On the Feasibility of Optimizing Software Product Lines with Quantum Annealing

Timothy Goodrich
Department of Computer Science
North Carolina State University
Raleigh, North Carolina 27606
tdgoodri@ncsu.edu

ABSTRACT

Software product lines are an abstract representation of real-world software engineering problems.

Keywords

Software Product Lines; Adiabatic Quantum Computing; Multi-Objective Optimization

1. INTRODUCTION

1.1 Motivation

Many software engineering problems reduce to optimization. How can we efficiently generate effect tests? How can we tune learners to accurately predict for bugs? How can we optimize the features offered in a product? Each of these classic problems are typically formulated as optimization problems at the intersection of search-based software engineering [14] and functional testing [1], defect prediction [12], and product line optimization [16], respectively. Unfortunately, these problems are nearly always NP-hard for an optimal solution and difficult to estimate, creating run times that hampered by exponential-sized search spaced. Developing fast, automated tools for solving these optimization problems, therefore, is of high practical interest to the software engineering community.

As mentioned above, one area of interest is Software Product Line (SPL) optimization (Figure 1). SPL models represent the full feature set of a product line. In the phone example, the different features such as GPS and screen density are listed, and specific phone instances (i.e. a specific phone product number) are found with valid configurations of the SPL model. For example, one valid cell phone can make calls, does not have a GPS, has a high resolution screen, and has a camera. Formally, SPLs are Constraint Satisfaction Problems, where a tree of features are connected with *mandatory* and *optional* edges, and *alternative* or (multiple-allowed) *or* children. Across this tree are *cross-tree constraints*, denoting *requirements* and *exclusions*. For example, the GPS feature excludes having a basic screen (and vice versa), and the camera requires having a high resolution screen.

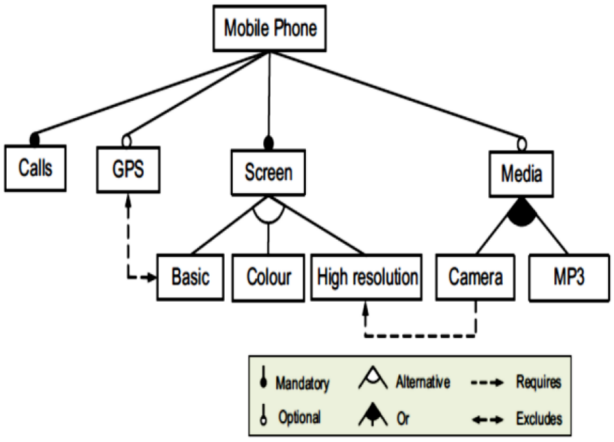

Figure 1: The phone software product line from [2]

While finding configurations to satisfy such a small, tree-like example is not hard, the problem becomes exponentially harder as the CSP becomes less and less tree-like. Specifically, CSPs are NP-hard using a trivial reduction of satisfiability (SAT). Therefore finding all valid configurations of a SPL model is considered difficult.

Compounding the complexity, usually more real-world information is provided with these models. Factors such as feature cost, feature reliability, and feature reception (i.e. tried-and-true measure) all provide different information and so must be solved using *multi-objective optimization*. Such optimization methods typically imply genetic algorithms such as NSGA-II [9] and SPEA 2 [6], but other optimization methods apply.

One such alternative is Chen et al.'s SWAY algorithm [7]. The SWAY algorithm begins by using a SAT-solver to generate all valid configurations of a SPL model, then uses precise sampling to efficiently find Pareto-optimal solutions. The authors found that the SAT-solver is actually the computational bottleneck in practice. Note that using a SAT solver this way is naturally a serial operation: once an initial solution is found, the ith solution is found by solving the same formula *and* the negation of the first $i-1$ solutions. With

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

this approach restricted to serial operation, a scalable alternative is desirable.

In this paper, we introduce the notion of using *quantum annealing* to find valid configurations of a SPL model. Based on the *adiabatic quantum computing* model by Farhi et al. [11], quantum annealing evolves an initial quantum system (represented by a Hamiltonian) to a final quantum system. The adiabatic theorem [11] guarantees that a slow-enough evolution leaves the final quantum system in a *ground state*, therefore a state of minimum energy can be computed with the system if appropriate *charges* are applied to the initial quantum system. Popularized by the D-Wave System Inc. production models, quantum annealing is rapidly becoming a viable alternative to optimization black boxes such as SAT-solvers.

Figure 2: The D-Wave System Inc.'s *D-Wave 2X* quantum annealer. The primary cooling unit houses a 1152-qubit processor, with IO and networking handled by a front-mounted server rack.

The primary motivation to use a quantum annealer for SPL model optimization is for the *quantum solutions*. In classical computing, an optimal solution to an optimization problem is an assignment such that an objective function is maximized or minimized (as desired). Several such optimal solutions can exist, of course. However, in quantum computing, an optimal solution is a *uniform probability distribution* over all assignments that maximize or minimize the objective function. One effect of this nuance is that very rugged terrain in the objective function leads to faster solution convergence; the solution takes advantage of *quantum tunneling* to burrow through objective function hills. The speedups from quantum tunneling have proven to be exponential, with work by [10] showing a speedup of over 10,000,000 compared to classical annealing algorithms.

More useful for the SPL model optimization, though, is a second effect: a single run of the quantum annealer returns a distribution over the solutions found. These solutions are ranked by energy, which is a dependent function on the optimization problem's objective function, therefore optimal solutions are immediately recognizable. Given the standard run time of a modern D-Wave 2X annealer, over 1000 solutions can be collected in under a second. Again, given the serialization of SAT solvers for finding all valid configurations, the run times of this quantum annealer offer a reasonable alternative to classical algorithms, even in its experimental stage.

The downside to the quantum annealer is the amount of resources available, though. The current generation D-Wave 2X offers 1152-qubits in a *Chimera* graph layout, which is very sparse in terms of connections between qubits. These connections provide the analog of "memory" when embedding an optimization problem, so in effect there are a very limited number of constraints possible per qubit. To overcome this problem, multiple qubits can be assigned to represent a single optimization problem variable, so then the bottleneck comes entirely from how many qubits the optimization problem requires.

1.2 Contributions

In this paper we set the groundwork for developing full-pipeline algorithms for embedding CSP (specifically SPL) optimization problems into the D-Wave 2X. Our goals are defined by the following research questions:

- **RQ1 (Available Methods):** *What methods are there for converting SAT-based optimization problems into Ising Model problems?*
- **RQ2 (Method Practicality):** *How badly do these conversion methods blow up the initial problem?*
- **RQ3 (Bottlenecks):** *What problems can we expect to run on current hardware and expected future hardware? What problems are keeping larger problems from running?*

The remainder of this paper is organized as follows. Section provides background on related work in optimizing Software Product Lines, along with a general overview on how to use the D-Wave 2X quantum annealer. Section 3 outlines our work on developing algorithms for converting SPL datasets from the SPLOT repository into the *Ising Model* format needed to communicate with the D-Wave 2X, and we provide an overview of the software developed for these algorithms. Section 4 summarizes our experimental results on the SPLOT datasets, and we offer observations the practical implications. Section 5 provides a summary of the results presented in this paper, a justification on the scientific nature of this work, an overview of planned additional work.

2. BACKGROUND

2.1 Related Work

Feature models were first introduced to software engineering by Kang et al. in the 1990s [16]. An excellent survey by Benavides et al. covers the last 20 years of approaches [2]. They note that SAT solvers are by far the most common approach, with some alternatives being binary decision diagrams (BDDs) and Carnegie Mellon's SMV model checker.

In more recent years, several product-line–specific algorithms have been published, such as Sayyad et al.'s IBEASEED

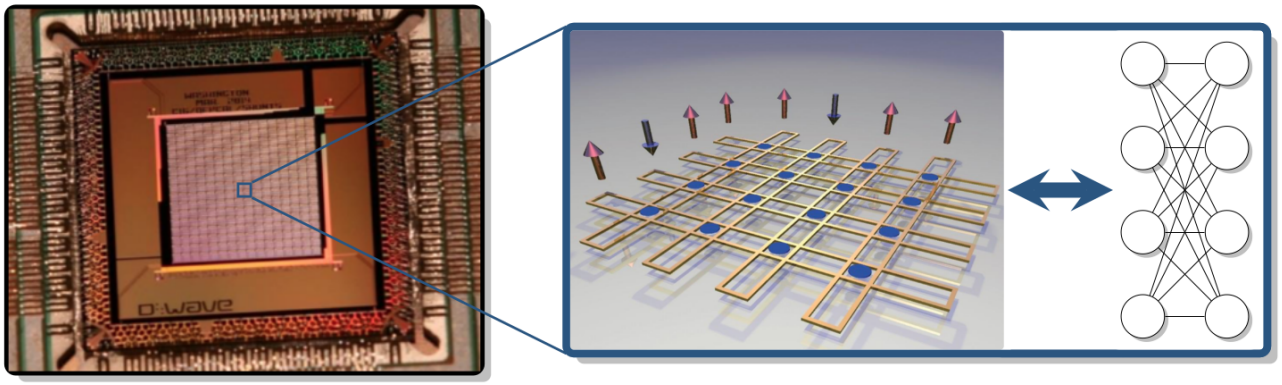

Figure 3: (Left) The D-Wave System Inc.'s upcoming *D-Wave 2000Q* 2048-qubit processor, a 16×16 grid of cells; (Center) a single cell, where 4 qubits (long) are overlaid on 4 other qubits orthogonally, with 16 overlapping areas that are used for qubit couplers. (Right) The graph-theoretic interpretation of a cell: a complete bipartite graph $K_{4,4}$. From [13].

[21] and Henard et al.'s SATIBEA [15]. The latter uses genetic algorithms to explore the (valid and invalid) solution space, and uses a SAT solver to patch together invalid solutions that resulted from mutation. We base our work off of the recent work by Chen et al. [7], who use a SAT solver to build the (valid) solution space first before doing multi-objective optimization.

In this context, we suggest the reader views quantum annealing as a black-box replacement for a SAT solver. As mentioned before, the quantum tunneling facet of the machine makes generating multiple solutions an advantage over a standard SAT solver, but this advantage could be used in other ways. For example, a majority of solutions could be generated by quantum annealing, following by using a SAT solver to solve the (reduced) formula exactly for the remaining few solutions. Alternately, some solutions in the quantum annealing distribution might not be valid, but rather a fairly-optimal solution to the MAX-SAT version of the problem. In this case we could view these solutions as mutations with a high number of satisfied clauses, and generate these mutations as a middle step in a modified version of Henard et al.'s SATIBEA algorithm.

2.2 D-Wave Quantum Annealing

Adiabatic quantum computing was first published about in the early 2000s by Farhi et al. [11]. The model was introduced as an alternative to the *circuit-based quantum computing* model that has dominated the literature. Theoretically, the two models are equivalent, and equivalent also to the *universal quantum computer*, an analog to the Turing Machine for quantum computing.

In 2011, D-Wave Systems Inc. produced their first production-quality quantum annealer, the 128-qubit D-Wave One. Based on adiabatic quantum computing, the quantum annealer is fundamentally limited by physical constraints and *not* equivalent to the universal quantum computer. However, the annealer excels at solving a quantum mechanics problem, the *Ising Model problem*:

$$\arg\min \sum_{1 \leq i \leq n} a_i s_i + \sum_{i<j} b_{ij} s_i s_j \tag{1}$$

where $a_i, b_{ij} \in \mathbb{R}$ and $s_i \in \{-1, +1\}$. The s_i variables are the qubits and are assigned *spins*. This equation is naturally represented as a graph with vertices s_i, edges $s_i s_j$, vertex weights a_i and edge weights b_{ij}. Figure 3 visualizes an upcoming D-Wave chip, the qubits, and the graph form.

One limitation of the D-Wave hardware vs. the theoretical quantum annealing model is that not all possible edges are present in the D-Wave hardware. The D-Wave hardware has classically been instances of *Chimera* graphs, an $M \times N$ grid of complete bipartite cells (Figure 3). To circumvent this issue, Choi [8] introduced the notion of *minor embedding* an optimization problem graph into the hardware graph. At a high level, minor embedding selects a connected set of *physical qubits* from the hardware graph to represent each *logical qubit* in the problem graph. The vertex and edge weights from the problem graph are then distributed appropriately onto the physical qubits, providing a mapping between the *Ising Model* version of the problem that the quantum annealer solves and the optimization problem that the user is interested in solving. Figure 4 visualizes this process.

We note that optimization problems are typically formulated as *quadratic unconstrained binary optimization problems (QUBOs)*, defined as the following:

$$\arg\min \sum_{i<j} c_{ij} x_i x_j \tag{2}$$

where $c_{ij} \in \mathbb{R}$ and $x_i \in \{0, 1\}$. Note how similar Equations 1 and 2 are, a simple variable change can convert from one to the other. We recommend the analogy of a QUBO is to quantum annealing what a linear program is to an LP-solver – each are formats for feeding optimization problems into tuned black boxes.

Unfortunately, minor embedding is itself an NP-hard problem, and Choi mentions that previous algorithms for distributing logical processes on physical nodes from parallel computing do not apply. However, in the last five years several papers have been published with embedding algorithms to handle different structural considerations [4, 13, 18], and D-Wave provides an in-house heuristic in their API [5].

In total, the D-Wave quantum annealer has experienced rapid improvements over the last five years. The first itera-

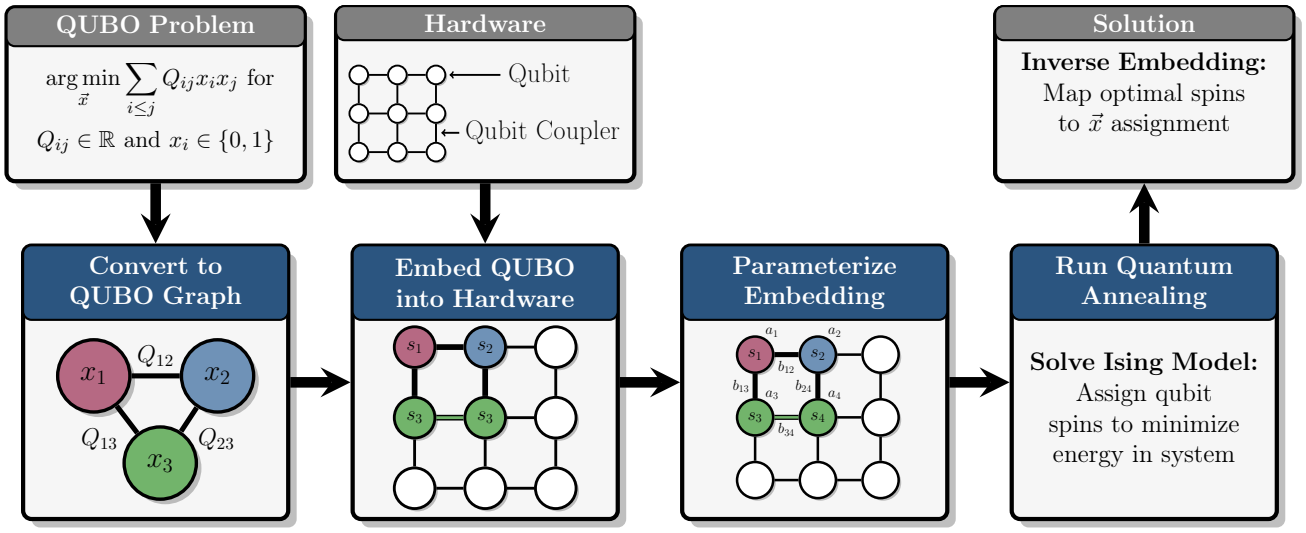

Figure 4: The algorithmic pipeline for embedding a QUBO into an Ising Model, running the hardware, and mapping the quantum solution back to the original optimization problem. From [13].

tion, the D-Wave One, was a 4×4 grid of $K_{4,4}$ cells. The next two models, the D-Wave Two and 2X, upped the grid sizes to 8×8 and 12×12, respectively. The D-Wave 2X is actually a partially-disabled 16×16 grid, and a more mature manufacturing process is currently under development to yield the next generation 2048-qubit D-Wave $2000Q$ model.

In terms of performance and applications, a recent paper by the research group at Google, one of the primary clients, showed a problem with a harsh energy landscape that the D-Wave annealer could solve 10,000,000 times faster than classical approaches [10]. Major applications also exist in computational chemistry [17], image recognition [20], NASA Mars missions [23], and Karp's 21 NP-hard problems [19].

3. RQ1: AVAILABLE METHODS

In the last section we saw how much of the D-Wave embedding research effort has assumed that an optimization problem is provided as a QUBO, which can then be embedded into the hardware's Ising Model. Therefore the first step to running Software Product Line optimization on the quantum annealer is to obtain a QUBO formulation. In this section we explore different ways of accomplishing this conversion. While none of these algorithms are spelled out explicitly, we use a few conversion techniques are used D-Wave documentation [22].

3.1 Method 1: Maximum Independent Set

The first method we will outline is based on solving a maximum independent set (MIS) problem. The algorithm first converts the SPL dataset into a satisfiability problem with any first-order logical operator (and, or, not, if, iff). Then a *conflict graph* (Figure 5) is constructed, where vertices are valid assignments to the SPL features and there is an edge between two vertices if they conflict on an assignment. For example, given the constraint **Mobile Phone** $\longleftrightarrow$ **Calls**, which corresponds to the "mandatory" edge from the SPL, there are two valid assignments: either both features are zero or both are one. For the constraint **GPS** $\rightarrow$ **Mobile Phone**

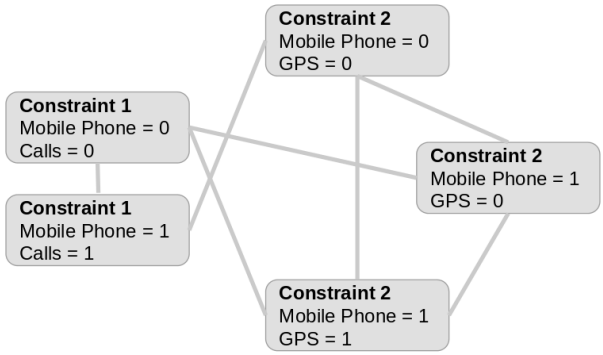

Figure 5: A conflict graph for two constraints: Mobile Phone $\longleftrightarrow$ Calls and GPS $\rightarrow$ Mobile Phone

(i.e. the GPS is optional) there are three valid assignments: everything is zero, everything is one, and the phone is one while the GPS is zero. These assignments generate the vertices of the graph and edges denote where assignments have conflicts. For example, we cannot set **Mobile Phone** $= 0$, **Calls** $= 0$ and **Mobile Phone** $= 1$, **GPS** $= 0$. While both assignments are valid individually, they disagree on **Mobile Phone**, and so are not a valid global assignment. Therefore there is an edge between these two assignments in the conflict graph.

Once the conflict graph is constructed, we find a valid assignment if and only if we can find an independent set of size equal to the number of constraints in the SPL model. In graph theory, in independent set is a set of vertices such that no two vertices have an edge between them. In this problem, an independent set says that we have found an assignment that does not have any conflicts, and we satisfy one constraint from the SPL model for each vertex in our independent set. Given that any two vertices for a single constraint will have an edge between them (i.e. they are

Constraint	Penalty
$z \iff \neg x$	$2xz - x - z + 1$
$z \iff x_1 \vee x_2$	$x_1x_2 + (x_1 + x_2)(1 - 2z) + z$
$z \iff x_1 \wedge x_2$	$x_1x_2 - 2(x_1 + x_2)z + 3z$
$z \iff x_1 \oplus x_2$	$2x_1x_2 - 2(x_1 + x_2)z - 4(x_1 + x_2)a + 4az + x_1 + x_2 + z + 4a$

Table 1: Boolean logic as penalty-based posiforms.

two distinct assignments), we know that a maximum independent set will satisfy all constraints, therefore a solution to this problem will satisfy the SLP model.

At first glance, choosing maximum independence set as a format seems like a poor choose, given that it is classically NP-hard. However, an early result from Choi [8] showed that MIS has a *posiform* formulation compatible with QUBOs:

$$\min_x \left\{ \sum_{v \in V} \bar{x}_v + M \sum_{v,v' \in E} x_v x_{v'} \right\} \tag{3}$$

where $x_v \in \{0, 1\}$ and $M \in \mathbb{R}^+$ is a penalty constant. Note what this equation is saying at a high-level: minimize this equation where $\bar{x}$ (not x) is minimized (i.e. the number of vertices is maximized), and the number of occurrences of $x_v = x_{v'} = 1$ with a multiplied penalty is minimized (i.e. there is a penalty for every edge in the independent set). Therefore an optimal solution to this equation maximizes vertices and minimizes edges, and is an independent set. This posiform is in the QUBO format after replacing $\bar{x}$ with $(1 - x)$.

In summary, this method converts SPL to a MIS instance, which then has a trivial conversion to a posiform, which then has a trivial conversion to a QUBO. In some sense this is the most efficient algorithm because we did not require the SPL model in any particular form, we just needed to derive the valid assignments. However, this assumption also makes this algorithm a poor choice in practice: each constraint forms a clique (a complete graph with all edges turned on). If a single constraint has several literals "OR"ed together then the clique is exponential in the number of literals in the constraint. This is a particular problem for the Chimera graph because Klymko et al. [18] show with a treewidth argument that the Chimera graph can only embed cliques of up $LN+1$ (the cell height and the smallest grid dimension). Therefore these complete graphs will render the generated QUBO un-embeddable.

3.2 Method 2: MAX-2-SAT Posiforms

To combat the problem from the last method (exponential-sized cliques), we developed a second method using another posiform that converts easily to a QUBO: MAX-2-SAT. Recall that SLP models are typically expressed as Constraint Satisfiability Problems (CSPs), which are a type of satisfiability problem (SAT). Another variation of SAT is MAX-2-SAT, where clauses have at most two literals and each comes with a weight, and the goal is to maximize the sum of the weights from satisfied clauses. While 2-SAT is in P, the maximization part makes MAX-2-SAT NP-hard. Formulated as a posiform:

$$\min_{\vec{x}} \sum_c w_c \bar{l}_{c,1} \bar{l}_{c,2} \tag{4}$$

where w_c is the clause's weight and $l_{c,1}$ and $l_{c,2}$ are the literals. Note that the literals are negated in the posiform, meaning that we want to minimize the number of unsatisfied clauses.

Using this new posiform as the base of our conversion algorithm, the formulations we need are as follows. First, we convert the SLP model into a Conjunctive Normal Form version of SAT (uses only "OR" and "NOT"). From CNF we use the gadgets from 3.1 to convert the problem to a MAX-SAT posiform, then conclude by converting the MAX-SAT posiform to a MAX-2-SAT posiform. What this means is that clauses with more than two literals need to be broken up by introducing auxiliary variables. Specifically, we need the following penalty function:

$$x_1x_2x_3 = \min_z \{zx_3 + MP(x_1, x_2; z))\} \tag{5}$$

where

$$P(x_1, x_2; z) = x_1x_2 - 2(x_1 + x_2)z + 3z. \tag{6}$$

At a high level, Equation 5 says that we can introduce an auxiliary variable z that represents x_1x_2, and it suffices to minimize zx_3 subject to a penalty if x_1x_2 does not agree with z. Equation 6 then is the gadget that links z with x_1x_2, taken straight from Table 3.1.

Note that the above gadget reduces a clause by one variable by introducing a new variable to represent two old ones, therefore the most efficient use of this gadget is in a binary tree. For example, for $x_1x_2x_3x_4x_5$ we would introduce z_1 to represent x_1x_2, z_2 to represent x_3x_4, z_3 to represent z_1z_2, and we would then evaluate x_5z_3.

In total, then, this method works by converting CSP to CNF, CNF to MAX-SAT posiform, MAX-SAT posiform to MAX-2-SAT posiform, then MAX-2-SAT posiform to QUBO using the same trivial change of variable used in the last method.

3.3 Summary

In summary, in this section we provide two methods for converting SPL model optimization problems into QUBOs. The first method utilizes maximum independent sets and required few conversions, but suffers from exponential-sized cliques. The second method requires many more conversions, but ultimately can handle the previously-problematic area (large "OR" clauses) with only a linear number of added constraints. In the next section we evaluate how this second method performs in practice.

4. RQ2: METHOD PRACTICALITY

In this section we evaluate the second method from the previous section. Specifically, we are interested in gauging if this conversion causes too much variable/constraint blowup

for the method to be practical on the current hardware. Additionally, we are interested in looking at the graph structure in the generated QUBO, such that tuned minor embedding algorithms can take advantage of the structure.

4.1 Data

For data, we use the Software Product Line data available from the Software Product Lines Online Tools (SPLOT) repository (http://www.splot-research.org/). Table 4.1 lists the specific datasets we selected, where the values listed are taken directly from the SPLOT online feature model editor.

4.2 Algorithm Implementation

We provide the source code for the implementation of method 2 at https://github.com/tdgoodrich/fss16tdg/tree/master/project/Code. We start by reading the SPLOT dataset in SXFM (Simple XML Feature Model) format into a CSPTree object. This tree was then pruned to remove the XOR and OR groups that SPLOT provides as vertices. We note that Chen et al. [7] did not do this postprocessing, resulting in CNF formulations with more variables and constraints. While we expect their formulations still led to correct results, the conversion method used in this paper maintains the statistics provided by SPLOT in the online feature model.

Once a dataset is read into a CSPTree object it is then converted into CNF using a standard conversion. This conversion can be manually derived by simply forming a truth table for the CSP constraint, listing the conjunction of the negation of the *invalid* solutions, and then using De Morgan's laws to distribution the negation into each clause. The CNF is then written to a .cnf file. The SPLOT datasets were found to have long IDs, we simply replaced the ith ID with x_i. Therefore we also provide a name association CSV that lists each vertex's name (e.g. **Mobile Phone**), dataset ID (e.g. _r_1), and internal ID (e.g. x_1). This code is all included in a parse_sxfm.py script.

We also provide a script (parse_dnf.py) for converting CNF to the posiforms and QUBOs. This script follows the steps outlined above for method 2. We output a QUBO where each term has been simplified. It is possible that there are multiple terms with the same variables, however these terms are quickly collected as the QUBO is read into a NetworkX graph object. The graph object is then used to compute the statistics provided.

4.3 Experimental Results

For the datasets from Table 4.1, the only preprocessing done was to replace special characters in names with an underscore. The parse_sxfm.py script was then run with the SPLOT XML file as input, and the outputted CNF data file was then fed into parse_cnf.py, where the graph statistics were printed to standard output.

The graph statistics collected are presented in Table 4.1. We note the wide range of variable blow-up: some datasets only increased 11-13%, while others increased by 272%! The biggest increase, Dell Laptop/Notebook Computers, can be explained by the unusually high number of cross-tree constraints. The next highest, Strategy Mobile Game, has a relatively high number of XOR functions compared to its features (21%).

While there is variable blow-up in converting the feature models to QUBOs, an encouraging sign is the fairly high diameter of the QUBOs. The diameter graph is the furthest closest distance over all pairs of vertices. We can think of the diameter as a measure of density, where the more edges we have the more "hubs" will form that provide quick access to the other vertices. In the extreme cases, a complete graph has diameter 1 and is the densest graph possible, whereas a path has diameter n and is as sparse as possible (without being trivial). Seeing diameters of 5+ is encouraging because it means that vertices can be more spread out, and we expect they will embed well on the Chimera (grid-like) hardware. A randomized heuristic like CMR [5] might not provide an immediate embedding, but the diameter is high enough so that we can most likely find an embedding by hand.

4.4 Expected Practicality

Given the QUBO sizes provided in 4.1, we know that Mobile Phone, Toyota Feature Model, and Game Entity Model are immediately embeddable on the D-Wave 2X hardware since it supports complete graphs of up to 33 vertices. Given the fairly large diameters, we also expect that Webshop, Strategy Mobile Game, and Bike Computer are embeddable with some careful consideration for structure. The largest three datasets (Dell Laptop/Notebook Computers, BankingSoftware, and Electronic Shopping) will need significantly larger hardware; even the 16×16 grid in the upcoming D-Wave 2000Q will not be enough for existing algorithms that exploit bipartite structure [13]. The development of diameter-exploiting embedding algorithms might provide the means of embedding into the D-Wave 2000Q, but certainly not the D-Wave 2X.

5. CONCLUSION

5.1 Summary of Results

In summary, we have shown how to take Software Product Line datasets and convert them to QUBO form, which are compatible with quantum annealing hardware. We have shown how a method based on maximum independent set yields unreasonably-sized cliques, rendering it ineffective for modern D-Wave hardware. We have also shown a SAT-based method that generates QUBOs with reasonable blow-up in variables. Additionally, we note the encouraging sign of a fairly high diameter in these QUBOs, which suggests that they will embed nicely on grid-based hardware like the current- and next-generation D-Wave hardware.

5.2 Future Work

The first step in future work is to complete the full embedding pipeline and run the smaller examples on existing hardware. Even if the problem is embeddable and has fast run time, we still do not know the expected solution quality for these datasets. While the quantum annealers are supposed to provide a uniform distribution over valid solutions, in practice there are some biases. In the extreme case, such a bias could return merely a handful of solutions when hundreds exist, therefore we should give a full proof-of-concept before developing algorithms for larger problems.

Supposing the hardware does provide useful solutions, the next step would be to develop embedding algorithms for the medium-sized (50 to 120) QUBOs, which requires a fundamental understanding of the existing QUBOs' structure. An encouraging sign is that there are not many edges per

Dataset Name	Features	# Mandatory	# Optional	# XOR	# OR	# Cross-Tree Constraints
Mobile Phone	10	2	2	1	1	2
Webshop	46	10	29	1	1	3
Toyota Feature Model	13	0	0	3	1	1
Game Entity Model	21	5	7	3	0	0
Strategy Mobile Game	33	6	5	7	1	4
Bike Computer	40	20	6	1	4	0
DELL Laptop/Notebook Computers	47	7	1	8	0	38
BankingSoftware	176	46	77	10	6	4
Electronic Shopping	290	75	82	0	40	21

Table 2: SPLOT datasets used.

Dataset Name	Feature → Vertex Increase	Vertices	Edges	Diameter
Mobile Phone	70%	17	22	5
Webshop	13%	52	63	8
Toyota Feature Model	92%	25	38	5
Game Entity Model	52%	32	44	7
Strategy Mobile Game	100%	66	98	8
Bike Computer	33%	53	60	7
DELL Laptop/Notebook Computers	272%	175	280	6
BankingSoftware	38%	243	300	11
Electronic Shopping	11%	323	331	∞

Table 3: Graph statistics for the QUBO generated by method 2 for each SPLOT dataset. Colors represent 0-49% increase (green), 50-99% increase (orange), and 100%+ increase (red).

vertex (roughly 2), and there's a larger diameter. We expect that these QUBOs still retain some structure of the original CSP. Understanding what this conversion has done on a local level, and developing methods for embedding based on this overall structure, is a good next step for novel research.

A final suggestion is to find methods for *decomposing* the initial CSP such that the pieces can be solved individually and then patched together. Bian et al. [3] provide two such embedding methods for constraint problems and claim speedup over cutting edge SAT solvers. However, the problem corpus is a specific set of fault diagnostics on circuits that embeds directly into the hardware, and the SAT-solver in question is from a 1985 benchmark, so we remain skeptical of the generality of their methods. Exploring how their decomposition algorithms could be applied or generalized is a good second step, once it is understood how best to embed CSP datasets with SPL properties.

5.3 Data Science vs. Data Dabbling

Good data or software science follows the scientific method, which is based on conjecture and (formal or evidence-based) proof. We claim that our work is data science, instead of mere data dabbling, based on the following facets:

- The asking and answering of concrete questions. Our goal is to establish baseline methods for converting SPL datasets to QUBOs, with the intention of embedding thee QUBOs into D-Wave hardware. By phrasing our research around this question, we are able to make particular assumptions such as looking at SPL data and its structure and the D-Wave hardware and its structure, and base our exploration and conclusions off of this direction. We avoid making claims that our methods apply to Constraint Satisfiability Problems in general. Although they might, that claim is certainly beyond the certification envelope provided with this work.

- We use open source datasets. Using privatized data is a very quick way to eliminate all reproducibility of an experiment, therefore we use open source data. Additionally, we link to the data repository, provide the exact dataset names, and provide dataset parameters in Table 4.1. The latter provides a certificate of the data that we used; any future changes to the data in the SPLOT repository may or may not apply to these results, and these statistics provide a way to verify that other users are running on the same data.

- We provide open source scripts. These scripts are written modularly so that each step can be written out to a file and read back in later. This allows intermediate improvements to be made without invalidating the entire script. Moreover, we checked output by hand to verify its correctness, and in doing so we found the previously mentioned incorrectness in Chen et al.'s script [7].

6. ACKNOWLEDGMENTS

The author would like to thank Professor Tim Menzies for his advice and support in the Foundations of Software Science course, and Dr. Travis Humble for introducing the author to quantum annealing.

7. REFERENCES

[1] B. Beizer. *Black-box testing: techniques for functional testing of software and systems*. John Wiley & Sons, Inc., 1995.

[2] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010.

[3] Z. Bian, F. Chudak, R. Israel, B. Lackey, W. G. Macready, and A. Roy. Mapping constrained optimization problems to quantum annealing with application to fault diagnosis. *arXiv preprint arXiv:1603.03111*, 2016.

[4] T. Boothby, A. D. King, and A. Roy. Fast clique minor generation in chimera qubit connectivity graphs. *Quantum Information Processing*, 15(1):495–508, 2016.

[5] J. Cai, W. G. Macready, and A. Roy. A practical heuristic for finding graph minors. *arXiv preprint arXiv:1406.2741*, 2014.

[6] P.-C. Chang, S.-H. Chen, and C.-H. Liu. Sub-population genetic algorithm with mining gene structures for multiobjective flowshop scheduling problems. *Expert systems with applications*, 33(3):762–771, 2007.

[7] J. Chen, V. Nair, R. Krishna, and T. Menzies. Is "sampling" better than "evolution" for search-based software engineering? *arXiv preprint arXiv:1608.07617*, 2016.

[8] V. Choi. Minor-embedding in adiabatic quantum computation: II. minor-universal graph design. *Quantum Information Processing*, 10(3):343–353, 2011.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[10] V. S. Denchev, S. Boixo, S. V. Isakov, N. Ding, R. Babbush, V. Smelyanskiy, J. Martinis, and H. Neven. What is the computational value of finite range tunneling? *arXiv preprint arXiv:1512.02206*, 2015.

[11] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000.

[12] N. E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Transactions on software engineering*, 25(5):675–689, 1999.

[13] T. D. Goodrich, B. D. Sullivan, and T. S. Humble. A graph-theoretic framework for adiabatic quantum computating compilation, 2016.

[14] M. Harman and B. F. Jones. Search-based software engineering. *Information and software Technology*, 43(14):833–839, 2001.

[15] C. Henard, M. Papadakis, M. Harman, and Y. Le Traon. Combining multi-objective search and constraint solving for configuring large software product lines. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 517–528. IEEE, 2015.

[16] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document, 1990.

[17] I. Kassal, J. D. Whitfield, A. Perdomo-Ortiz, M.-H. Yung, and A. Aspuru-Guzik. Simulating chemistry using quantum computers. *arXiv preprint arXiv:1007.2648*, 2010.

[18] C. Klymko, B. D. Sullivan, and T. S. Humble. Adiabatic quantum programming: minor embedding with hard faults. *Quantum information processing*, 13(3):709–729, 2014.

[19] A. Lucas. Ising formulations of many np problems. *arXiv preprint arXiv:1302.5843*, 2013.

[20] H. Neven, G. Rose, and W. G. Macready. Image recognition with an adiabatic quantum computer i. mapping to quadratic unconstrained binary optimization. *arXiv preprint arXiv:0804.4457*, 2008.

[21] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Scalable product line configuration: A straw to break the camel's back. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 465–474. IEEE, 2013.

[22] D-Wave Systems Inc. Programming with qubos release 2.4, 2016.

[23] D. Venturelli, S. Mandrà, S. Knysh, B. OâĂŹGorman, R. Biswas, and V. Smelyanskiy. Quantum optimization of fully connected spin glasses. *Physical Review X*, 5(3):031040, 2015.