectively. Each of these complexities has the form $O(kn+f(k))$, thus satisfying the FPT definition under the assumption that input length is $\Omega(n)$. No algorithms have been discovered for k-Clique or IS of size k with time $2^{O(k)}n^{O(1)}$, so these problems remain in $W[1]$. In fact, they are classified as $W[1]$-complete.

The classifications of parameterized complexity theory are, by definition, parameter-specific. VC/CL/IS are closely tied by logical duality, so we might expect them to show similar complexities for corresponding parameters. Clique is the dual image of Independent Set with respect to the edge set; Vertex Cover is the dual image of Independent Set with respect to the vertex set; and Vertex Cover is the dual image of Clique with respect to both the vertex set and the edge set. If we consider the parameter $d = n - k$, the number of vertices that are not in the solution, we would expect to find that k-Clique and k-IS are in FPT for the parameter d, and that k-Vertex Cover is $W[1]$-complete.

Putting comments about parameter-dependent distinctions aside, however, the goal of this study is to determine whether the classification of Vertex Cover as FPT is representation dependent. In the sections below we examine whether the classification is valid for all regions of the problem space when space-efficient symmetric representation is used for the graph.

2 Symmetric Representation

Symmetric representation is described as a tool for complexity analysis in [7] and [8]. We provide an overview of that discussion here. Symmetric representation can have an impact on any complexity classification where input length plays a role in the analysis. This includes both classical and parameterized complexity theory.

The list of elements is a natural representation for a set of objects. If the set is nearly full with reference to some universal set, however, it may be more efficient to maintain a list of elements that are not in the set. We will call a representation scheme r for subsets S of a universal set U symmetric if the representation of a set is the same length as the representation of the set's complement, i.e. $|r(S)| = |r(\bar{S})|$.

2.1 Flip List Representation

A flip list is an example of a variable-length symmetric representation scheme. It stores the list of elements in a set or the list of elements in a set's complement – whichever is shorter. It has a polarity to indicate how it is to be interpreted. We note that the variable-length symmetric schemes are space efficient. A flip list never exceeds half the length of a conventional list. The basic operations on flip

lists are designed to take time proportional to the length of the representation, which may be significantly smaller than the actual size of the set. In order to accomplish this, the universe must be enumerable.

Definition 1. *An enumerable universe is an ordered set of elements in which each element can be represented as a string of symbols over some finite alphabet. It provides the operations listed below. The notation $|e|$ is understood to refer to the length of the string representation for element e.*

first() *returns the first element e_0 of the set in time $O(|e_0|)$.*
last() *returns the last element $e_{\max}$ of the set in time $O(|e_{\max}|)$ (needed only for a finite universe).*
successor(e) *returns the successor of element e in time $O(|e|)$.*
predecessor(e) *returns the predecessor of element e in time $O(|e|)$.*

The flip list scheme implements a subset of an enumerable universe.

Definition 2. *A flip list $F^{\pm} = (\mu, c, l, p)$ is a list representation for a finite subset S of a finite enumerable universe U with the following components and properties:*

A meta-data string μ *which defines a finite enumerable universe U of possible elements. An alphabet and maximum string length would be adequate for a set of strings, or a maximum value for a set of natural numbers. The length of the meta-data string is constrained to be $O(|e|)$ where e is the largest single element of U.*
A count c *of the number of elements in the set. $c = |S|$.*
An iterable list of elements l *whose length does not exceed half the size of the universal set. $|l| \leq |U|/2$.*
A polarity p *in $\{+, -\}$. Positive polarity indicates the list l contains the elements in S. Negative polarity indicates the list l contains elements in $\bar{S}$.*

As a notational convention, $F^{\pm}$ indicates a flip list whose polarity is unspecified, F^{+} indicates a flip list known to be positive, and F^{-} indicates a flip list known to be negative.

The standard operations on the flip list are implemented in time $O(|l| \cdot |e_{\max}|)$ (where $|l|$ may be smaller than $|S|$ or $|U|$). Search, insertion, and deletion are implemented by traversal of $F^{\pm}.l$. For negative lists, the search result is negated, insertion is implemented as deletion, and vice versa. When the length of the list l becomes greater than $|U|/2$ during insertion or deletion of an element, l is rebuilt in a traversal of the universal set U. This operation takes time $O(|U| \cdot |e_{\max}|)$, which in this case is also $O(|l| \cdot |e_{\max}|)$. We can also build an iterator for the set S that can be used to enumerate the elements of S in time $O(|S| \cdot |e_{\max}|)$. This is implemented by iterating l if the list is positive, or by iterating U and l simultaneously if the list is negative. In either case, an amortized analysis of the enumeration shows that each element is retrieved in $O(|e_{\max}|)$ time. When the list is negative, $|S| > |U|/2$, so $|U|$ is $\Theta(|S|)$. Finally, we note that the complement operation is achieved in constant time by flipping the polarity of the list, and that the cost of converting a conventional list to a flip list is at most linear.

2.2 A Compound Flip List Scheme for Graphs

A graph $G = (V, E)$ can be implemented as a pair of flip lists with a single universe. We can number the vertices $1, 2, \ldots, n$ and use this set as the universe for vertices. One flip list indicates which vertices are in the graph. It is initially full – with negative polarity and an empty element list, indicating that there are no missing vertices. The other flip list keeps track of the edges, which are vertex pairs. The implicit universe for the edges is $V \times V$, and the successor function for the edge universe relies on the successor function for the vertex universe. For a graph with n vertices, the storage requirement for a complete graph or a graph with no edges is $O(\lg n)$. If the number of edges or the number of missing edges is $O(\lg n)$, then the size of the representation is $O((\lg n)^2)$. The maximum space requirement for a graph is $\Theta(n^2 \lg n)$.

3 Applying Symmetric Representation to Vertex Cover

The complexity distinction between Clique and IS/VC in classical complexity theory disappears when a flip list is used to represent a graph's edge list. With flip lists, the representations of the edge set and its complement have exactly the same length, and the complement operation requires only constant time. This argument is fully developed in [8]. Using symmetric representation, Clique, IS, and VC all have power index 1. Any algorithm for one of the problems can be used for the other two without additional cost. When the complexity parameter is input length, all three problems have the same worst-case time complexity.

3.1 Using Symmetric Representation with Previous Algorithms

We now consider whether the parameterized complexity distinctions for Vertex Cover/Clique/Independent Set survive the test of symmetric representation. Assume we have a graph $G = (V, E)$ with $|V| = n$ and $|E| = e$. When the length of the graph representation is $\Omega(n)$, any of the algorithms from [3, 1, 2] is sufficient to establish that Vertex Cover is in FPT for parameter k. Under symmetric representation, however, there are regions of the problem space where the graph representation will have length $O((\lg n)^2)$. This happens when either $|E|$ is $O(\lg n)$ or $|\bar{E}|$ is $O(\lg n)$. This is problematic for the algorithms cited above, whose time functions have a factor n. To meet the FPT requirement, the running time would need to be $f(k) \cdot |x|^{O(1)}$ where x is the representation of the problem instance. If $|x|$ is $O((\lg n)^2)$, then a factor of n in the running time is $O(2^{\sqrt{|x|}})$, not $|x|^{O(1)}$. Thus, it appears that the algorithms cited above provide only representation-dependent evidence that Vertex Cover is in FPT.

3.2 A Backtracking Algorithm Adapted to Symmetric Representation

Representation-independent classification of Vertex Cover in FPT thus depends on our ability to design a decision algorithm that operates in time $f(k) \cdot ((\lg n)^2)^{O(1)} =$

```
Algorithm kCover(V±, E±, k) // Given flip lists for a non-empty vertex
                            // set and edge set for graph G = (V, E),
                            // return whether G has a vertex cover of size k.
1)  if k < 0 or k > |V|                          // k out of bounds
2)      return false
3)  else if k ≥ |E| or E± is positive and empty // few or no edges
4)      return true
5)  else if E± is negative and empty             // no missing edges
6)      return k ≥ |V| − 1
7)  else if k < |V| − ⌈√(|V|(|V| − 1) − 2|E|)⌉
8)      return false
9)  q := a vertex from V
10) d := degree(q, E±)
    // Compute a smallest cover with q:
11) V1± := V± − {q}
12) E1± := E± − {(v, w)| v = q or w = q}         // restrict E± to V± − {q}
13) if kCover(V1±, E1±, k − 1)
14)     return true
15) else if k < d
16)     return false
    // Compute a smallest cover without q:
17) else
18)     N := neighbors(q, V±, E±)
19)     V2± := V1± − N
20)     E2± := E1± − {(v, w)| v ∈ N or w ∈ N} // remove edges covered by N
21)     return kCover(V2±, E2±, k − d)
```

Fig. 1. A backtracking algorithm for Vertex Cover, adapted to flip lists

$f(k) \cdot (\lg n)^{O(1)}$ for symmetrically represented instances with $O(\lg n)$ edges or all but $O(\lg n)$ edges. We offer the *kCover* algorithm in Figure 1 as a candidate. It operates on the principle that if a vertex q is not in the cover, then all of q's neighbors must be in the cover. First we observe that the number of recursive activations of *kCover* cannot exceed 2^k. There are two recursive calls for each activation of the procedure, with a smaller value for the parameter k in each call. So the depth of recursion cannot exceed k, and every node in the activation tree has at most two children.

To complete the analysis, we must confirm that no operation in the code exceeds time proportional to the length of the representation. This poses a potential problem when either $|E|$ or $|\bar{E}|$ is $O(\lg n)$, implying that $|x|$ is $O((\lg n)^2)$. If $|E|$ is $O(\lg n)$, for example, the graph is very sparse and most nodes have degree 0. Such nodes will not be needed for any smallest vertex cover. The technical challenge is to eliminate these nodes without explicitly processing them. We need to verify that no operation within a single activation will require $\Omega(n)$ time.

Tests based on k, $|V|$, $|E|$ (lines 1-8) The counts $|V|$ and $|E|$ are explicitly stored as fields in the flip list representation, so the relational and arithmetic expressions can be calculated in time $\lg(\max(|V|, |E|) = O(\lg n)$.

Choose a vertex q (line 9) A vertex is chosen by accessing the first edge from the edge list $E^{\pm}$ and selecting either of its endpoints. This operation is $O(\lg n)$ for a positive edge list and $O(|E^{-}.l| \cdot \lg n)$ for a negative edge list.

Compute the degree of q (line 10) If the list of edges is positive, traverse E^{+}, counting occurrences of q. If the edge list is negative, traverse E^{-} incrementing a counter d for each occurrence of q, and return $n-d$. $O(|E^{\pm}.l| \cdot \lg n)$.

Remove q from V (line 11) The deletion of q from the negative list V^{-} is implemented as the insertion of q into $V^{-}.l$. The length of $V^{-}.l$ cannot exceed the depth of recursion k, so the operation count is $O(k \lg n)$.

Remove q's edges from E (line 12) Whether the edge list is positive or negative, this step is accomplished by traversing the list and removing edges that mention q, which is no longer in the graph. $O(|E^{\pm}.l| \cdot \lg n)$.

Remove all neighbors of q from V_1 (lines 18-19) Following the execution of line 15, it must be true that $d \leq k$, and the length of V_1^{-} also does not exceed k. $O(k^2(\lg n))$.

Remove all edges of q's neighbors from E_1 (line 20) Since both vertices and edges are being removed for neighbors of q, the edge list E_1 will be shortened whether it is positive or negative, and $|E_1| < |E|$. Since there are no more than k neighbors the total operation count is $O(k \cdot |E^{\pm}.l| \cdot \lg n)$.

In examining the operation counts for each line of the code as described above, we observe that the number of operations per recursive activation of *kCover* is $O(k \lg n \cdot \max(k, |E^{\pm}.l|))$. Line 3 eliminates cases where $k \geq |E|$, but there are non-trivial instances with $k \geq |\bar{E}|$. So we consider both $k \geq |E^{\pm}.l|$ and $k < |E^{\pm}.l|$, deriving total operation counts of $k^2 2^k \lg n$ and $k \cdot 2^k \cdot |E^{\pm}.l| \lg n$ respectively. Since $|x| = O(|E^{\pm}.l| \cdot \lg n)$ both possible complexities satisfy the definition of *FPT*.

4 The Weighted Monotone Satisfiability Problem

The Weighted Monotone q-Satisfiability problem is the same as the Weighted q-Satisfiability problem, except that all literals in the Boolean expression are positive. The input to the problem is a monotone Boolean expression in conjunctive normal form and a parameter k. The expression is a conjunction of clauses over n variables, and each clause in the expression is a disjunction of exactly q distinct variable names. The task is to determine whether all clauses in the expression can be satisfied by assigning the value *true* to at most k variables. The Weighted q-Satisfiability problem is categorized as $W[1]$-complete in [5], but the monotone version of the problem is not classified.

Weighted Monotone q-Satisfiability is a generalization of Vertex Cover, and the algorithm of Figure 1 can be both generalized and simplified to solve it. We can also generalize the flip list of section 2.2 to represent a Boolean expression.

```
Algorithm MonoSat(V±, C±, k) // Given flip lists for a non-empty
                             // variable set V and a set of clauses C
                             // for a CNF Boolean expression B,
                             // return whether B is satisfied by a
                             // truth assignment with at most
                             // k true variables.
1)  if k < 0 or k > |V|                          // k out of bounds
2)      return false
3)  else if k ≥ |C| or C± is positive and empty // few or no clauses
4)      return true
5)  else if C± is negative and empty             // no missing clauses
6)      return k ≥ |V| − q + 1
7)  c := a clause from C
8)  for each variable x in c
9)      V1± := V± − {x}
10)     C1± := C± − {c ∈ C± | c contains x}      // remove satisfied clauses
11)     if MonoSat(V1±, C1±, k − 1)
12)         return true
13) return false
```

Fig. 2. An algorithm for Monotone Satisfiability, adapted to flip lists

We define a Boolean expression $B = (V, C)$ to be a set of clauses C over a set of variables V. The variables are numbered $1, 2, \ldots, n$ and stored in flip list $v^{\pm}$. The clauses are q-tuples of variable numbers stored in the flip list $C^{\pm}$. The variable set defines the universe for iteration of a negative clause list. The variable list is initially full, with negative polarity and an empty element list. For a q-CNF expression with n variables, the storage requirement for an expression with no clauses or no missing clauses is $O(\lg n)$. If the number of clauses or missing clauses is $O(\lg n)$, then the size of the representation is $O((\lg n)^2)$. The maximum space requirement for an expression is $\Theta(n^q \lg n)$.

The *MonoSat* algorithm is shown in Figure 2. It operates on the simple principle that for each clause, at least one of its variables must be assigned *true*. It selects a clause and makes q recursive calls, one for each variable in the clause. The parameter k is decremented to $k - 1$ for the recursive calls (line 11), and the code returns *false* if k becomes negative (lines 1-2). The computation can therefore be modeled as a tree with degree q and height k, and it is immediately apparent that the total number of recursive code activations cannot exceed q^k, which is a function of k when q is constant. As with the Vertex Cover algorithm, it remains to show that no operation in the code exceeds time proportional to the length of the representation.

Tests based on k, $|V|$, $|C|$ (lines 1-6) The counts $|V|$ and $|C|$ are stored as fields in the flip list representation, so the relational and arithmetic expressions can be calculated in time $\lg(\max(|V|, |C|) = O(\lg n)$.

Choose a clause c (line 7) A clause is chosen by accessing the first clause from the clause list $C^{\pm}$. This operation is $O(\lg n)$ for a positive clause list and $O(|C^{-}.l| \cdot \lg n)$ for a negative clause list.

Loop iterations (lines 8-12) The body of the loop executes at most q times.

Remove x from V (line 9) The deletion of x from the negative list V^{-} is implemented as the insertion of x into $V^{-}.l$. The length of $V^{-}.l$ cannot exceed the depth of recursion k, so the operation count is $O(k \lg n)$.

Remove x's clauses from C (line 10) Whether the clause list is positive or negative, this step is accomplished by traversing the list and removing clauses that mention x, which has been removed from the expression. $O(|C^{\pm}.l| \cdot \lg n)$.

The sum of the operation counts, as specified above, for a single activation $MonoSat$ is $O(\lg n \cdot \max(k, |C^{\pm}.l|))$. Considering both $k \geq |C^{\pm}.l|$ and $k < |C^{\pm}.l|$, we derive total operation counts of $k2^k \lg n$ and $2^k \cdot |C^{\pm}.l| \lg n$ respectively. Since $|x| = O(|C^{\pm}.l| \cdot \lg n)$ both complexities satisfy the definition of FPT.

5 Conclusion

While parameterized complexity is less sensitive to input representation than classical complexity, some parameterized complexity classifications are representation dependent as well as parameter dependent. Symmetric representation for very dense or very sparse problem instances can have length $O((\lg n)^2)$, and algorithms whose time functions have a factor of $n^{O(1)}$ may not be sufficient to establish that a problem is in FPT. The Vertex Cover problem provides an example of this phenomenon. It is possible, however, to design algorithms that have complexity $f(|y|) \cdot |x|^{O(k)}$ for both Vertex Cover and Weighted Monotone q-Satisfiability, even when $|x|$ is $O(\lg n)$. These algorithms provide representation-independent evidence that both problems belong in FPT when the parameters are cover size and the number of *true* variables, respectively.

References

1. J. Buss and J. Goldsmith, Nondeterminism within P, *SIAM Journal on Computing* **22** (1993), pp. 560-572.
2. L. Chandran and F. Grandoni, Refined Memoisation for Vertex Cover, *Lecture Notes in Computer Science* **3162** (2004), pp. 61-70.
3. J. Chen, I. A. Kanj, and G. Xia, Improved Parameterized Upper Bounds for Vertex Cover, *Proceedings of the 31st International Symposium on the Mathematical Foundations of Computer Science (MFCS 2006)*, (2006) pp. 238–249.
4. M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, (Freeman Press, San Francisco, CA, 1979).
5. R.G. Downey and M.R. Fellows, Fixed-parameter Tractability and Completeness I: Basic Results, *SIAM Journal on Computing* **24** (1995), pp. 873-921.
6. R. Karp, Reducibility Among Combinatorial Problems, in *Complexity and Computer Computations*, eds. R. E. Miller and J. W. Thatcher (Plenum Press, New York, 1972), pp. 85–103.

7. T. E. O'Neil, The Importance of Symmetric Representation, *Proceedings of the 2009 International Conference on Foundations of Computer Science* (CSREA Press, 2009), pp. 115–119.
8. T. E. O'Neil, Complement, Complexity, and Symmetric Representation, accepted for publication, *International Journal of Foundations of Computer Science* (World Scientific, January 2015).
9. R. Stearns and H. Hunt, Power Indices and Easier Hard Problems, *Mathematical Systems Theory* **23** (1990), pp. 209–225.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de