On the Computational Complexity of Software (Re)Modularization: Elaborations and Opportunities

Todd Wareham
Department of Computer Science
Memorial University of Newfoundland
St. John's, NL Canada
harold@mun.ca

ABSTRACT

Software system modularization and remodularization are key challenges in software engineering. All previous research has assumed that these problems are computationally intractable and hence focused on heuristic methods such as hill-climbing, evolutionary algorithms, and simulated annealing that are fast but do not guarantee to produce solutions of optimal or even near-optimal quality. However, this intractability has never been formally established. In this paper, we give the first proofs of the NP-hardness of software modularization and remodularization relative to several models of module-internal connectivity. We also review three popular algorithmic approaches for producing provably optimal or near-optimal solutions efficiently and both discuss the applicability of these approaches in and list results in the literature relevant to practical software modularization and remodularization.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering models*; D.2.2 [**Software Engineering**]: Design Tools and Techniques; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical algorithms and problems—*computations on discrete structures*

General Terms

Algorithms, Design, Measurement, Performance, Theory

Keywords

software (re)modularization, computational complexity, parameterized computational complexity

1. INTRODUCTION

The modularization of software systems, *i.e.*, the assignment of software-units to modules that are minimally coupled and maximally cohesive, is a central challenge in software system design; of equal importance later on in software maintenance is the remodularization of such systems to re-optimize cohesion and coupling as these systems are modified in response to changing requirements. Over the last 30 years, much effort has been put into developing automated techniques to aid in modularizing and remodularizing software systems.

Much of this work has assumed that these problems are computationally intractable and hence cannot be solved both efficiently and optimally. This assumption is based on the similarity of modularization to intractable problems like graph partitioning [17] and the very large number of possible modularizations of a software system [20, page 194]. Given this assumed but unproven intractability, research has focused on heuristic methods such as hill-climbing [20], evolutionary algorithms (see [13, Sections 7.1 and 7.2] and references), and simulated annealing [20, 27] that are fast but are not guaranteed to produce solutions of optimal or even near-optimal quality. However, this may be unnecessary — even if modularization and remodularization are intractable, there may yet be fast methods that give provably optimal or near-optimal solutions in practice.

In this paper, we address the issues raised above by (1) giving the first proofs of the computational intractability of software modularization and remodularization and (2) reviewing existing approaches for efficiently obtaining provably optimal or near-optimal modularizations and remodularizations and the restrictions (if any) under which these approaches operate. We will focus in particular on the promise of fixed-parameter tractable algorithms [7, 8] whose runtimes are exponential in general but may run in effectively polynomial time on inputs that occur in practice.

This paper is organized as follows. In Section 2, we formalize the problems of software system modularization and remodularization. Section 3 demonstrates the intractability of these problems in general relative to both deterministic and probabilistic algorithms. Section 4 reviews efficient algorithmic approaches for modularization and remodularization whose performance is provably optimal or near-optimal as well as existing results relative to those approaches. In order to focus in the main text on the implications of our results and reviewed approaches for (re)modularization research, all proofs of results are given in Appendix B. Finally, our conclusions are given in Section 5.

BICT 2015, December 03-05, New York City, United States

DOI 10.4108/eai.3-12-2015.2262449

2. FORMALIZING SOFTWARE (RE)MODULARIZATION

To formalize software modularization and remodularization, we need to formalize the following four entities:

1. **Software systems**: Following [20], a software system is represented as a Unit Dependency Graph (UDG) $G = (V, E)$ in which vertices are software units, *e.g.*, procedures, classes, data types, and edges are dependencies between pairs of units, *e.g.*, inheritance, procedure invocation, data-type use. Though dependency graphs are usually directed to allow for the directionality of dependencies, this directionality affects neither the definitions of software system coupling and cohesion nor the manner in which dependent units are assigned to the same module during (re)modularization. Hence, we will here ignore this directionality and consider only undirected UDG.

2. **software system modularization**: A k-modularization is a partition of a UDG G into k disjoint connected components[1]; this will be formalized as a function $M : V \Rightarrow \{1, 2, \ldots, k\}$ such that $M(u)$ is the number of the module to which unit u is assigned.

3. **modularization quality measure**: There are many measures of modularization quality in the literature, *e.g.*, MQ [20, Section 2.3], Q [27, Section 2.4]. However, all such measures are ultimately based on the concepts of coupling (the degree of dependency between modules) and cohesion (the extent to which the units in a module depend on each other and hence belong together in that module). Coupling will be formalized as the number of dependencies in UDG $C = (V, E)$ between distinct modules under M, *i.e.*, $F_{cpl}(C, M) = |\{(u, v) \mid (u, v) \in E \text{ and } \mathrm{M(u)} \neq \mathrm{M(v)}\}|$ and cohesion will be formalized in a complementary fashion as the number of dependencies between units in C in the same module under M, *i.e.*, $F_{coh}(C, M) = |\{(u, v) \mid (u, v) \in E \text{ and } \mathrm{M(u)} = \mathrm{M(v)}\}|$. As each edge in a modularized UDG is either between modules or inside a module, $F_{cpl}(C, M) + F_{coh}(C, M) = |E|$, and by explicitly minimizing coupling (maximizing coherence), one also implicitly maximizes coherence (minimizes coupling).

   We will further refine the standard definition of cohesion given above by considering three degrees of connectivity via dependencies among the units in each module of a modularization M:

   (a) **Basic Connectivity (BC)**: There is a dependency–path in the module between every pair of units.

   (b) **High Connectivity (HC)**: There is a dependency–path of length at most 2 in the module between every pair of units. This is equivalent to each unit in a module with n units having at least $n/2$ dependencies to other units in that module.

   (c) **Complete Connectivity (CC)**: There is a dependency in the module between every pair of units.

4. **remodularization modifications**: A remodularization changes the module-assignments of one or more units, where each such change is a move refactoring [19, 27]. To allow control over the number of changes in a remodularization (in order to, for example, ensure that the remodularization respects the original modularization as much as possible [1]), we will define the number of module-assignment changes between two modularizations M and M' of a UDG $C = (V, E)$ as $D(C, M, M') = |\{v \mid v \in V \text{ and } \mathrm{M(v)} \neq \mathrm{M'(v)}\}$.

[1] In this paper, the term "component" will be used in its graph-theoretic sense to denote a set of vertices in a graph G that is not connected by edges to any other vertex in G.

The above yields the following problems:

Software Modularization with Connectivity **X**
(SMod-**X** where **X** ∈ {BC,HC,CC})
Input: UDG $C = (V, E)$, integers $k, s > 0$.
Output: A k-modularization M of C with connectivity **X** and $F_{cpl}(C, M) \leq s$, if such an M exists, and special symbol $\perp$ otherwise.

Software Remodularization with Connectivity **X**
(SReMod-**X** where **X** ∈ {BC,HC,CC})
Input: UDG $C = (V, E)$, a k_I-modularization M of C such that $F_{cpl}(C, M) = s$, integers $k_F, c_s, c_m > 0$.
Output: A k_F-modularization M' of C with connectivity **X** such that $F_{cpl}(C, M') \leq F_{cpl}(C, M) - c_s$ and $D(C, M, M') \leq c_m$, if such an M' exists, and special symbol $\perp$ otherwise.

Note that in assessing the quality of a remodularization, we require only that this remodularization improve by at least a specified amount c_s relative to the quality of the given modularization. In combination with the control over the number of remodularization changes granted by parameter c_m, this allows assessment of the computational difficulty of incremental schemes for remodularization [2, 27]

The above is already useful, in that it establishes that software modularization is not, as it is sometimes construed in the literature [17], the problem Graph Partitioning [12, Problem ND14]. This is so because Graph Partitioning incorporates the additional constraint that all components in a modularization have a specified maximum size. Rather, modularization with basic and complete connectivity corresponds to the following problems:

k-way Cut (kWC)
Input: An undirected graph $G = (V, E)$, integers $k, s > 0$.
Question: Is there a subset $E' \subseteq E$, $|E'| \leq s$, such that the graph G' created by removing E' from G consists of k connected components?

k-Cluster Deletion (kCD)
Input: An undirected graph $G = (V, E)$, integers $k, s > 0$.
Question: Is there a subset $E' \subseteq E$, $|E'| \leq s$, such that the graph G' created by removing E' from G consists of k cliques[2]?

[2] A clique is a graph G in which there is an edge between each pair of vertices in G.

The connected components and cliques in kWC and kCD are equivalent to the modules in SMod-BC and SMod-CC in both number and connectivity-type, and sets E' of edges in both kWC and kCD are equivalent to the sets of coupling-dependencies counted by $F_{cpl}()$ in SMod-BC and SMod-CC. This gives us the following:

Observation 1. Problems k-WAY CUT and SMod-BC are equivalent.

Observation 2. Problems k-CLUSTER DELETION and SMod-CC are equivalent.

As we shall see below, these observations will be most useful both in applying known results to our problems and in proving new results.

3. SOFTWARE (RE)MODULARIZATION IS INTRACTABLE

Following general practice in Computer Science [12], we define tractability as being solvable in the worst case in time polynomially bounded in the input size. We show that a problem is not polynomial-time solvable, *i.e.*, not in the class P of polynomial-time solvable problems, by proving it to be at least as difficult as the hardest problems in problem-class NP (see [12] and Appendix A for details).

Result A: SMod-BC, SMod-HC, SMod-CC, SReMod-BC, SReMod-HC, and SReMod-CC are NP-hard.

Modulo the conjecture $P \neq NP$ which is widely believed to be true [11], the above shows that software modularization and remodularization cannot be done optimally in polynomial time in general.

The NP-hardness of SMod-CC and SReMod-CC is perhaps unsurprising given the computational intractability of finding cliques of a specified size in a given graph [12, CLIQUE, Problem GT19] (indeed, this is reflected in the fact that the NP-hardness of SMod-CC and SReMod-CC holds when the numbers of requested modules k and k_F, respectively, have any fixed value greater than or equal to three). As clique-modules have the highest possible number of module-internal dependencies, this implies that finding modularizations and remodularizations with the highest possible values of MQ [20] and Q [27] is intractable; as our problems remain NP-hard relative to basic connectivity, this intractability continues to hold for much lower values of MQ and Q.

Result A has very interesting additional consequences. It is widely believed that $P = BPP$ [25, Section 5.2] where BPP is considered the most inclusive class of problems that can be efficiently solved using probabilistic methods (in particular, methods whose probability of correctness can be efficiently boosted to be arbitrarily close to probability one). Hence, our results also imply that unless $P = NP$, there are no probabilistic polynomial-time methods which correctly modularize or remodularize software systems with high probability for all inputs. Taken together, the above constitutes the first proof that *no* currently-used method (including those based on hill-climbing [20], evolutionary algorithms (see [13, Sections 7.2 and 7.2] and references), or simulated annealing [20, 27]) can guarantee both efficient and correct operation for all inputs for these problems.

4. PROVABLY (NEAR-)OPTIMAL ALGORITHMIC OPTIONS FOR SOFTWARE (RE)MODULARIZATION

Though the intractability results in Section 3 are elegant and powerful, the inconvenient fact remains that we would still like to perform software system modularization and remodularization in an efficient and reliable manner. In this section, we will consider three options for accomplishing this — namely, polynomial-time approximation algorithms, (Section 4.1) restricted-case polynomial-time algorithms (Section 4.2), and fixed-parameter tractable algorithms (Section 4.3) — and discuss their applicability in practical software modularization and remodularization (Section 4.4).

4.1 Poly-time Approximation Algorithms

A polynomial-time approximation algorithm is an algorithm that gives a solution whose value relative to a particular parameter (such as solution quality, *e.g.*, parameter s in problem SMod-BC) is provably within an additive or multiplicative factor of the value of that parameter in an optimal solution [6]. Very few problems have polynomial-time additive-factor approximation algorithms. Given this, the desirable situation is a polynomial-time $(1 + \epsilon)$-multiplicative factor algorithm, where ϵ is very small, *e.g.*, $\frac{1}{10}$.

At present, there is only one such result for our problems.

Result B: SMod-BC can be approximated in polynomial time to a multiplicative factor of $2 - \frac{c^2}{k^2}$ for some constant c (follows from Observation 1 and [26, Theorem 2.1]).

One cannot set $c = k$ to get a polynomial-time optimal solution algorithm as the algorithm runtime increases dramatically as the value of c increases, yielding an exponential runtime as c approaches the value of k. Moreover, though this algorithm may produce solutions close in value to optimal on some inputs, it will produce solutions whose value is effectively twice that of optimal on others, and there is no way to tell which situation holds for any given input; hence, it is not useful in practical modularization.

4.2 Restricted-case Poly-time Algorithms

Many problems have algorithms whose runtimes are non-polynomial in general, *e.g.*, $O(n^k)$ for problem-parameters n and k, but polynomial if one or more parameter have their values fixed to constants, *e.g.*, if $k = c$ for some constant c, $O(n^k) \Rightarrow O(n^c)$. Such restricted-case polynomial-time algorithms are practical if the parameters k and n are of very small and moderate value, respectively, in inputs encountered in practice.

At present, there are several such results for our problems.

Result C: SMod-BC can be solved in $O(|V|^{2k})$ time (follows from Observation 1 and [23]).

Running times can often be improved for small constant values of a parameter (for an overview of such algorithms for SMod-BC when $k \leq 6$, see [26, Section 1]). Such algorithms may indeed be useful in the case of modularizing systems that are small (and hence have both few units and few modules) or as part of an iterative strategy that mod-

ularizes small portions of a larger system. However, they are unlikely to be practical for one-shot modularizations of systems consisting of a large number of units, even if the number of requested modules is small.

4.3 Fixed-parameter Tractable Algorithms

As the problem with restricted-case polynomial-time algorithms noted above is that n may be very large even if k is very small, it would seem reasonable to relax the requirement that an algorithm's runtime be polynomial in all of its parameters. This insight underlies the theory of parameterized computational complexity [8]. It turns out that a number of NP-hard problems have been successfully solved by algorithms whose runtimes are polynomial in the overall input size and non-polynomial in parameters whose values are small in the inputs encountered in practice (see [8, 22] and references). The following states this insight formally.

Definition 1. Let Π be a problem with parameters $k_1, k_2, \ldots$. Then Π is said to be **fixed-parameter (fp-) tractable** for parameter-set $K = \{k_1, k_2, ...\}$ if there exists at least one algorithm that solves Π for any input of size n in time $f(k_1, k_2, ...)n^c$, where $f(\cdot)$ is an arbitrary function and c is a constant. If no such algorithm exists then Π is said to be **fixed-parameter (fp-) intractable** for parameter-set K.

In other words, a problem Π is fp-tractable for a parameter-set K if all superpolynomial-time complexity inherent in solving Π can be confined to the parameters in K.

There are many techniques for designing fp-tractable algorithms [7, 8], and fp-intractability is established in a manner analogous to polynomial-time intractability by proving a parameterized problem is at least as difficult as the hardest problems in one of the problem-classes in the W-hierarchy $\{W[1], W[2], ...\}$ (see [8] and Appendix A for details). At present, there are several such results for our problems.

Result D: SMod-BC is fp-intractable for $\{k\}$ (follows from Observation 1 and [9, Theorem 1]).

Result E: SMod-CC is fp-intractable for $\{k\}$ (follows from Observation 2, [21, Theorem 14], and [24, Lemma 2.1.35]).

Result F: SReMod-CC is fp-intractable for $\{k_F\}$ (follows from Lemma 4 in Appendix B, [21, Theorem 14], and [24, Lemma 2.1.35]).

Result G: SMod-BC is fp-tractable for $\{s\}$ (follows from Observation 1 and [15, Theorem 1]).

The situation is better than it may first appear as a problem that is fixed-parameter for a parameter-set K is also fp-tractable for any parameter-set K' that is a superset of K [24, Lemma 2.1.30] and the runtimes of algorithms derived relative to K' are often much better than those derived relative to K. Such additional restrictions will often even banish fp-intractability.

Result H: SMod-BC is fp-tractable for $\{k\}$ when the average of the vertex-degrees in the given UDG is a constant (follows from Observation 1 and [15, Corollary 2]).

Result I: SMod-BC is fp-tractable for $\{k\}$ when the given UDG is planar (follows from Observation 1 and [15, Proposition 3]).

Given that the running times of these algorithms are (to be blunt, ludicrously) impractical and k and s may only be small in instances that are easily modularizable by humans, the results above are not immediately useful for real-world software modularization. However, such impracticalities are typical of the initial fp-algorithms derived relative to a parameter or parameter-set. Experience has shown that once fp-tractability is proven, surprisingly effective fp-algorithms are often subsequently developed, sometimes by incorporating parameters that were not considered in the original analysis (see [7, 8] and references). Hence, the results given here should be seen as promissory notes on algorithms that will be developed in future, possibly within a research program like that sketched in the next section.

4.4 Discussion

Though none of the three approaches reviewed above have results that are immediately useful in practical modularization and remodularization, all three are potentially of use. The most promising of these is fixed-parameter tractable algorithms. Such fp-algorithms are ideal for exploiting restrictions characterizing instances of modularization and remodularization that occur in practice, *e.g.*, incremental remodularization in which the requested degree of modularization quality (c_s) and structural (c_m) change are both small [2, 27]. Moreover, the relaxed tractability encoded in the fixed-parameter approach may yield improvements when combined with other types of algorithms, *e.g.*, fixed-parameter approximation [18], hill-climbing [10], and evolutionary [16] algorithms.

The best way to start a fixed-parameter (re)modularization research program would be to characterize the UDG that underlie actual software systems with an eye to finding both parameters whose values are small in practice as well as the most restricted types of graphs that encode UDG encountered in practice. The specific situations in which modularization and remodularization are done in practice, *e.g.*, incremental (re)modularization, should also be scrutinized to look for additional parameters of small value. Such parameters and graph-types would then guide the derivation of useful fp-(in)tractability results relative to (if necessary, reformulations of) the modularization and remodularization problems defined in Section 2. This derivation process may benefit from both the results listed above and results for closely related problems, *e.g.*, fp-tractability results for Cluster Deletion [4, 5] and Highly Connected Deletion [14] and algorithms for Graph Partition [3].

5. CONCLUSIONS

We have presented a formal characterization of the problems of software system modularization remodularization relative to several types of module-internal connectivity and given the first proofs that all of these problems are computationally intractable in general. This intractability makes unlikely the existence of polynomial-time deterministic or probabilistic methods that produce optimal solutions for these problems. We have also reviewed several algorithmic options for producing optimal or near-optimal solutions —

namely, polynomial-time approximation algorithms, restricted-case polynomial-time algorithms, and fixed-parameter tractable algorithms. Though none of these approaches has yet produced results that are of immediate use in practical software modularization and remodularization, the fixed-parameter approach shows some promise and we have accordingly sketched the outlines of a fixed-parameter-based research program. It is our hope that even if our recommendations are not adopted, the results and approaches discussed here will be of use in guiding future research on software modularization and remodularization.

6. ACKNOWLEDGMENTS

I would like to thank the three reviewers for comments which helped greatly in improving the content and presentation of this paper. This work was supported by NSERC Discovery Grant RGPIN 228104-2015.

7. REFERENCES

[1] H. Abdeen, H. Sahraoui, O. Shata, N. Anquetil, and S. Ducasse. Towards automatically improving package structure while respecting original design decisions. In *Proceedings of the 20th Working Conference on Reverse Engineering (WCRE)*, pages 212–221, 2013.

[2] G. Bavota, F. Carnevale, A. De Lucia, M. Di Penta, and R. Oliveto. Putting the developer in-the-loop: An interactive GA for software re-modularization. In *Search Based Software Engineering*, pages 75–89. Springer, 2012.

[3] C.-E. Bichot and P. Siarry. *Graph Partitioning*. John Wiley & Sons, 2013.

[4] S. Böcker and P. Damaschke. Even faster parameterized cluster deletion and cluster editing. *Information Processing Letters*, 111(14):717–721, 2011.

[5] F. Bonomo, G. Durán, and M. Valencia-Pabon. Complexity of the cluster deletion problem on subclasses of chordal graphs. *Theoretical Computer Science*, in press.

[6] G. A. P. Crescenzi, S. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, Berlin, 1999.

[7] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

[8] R. Downey and M. Fellows. *Fundamentals of Parameterized Complexity*. Springer, Berlin, 2013.

[9] R. G. Downey, V. Estivill-Castro, M. Fellows, E. Prieto, and F. A. Rosamund. Cutting up is hard to do: The parameterised complexity of k-Cut and related problems. *Electronic Notes in Theoretical Computer Science*, 78:209–222, 2003.

[10] M. R. Fellows, F. V. Fomin, D. Lokshtanov, F. Rosamond, S. Saurabh, and Y. Villanger. Local search: Is brute-force avoidable? *Journal of Computer and System Sciences*, 78(3):707–719, 2012.

[11] L. Fortnow. The Status of the P Versus NP Problem. *Communications of the ACM*, 52(9):78–86, 2009.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W.H. Freeman, 1979.

[13] M. Harman, S. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys*, 45(1):11, 2012.

[14] F. Hüffner, C. Komusiewicz, A. Liebtrau, and R. Niedermeier. Partitioning biological networks into highly connected clusters with maximum edge coverage. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(3):455–467, 2014.

[15] K. Kawarabayashi and M. Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 160–169. IEEE Press, 2011.

[16] S. Kratsch and F. Neumann. Fixed-parameter evolutionary algorithms and the vertex cover problem. *Algorithmica*, 65(4):754–771, 2013.

[17] A. S. Mamaghani and M. Hajizadeh. Software modularization using the modified Firefly algorithm. In *Proceedings of the 8th Malaysian Software Engineering Conference*, pages 321–324, 2014.

[18] D. Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

[19] T. Mens and T. Tourwé. A survey of software refactoring. *IEEE Transactions on Software Engineering*, 30(2):126–139, 2004.

[20] B. S. Mitchell and S. Mancoridis. On the automatic modularization of software systems using the Bunch tool. *IEEE Transactions on Software Engineering*, 32(3):193–208, 2006.

[21] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1):173–182, 2004.

[22] U. Stege. The Impact of Parameterized Complexity to Interdisciplinary Problem Solving. In *The Multivariate Algorithmic Revolution and Beyond*, pages 56–68. Springer, Berlin, 2012.

[23] M. Thorup. Minimum k-way cuts via deterministic greedy tree packing. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, pages 159–166, 2008.

[24] T. Wareham. *Systematic Parameterized Complexity Analysis in Computational Phonology*. Ph.D. thesis, University of Victoria, 1999.

[25] A. Wigderson. P, NP and mathematics — A computational complexity perspective. In *Proceedings of ICM 2006: Volume I*, pages 665–712, 2007.

[26] M. Xiao, L. Cai, and A. C.-C. Yao. Tight approximation ratio of a general greedy splitting algorithm for the minimum k-way cut problem. *Algorithmica*, 59(4):510–520, 2011.

[27] M. S. Zanetti, C. J. Tessone, I. Scholtes, and F. Schweitzer. Automated software remodularization based on move refactoring: a complex systems approach. In *Proceedings of the 13th International Conference on Modularity*, pages 73–84, 2014.

APPENDIX

A. PROVING INTRACTABILITY

Given some criterion of tractability like polynomial-time or fixed-parameter solvability, we can define the class T of all

computational problems that are tractable relative to that criterion. For example, T could be the class P of decision problems (see below) solvable in polynomial-time, or FPT, the class of parameterized problems that are fp-tractable. We can show that a particular problem is not in T (and thus that this problem is intractable) by showing that this problem is at least as hard as the hardest problem in some class C that properly includes (or is strongly conjectured to properly include) T. For example, C could be NP, the class of decision problems whose candidate solutions can be verified in polynomial time, or a class of parameterized problems in the W-hierarchy $= \{W[1], W[2], \ldots XP\}$ (see [12] and [8], respectively, for details).

To establish such relative problem hardness, we will use reductions between pairs of **decision problems**, *i.e.*, problems whose outputs are either "Yes" or "No". The types of reductions used here are as follows.

Definition 2. Given a pair Π, Π' of decision problems, Π **polynomial-time reduces to** Π if there is a polynomial-time computable function f mapping instances I of Π to instances $f(I)$ of Π' such that the answer to I is "Yes" if and only if the answer to $f(I)$ is "Yes".

Definition 3. Given a pair Π, Π' of parameterized problems with parameters p and p, respectively, , Π **fp-reduces to** Π if there is a function f mapping instances $I = (x, p)$ of Π to instances $I' = (x', p')$ of Π' such that (i) f is computable in $g(p)|x|^{\alpha}$ time for some function $g()$ and constant α, (ii) $p' = h(p)$ for some function $h()$, and (iii) the answer to I is "Yes" if and only if the answer to $f(I)$ is "Yes".

A reducibility is appropriate for a tractability class T if whenever Π reduces to Π' and $\Pi' \in T$ then $\Pi \in T$. We say that a problem Π is C**-hard** for a class C if every problem in C reduces to Π. A C-hard problem is essentially as hard as the hardest problem in C.

Reducibilities become useful given the following properties:

1. If Π reduces to Π' and Π is C-hard then Π' is C-hard.
2. If Π is C-hard and $T \subset C$ then $\Pi \notin T$, *i.e.*, Π is not tractable.
3. If Π is C-hard and $T \subseteq C$ then $\Pi \notin T$ unless $T = C$, *i.e.*, Π is not tractable unless $T = C$.

The first and third properties are used in Section 4.3 and Appendix B to show intractability relative to T-classes P and FPT and C-classes NP, $W[1]$, and XP. These intractability results hold relative to the conjectures $P \neq NP$ and $FPT \neq W[1]$ which, though not proved, are commonly accepted as true within the Computer Science community (see [8, 11, 12] for details).

Note that though the modularization and remodularization problems given in Section 2 are not decision problems, they can be made into decision problems by asking if the requested outputs exist. Moreover, as any of the problems in Section 2 can be used to solve their corresponding decision versions, the (fp-)intractability of these decision versions also implies that the problems given in Section 2 are not (fp-)tractable unless $P = NP$ ($FPT = W[1]$).

B. PROOFS OF RESULTS

All of our intractability results will be derived using reductions from problems k-WAY CUT and k-CLUSTER DELETION given in Section 2 and the following problem:

RESTRICTED PARTITION INTO TRIANGLES (RPIT)
Input: An undirected 4-regular neighbourhood-restricted graph $G = (V, E)$.
Question: Can G be partitioned into $|V|/3$ sets such that each set of the partition induces a triangle-graph in G?

As knowing the definition of a 4-regular neighbourhood-restricted graph is not necessary to understand our proofs, the interested reader is referred to [14] for this definition.

LEMMA 1. *RPIT polynomial-time reduces to SMod-HC.*

PROOF. This reduction is a modification of that given in Lemma 2 of [14]. In that reduction, a given instance $\langle G = (V, E)\rangle$ of RPIT is transformed by a polynomial-time preprocessing algorithm into an instance $\langle G' = (V', E'), s = |V'|\rangle$ of problem HIGHLY CONNECTED DELETION where G' may have fewer vertices and/or edges than G. Problem HIGHLY CONNECTED DELETION is our problem SMod-HC without the requirement that a modularization have k highly connected components, *i.e.*, HIGHLY CONNECTED DELETION produces a partition of its given graph into an arbitrary number of arbitrary-size highly connected components. However, by the nature of the preprocessing, all highly-connected clusters of vertices in G' have size exactly three, which helps ensure that the given instance of RPIT has a solution if and only if at most $s = |V'|$ edges can be removed from G' to leave $|V'|/3$ triangles. One can thus rephrase this reduction as a transformation from a given instance $\langle G = (V, E)\rangle$ of RPIT into an instance $\langle G' = (V', E'), k = |V'|/3, s = |V'|\rangle$ of SMod-HC. □

LEMMA 2. *kWC polynomial-time reduces to SReMod-BC such that in the constructed instance of SReMod-BC, k_F is a function of k in the given instance of kWC*

PROOF. Given an instance $\langle G = (V, E), k, s\rangle$ of kWC, the constructed instance $\langle C, k_I, M, k_F, c_s, c_m\rangle$ of SReMod-BC has $C = G$, $k_I = |V|$, $M(v_i) = i$ for $1 \leq i \leq |V|$, $k_F = k$, $c_s = |E| - s$ and $c_m = |V| - k$. Note that this instance of SReMod-BC can be constructed in time polynomial in the size of the given instance of kWC.

If there is a set of s edges in the given instance of kWC whose deletion leaves G consisting of k connected components C_1, C_2, $\ldots C_k$, create M' as follows: for each C_i, select an arbitrary vertex v in C_i and set $M'(v) = M(v)$ and $M'(v') = M(v)$ for all other vertices v' in C_i. Note that M' is a $k = k_F$-modularization of C obtained by exactly $c_m = |V| - k$ module-assignment changes relative to M such that $F_{cpl}(C, M') = s$; moreover, as $F_{cpl}(C, M) = |E|$, $F_{cpl}(C, M') \leq F_{cpl}(C, M) - c_s = |E| - (|E| - s) = s = F_{cpl}(C, M')$. Conversely, suppose that there is a k_F-modularization M' of C such that $F_{cpl}(C, M') \leq F_{cpl}(C, M) - c_s$. As $k_F = k$ and each of the modules in C relative to M has connectivity BC, M' is a k-partition of G into connected components; moreover, as $F_{cpl}(C, M) = |E|$, $F_{cpl}(C, M') \leq$

$F_{cpl}(C, M) - c_s = |E| - (|E| - s) = s$, which implies that at most s edges needed to be removed to induce that k-partition of G.

To complete the proof, note that in the constructed instance of SReMod-BC, $k_F = k$. □

LEMMA 3. *RPIT polynomial-time reduces to SReMod-HC.*

PROOF. This can be proved using the following reduction that is a composition of the reductions in Lemmas 1 and 2. Given an instance $\langle G = (V, E)\rangle$ of RPIT, the constructed instance $\langle C, k_I, M, k_F, c_s, c_m\rangle$ of SReMod-BC has $C = G' = (V', E')$, where G' is created from G by the preprocessing algorithm from [14] mentioned in Lemma 1, $k_I = |V'|$, $M(v_i) = i$ for $1 \leq i \leq |V'|$, $k_F = |V'|/3$, $c_s = |V'|$ and $c_m = 2|V'|/3$. Note that this instance of SReMod-HC can be constructed in time polynomial in the size of the given instance of RPIT.

Suppose G can be partitioned into a set of $|V|/3$ triangle-graphs. As noted in the proof of Lemma 1, this is possible if and only if at most $|V'|$ edges can be removed from the graph G' created by preprocessing G to leave $|V'|/3$ triangle-graphs C_1, C_2, $\ldots C_{|V'|/3}$. From these triangle-graphs in G', create M' as follows: for each C_i, select an arbitrary vertex v in C_i and set $M'(v) = M(v)$ and $M'(v') = M(v)$ for the other two vertices v' in C_i. Note that M' is a $|V'|/3 = k_F$-modularization of C obtained by exactly $c_m = |V'| - (|V'|/3) = 2|V'|/3$ module-assignment changes relative to M such that $F_{cpl}(C, M') = |E'| - |V'|$ (as the $|V'|/3$ modules under M' each contain 3 edges); moreover, as $F_{cpl}(C, M) = |E'|$, $F_{cpl}(C, M') \leq F_{cpl}(C, M) - c_s = |E'| - |V'| = F_{cpl}(C, M')$. Conversely, suppose that there is a k_F-modularization M' of C such that $F_{cpl}(C, M') \leq F_{cpl}(C, M) - c_s$. As $k_F = |V'|/3$ and each of the modules in C relative to M has connectivity-type HC, M' is a $|V|/3$-partition of G' into connected components, each of which (by the nature of the preprocessing algorithm, as described in Lemma 1) must be triangle-graphs Moreover, as $F_{cpl}(C, M) = |E'|$, $F_{cpl}(C, M') \leq F_{cpl}(C, M) - c_s = |E'| - |V'|$, which implies that at most $|V'|$ edges needed to be removed to induce that partition of G'. Together this implies that G can be partitioned into $|V|/3$ triangle-graphs, which completes the proof. □

LEMMA 4. *kCD polynomial-time reduces to SReMod-CC such that in the constructed instance of SReMod-CC, k is a function of k in the given instance of kCD.*

PROOF. Observe that neither the construction in nor the proof of correctness of the reduction given in Lemma 2 from kWC to SReMod-BC depends on the type of connectivity of the components and modules in these problems; moreover, the only difference between problems kWC and kCD is the type of connectivity in the components remaining after the removal of the edges in E'. Hence, the reduction in 2 is also a valid reduction from kCD to SReMod-CC such that in the constructed instance of SReMod-CC, $k_F = k$. □

Result A: *SMod-BC, SMod-HC, SMod-CC, SReMod-BC, SReMod-HC, and SReMod-CC, are NP-hard.*

PROOF. The NP-hardness of SMod-BC (SMod-CC) follows from Observation 1 (2) and [9, Theorem 1] ([21, Theorem 14]). The NP-hardness of SMod-HC, S-ReMod-BC, SReMod-HC, and SReMod-CC follows from the NP-hardness of kWC, kCD, and RPIT (see [14, Section 2] and references) and Lemmas 1, 2, 3, and 4, respectively. □