# What Works Best When? A Systematic Evaluation of Heuristics for Max-Cut and QUBO

Iain Dunning, Swati Gupta, John Silberholz
Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139,
idunning@mit.edu, swatig@mit.edu, josilber@mit.edu

Though empirical testing is broadly used to evaluate heuristics, there are shortcomings with how it is often applied in practice. In a systematic review of Max-Cut and Quadratic Unconstrained Binary Optimization (QUBO) heuristics papers, we found only 4% publish source code, only 14% compare heuristics with identical termination criteria, and most experiments are performed with an artificial, homogeneous set of problem instances. To address these limitations, we implement and release as open-source a code-base of 10 Max-Cut and 27 QUBO heuristics. We perform heuristic evaluation using cloud computing on a library of 3,296 instances. This large-scale evaluation provides insight into the types of problem instances for which each heuristic performs well or poorly. Because no single heuristic outperforms all others across all problem instances, we use machine learning to predict which heuristic will work best on a previously unseen problem instance, a key question facing practitioners.

*Key words*: computational testing, reproducible research, heuristics, binary quadratic optimization, Max-Cut, hyper-heuristics, test bed, interpretable machine learning

## 1. Introduction

Many classical optimization problems encountered in operations research practice are NP-hard; due to the exponential worst-case runtime of known exact approaches for such problems, heuristics and approximation algorithms are the main tools available to solve large-scale problem instances. Theoretical justifications such as worst-case approximation ratios are often used to evaluate the performance of approximation algorithms and are attractive due to their generality — they apply across all possible problem instances. However, it is well known that procedures will often significantly outperform their worst-case bounds on real-world instances, and practitioners are often interested in how well a procedure will perform in practice. As a result, many heuristics papers perform empirical testing, in which a procedure is tested on a set of problem instances.

Though empirical testing is the major tool for evaluating heuristics, in this work we highlight a number of shortcomings with how it is often applied in practice and address these weaknesses for the Max-Cut and Quadratic Unconstrained Binary Optimization

(QUBO) problems. Max-Cut ($\max_{\mathbf{y}\in\{-1,1\}^n} \frac{1}{4}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(1-y_iy_j)$, given weighted adjacency matrix $\mathbf{W}\in\mathbb{R}^{n\times n}$) is one of Karp's 21 NP-complete problems (Karp 1972), and QUBO ($\max_{\mathbf{x}\in\{0,1\}^n} \mathbf{x}'\mathbf{Q}\mathbf{x}$, given $\mathbf{Q}\in\mathbb{R}^{n\times n}$) is also a classical optimization problem; a simple transformation exists to reduce each problem to the other[1]. (Hammer 1965). Many heuristics have been designed for these problems — in a systematic review completed as part of this work (see Appendix C), we found that 95 papers have performed empirical testing on such heuristics, 31 of which were published since 2010. These problems are also of interest due to their broad practical applicability: Max-Cut applications include VLSI design, network design, statistical physics, and image segmentation (Barahona et al. 1988, Barahona 1996, de Sousa et al. 2013), and QUBO applications include capital budgeting and financial analysis, computer aided design, traffic message management problems, machine scheduling, and molecular conformation (Merz and Freisleben 2002).

Despite the volume of heuristics research for Max-Cut and QUBO and the problems' range of applications, empirical testing of new heuristics for these problems has a number of deficiencies.

**Reproducibility:** To the best of our knowledge, only 4% of heuristics papers for Max-Cut and QUBO have publicly accessible source code, meaning any attempt to replicate empirical testing in one of the remaining papers would require either reimplementing that heuristic from its pseudocode or requesting its source code from the paper's authors. Even if source code were obtained, other reporting limitations hinder reproducing the computational environment used for testing: 20% of papers do not report the processor used and 87% of papers with compiled languages do not report the compiler flags used.

**Fair, broad comparison:** Though 81% of heuristics papers for Max-Cut and QUBO perform a comparison against a previously published procedure using a specified set of instances, most do not do so in a fair or broad manner (see Figure 1). In 86% of comparisons, solution qualities are compared across procedures run with different termination criteria, often leaving readers unsure if one method had higher solution quality than another due to its effectiveness or due to it being given a larger computational budget. Even though

[1] An $n$-variable QUBO instance ($\max_{\mathbf{x}\in\{0,1\}^n} \mathbf{x}'\mathbf{Q}\mathbf{x}$) can be reduced to an $(n+1)$-node Max-Cut instance ($\max_{\mathbf{y}\in\{-1,1\}^{n+1}} \frac{1}{4}\sum_{i=1}^{n+1}\sum_{j=1}^{n+1} w_{ij}(1-y_iy_j)$) by assigning $w_{ij}=-q_{ij}$ for all $i,j\in\{1,\ldots,n\}$ and by assigning $w_{i,n+1}=w_{n+1,i}=q_{ii}+\sum_{j=1}^{n} q_{ij}$ for all $i\in\{1,\ldots,n\}$. The optimal solution $x_i^*$ of QUBO relates to the optimal solution $y_i^*$ of Max-Cut as $x_i^*=\frac{1}{2}(1-y_i^*y_{n+1}^*)$. Similarly, an $n$-node Max-Cut instance can be expressed as an $n$-variable QUBO instance by considering $q_{ij}=-w_{ij}$ for all $i,j\in\{1,\ldots,n\}$, $i\neq j$ and $q_{ii}=\sum_{j=1}^{n} w_{ij}$ for all $i\in\{1,\ldots,n\}$, and the optimal solutions relate as $y_i^*=2x_i^*-1$

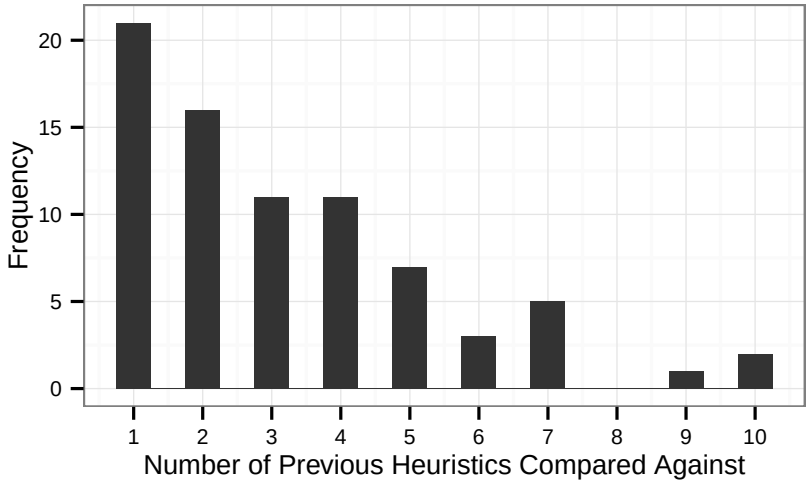


**Figure 1** **Histogram of the number of previously published heuristics with which a comparison was performed across the 77 Max-Cut and QUBO papers that compare against previously published heuristics.**

more than 90 papers have published heuristics for Max-Cut and QUBO, papers that perform comparative testing between heuristics have compared their procedures against a median of three previously published heuristics; none have compared against more than 10 other procedures (even though we show in Section 4 that 30 out of 37 heuristics that we consider in this work outperformed all others on at least one problem instance in our test bed). Though Max-Cut and QUBO are considered a test bed for new ideas in heuristic design (Martí et al. 2009), the limited scope of comparative testing prevents authors from taking advantage of this by determining how their new procedure performs against a diverse set of previously published heuristics.

**Appropriate problem instances:** Standard instance libraries provide a convenient way to compare the performance of heuristics, and 69% of Max-Cut and QUBO heuristics test using all or part of the Max-Cut or QUBO standard libraries. Unfortunately, all 231 problem instances in these libraries are randomly generated, so empirical testing cannot be used to evaluate how well heuristics can be expected to perform in real-world settings. Further, the instances in standard instance libraries are homogeneous in structure. For instance, as displayed in Figure 2, the Max-Cut standard instance library only contains sparse graphs while the QUBO standard instance library only contains dense matrices (50% and above for the largest instances). The standard instance libraries are homogeneous with respect to a range of other problem instances properties (see Section 2.2), and this homogeneity harms

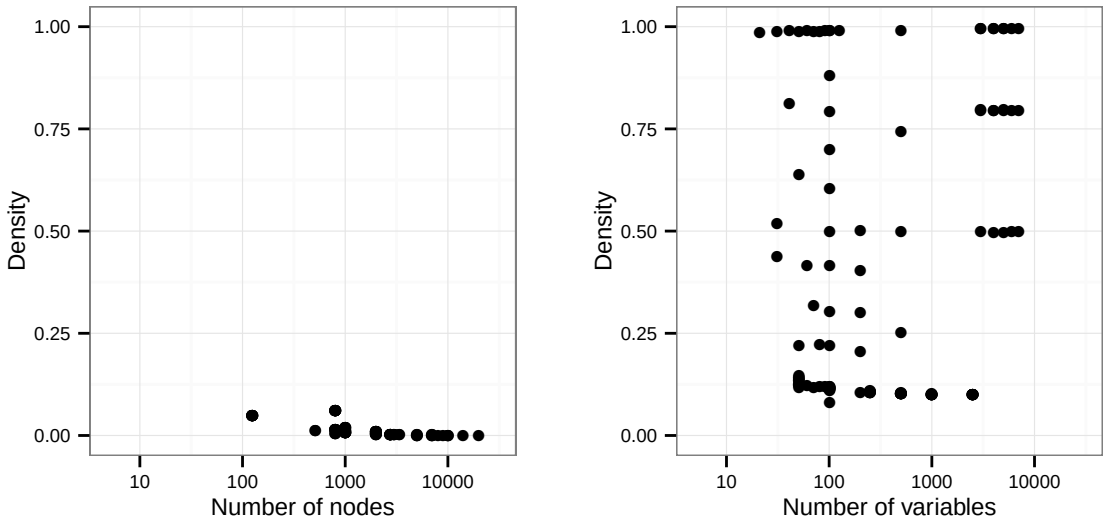


**Figure 2** **Distribution of size and density of graphs in the Max-Cut standard instance library (left) and number of variables and density of matrices in the QUBO standard instance library (right).**

the generalizability of conclusions drawn from empirical testing using these instances (see Section 4.2).

Some of these concerns have been raised about other problems in the literature. For instance, an examination of the commonly used multi-dimensional knapsack benchmark instances from the OR Library (Beasley 1990) revealed that they are homogenous in terms of metrics such as correlations between instance parameters and constraint slackness ratios, leading Hill and Reilly (2000) to develop a broader set of instances.

Why do we see these limitations in empirical testing? Though best practices for empirical testing have long been published in both the industrial engineering/OR literature (Barr et al. 1995, Rardin and Uzsoy 2001, Silberholz and Golden 2010) and the computer science literature (Johnson 2002, Eiben and Jelasity 2002, Bartz-Beielstein 2006, McGeoch 2012), these best practices are difficult to follow because researchers rarely publish source code for new heuristics. This happens for a number of reasons, including concerns about scrutiny of a paper's results, fears of being "scooped" on future papers based on the code, company rules, a status quo of heuristics researchers not publishing code, and most journals not requiring published code. As a result, fair comparison to other heuristics typically requires reimplementation or contacting authors of previous studies. While this is clearly a barrier to comparison between a large number of heuristics, it also discourages instance library

expansion because it is difficult to evaluate how previously published heuristics perform on new problem instances.

In this work, we systematically perform empirical testing of heuristics for Max-Cut and QUBO, addressing each weakness identified in current empirical testing. The first part of our work involves performing a reproducible, fair comparison of previously published heuristics over a heterogeneous problem instance database at a large scale:

- **Large-scale, heterogeneous instance library:** We perform experiments with a heterogeneous, publicly accessible instance library with 3,296 problem instances. This library combines both real-world problem instances and random problem instances from multiple random generators. No previous work on Max-Cut or QUBO heuristics has done empirical testing on more than 134 publicly accessible problem instances.
- **Open-source heuristic implementation:** We perform a systematic literature review and provide open-source implementations of 37 Max-Cut and QUBO heuristics using shared code and data structures wherever possible. Prior to this work, no empirical testing for Max-Cut or QUBO had compared a new heuristic to more than four previously published heuristics using the same hardware and termination criteria to test each heuristic.
- **Empirical testing on the cloud:** We run all implemented heuristics on all instances in the expanded instance library, giving each heuristic an identical computational budget $b(i)$ for a given instance $i$.

The second part of our work leverages our large-scale comparison of previously published heuristics:

- **Understanding heuristic performance:** Evaluation of each heuristic across a heterogeneous set of problem instances enables us to build interpretable machine learning models to identify certain types of problem instances for which a heuristic or class of heuristics performs particularly well or poorly. Interpretable models can provide useful insights to readers of papers presenting new heuristics and can also serve as a hypothesis generation step for further *scientific testing*, in which researchers run controlled experiments to identify the impact of problem instance and heuristic characteristics on heuristic performance.
- **State-of-the-art algorithm portfolio:** The performance of different heuristics across a range of problem instances can be used to train machine learning models that

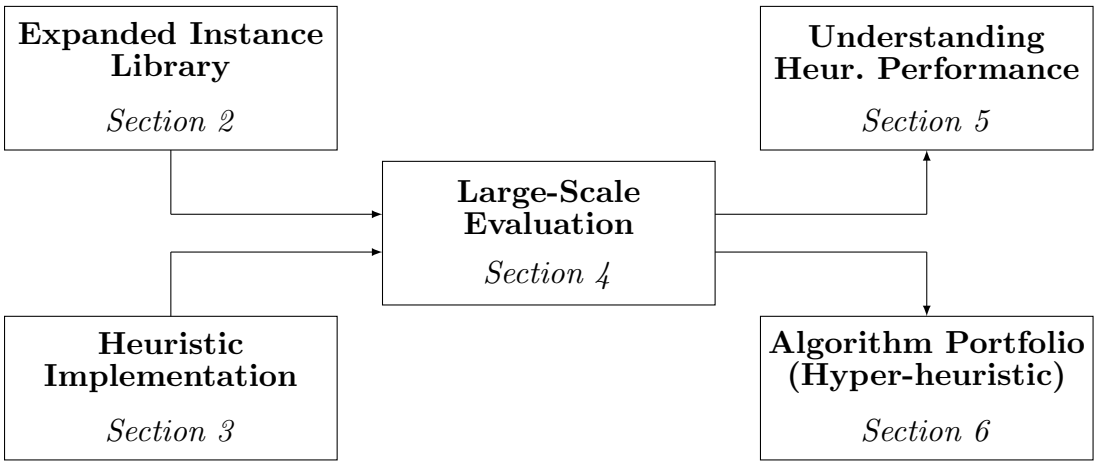


**Figure 3** **Our approach to systematic heuristic evaluation.**

predict the best-performing heuristic or set of heuristics for a given problem instance based on the instance's characteristics. Using these models we build an algorithm portfolio (or a hyper-heuristic) for Max-Cut and QUBO that selects the heuristic or set of heuristics to run for a problem instance based on the instance's properties. Our algorithm portfolio outperforms any single heuristic for these problems in out-of-sample evaluation, and we show how it can be used to improve the quality of Max-Cut solutions obtained by practitioners.

As displayed in Figure 3, we discuss the implementation and evaluation of a broad set of Max-Cut and QUBO heuristics in Sections 2–4. We then build interpretable models to understand the performance of heuristics and classes of heuristics in Section 5. In Section 6, we build an algorithm portfolio and demonstrate how it could help practitioners solve the Max-Cut problem in an image segmentation application. Finally, we discuss limitations and future directions in Section 7.

## 2. Expanded Instance Library

A key component of any empirical testing is the problem instances used for evaluation, and using an overly narrow set of problem instances might limit the generalizability of the testing as well as the ability of researchers to identify classes of problem instances for which a heuristic performs particularly well or poorly. In this section, we discuss the large-scale, heterogeneous problem instance library we constructed for Max-Cut and QUBO and provide a qualitative and quantitative comparison with the standard instance libraries for both of these problems.

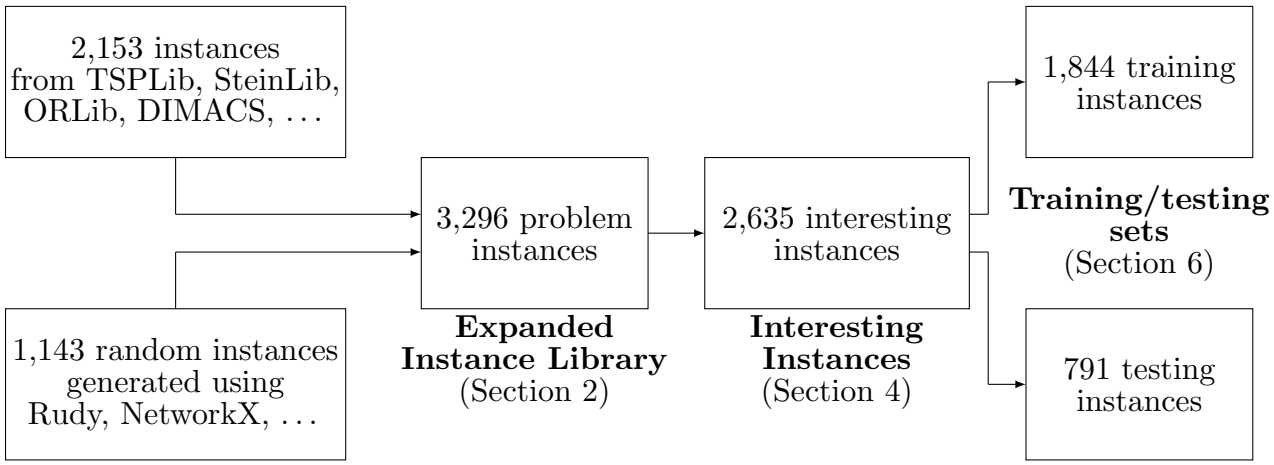


**Figure 4** **Problem instances used in this work.**

### 2.1. New Problem Instances

Given the simple reductions available between Max-Cut and QUBO, we decided to construct a single library of Max-Cut problem instances (weighted graphs) for use in evaluating all Max-Cut and QUBO heuristics. Throughout this work, QUBO heuristics are evaluated on $n$-variable QUBO matrices corresponding to the $n$-node graph instances.

To construct a diverse set of problem instances, we obtained graphs from a variety of sources. In addition to the Max-Cut and QUBO standard test beds, we obtained instances from standard libraries for other graph-based optimization problems, from programming challenges, and from Internet queries. Though an essential feature of an instance library is its ability to capture the properties of real-world instances that practitioners might encounter in practice, all instances in the Max-Cut and QUBO standard instance libraries are randomly generated, making it unclear if they generalize to real-world scenarios. By drawing from many sources, our new instance library includes real-world instances from many areas including VLSI design, telecommunication networks, and integrative biological network analysis, thereby covering a number of Max-Cut and QUBO application areas. Further, we constructed a large number of random instances using graph generators inspired by network science research (Watts and Strogatz 1998, Barabási and Albert 1999, Newman et al. 2001, 2002) that capture many structural properties of biological and social networks. In total we constructed an expanded library with 3,296 instances, as depicted in Figure 4. Details of the standard and expanded instance libraries are provided in Appendix A.

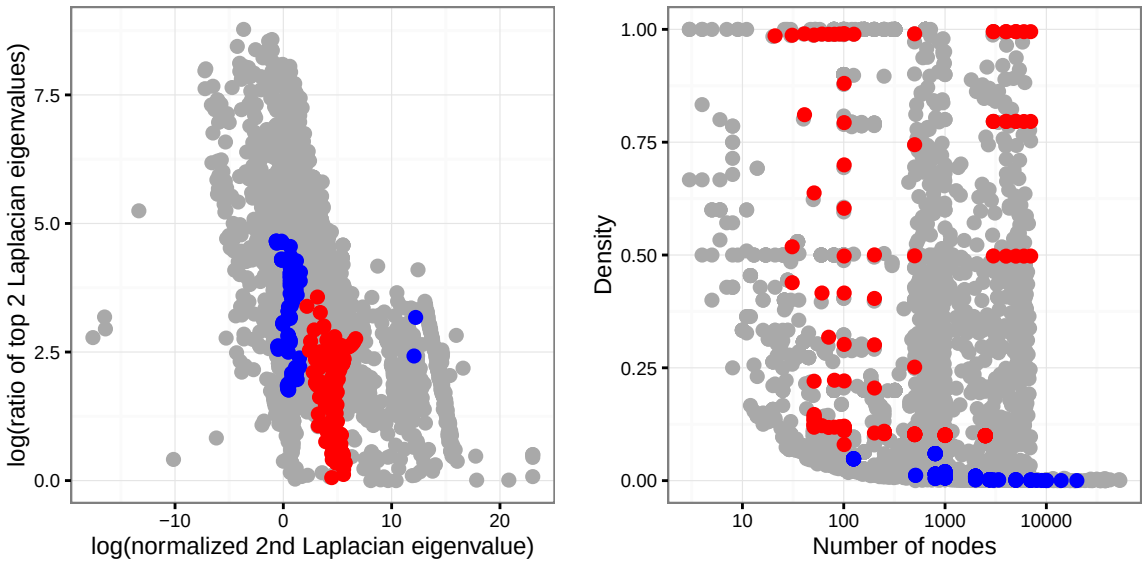


**Figure 5** **Visual comparison of the coverage of various graph metrics by the standard Max-Cut and QUBO instance libraries (blue and red points respectively) and the expanded instance library (grey points).**

### 2.2. Graph Metrics and Generalizability

Having constructed a large database of instances, we were faced with two related questions: (i) what instance metrics are important for predicting heuristic performance, and (ii) how well did we cover the space of possible instances with respect to these metrics? To answer the first question, we consider various studies that characterize hard problems by showing existence of phase transitions (Cheeseman et al. 1991, Hartmann and Weigt 2006), or by performing landscape analysis of the instance space to infer hardness (Stadler and Schnabl 1992). Such studies have shown that certain instance metrics play an important role in revealing algorithm performance, for instance, dominance features (standard deviation of the matrix entries divided by their average value) for the quadratic assignment problem (Vollmann and Buffa 1966), relation of average degree of the vertices to $k$ for the $k$-colorability problem (Krzkakała et al. 2004), extent of connectivity for the minimum vertex cover problem (Hartmann and Weigt 2003, Weigt and Hartmann 2000) and density of the graph for the Euclidean traveling salesman problem (Gent and Walsh 1996). Other metrics that have been considered to study heuristic performance on graph problems like Max-Cut, coloring, covering and network flow include number of vertices, density of edges, statistical properties of the vertex degree distribution, existence of regularity, triangle-freeness, and the eigenvalues of the graph Laplacian (Smith-Miles et al. 2010, Smith-Miles and Lopes 2011, 2012, Wang et al. 2013).

Drawing from these studies, we consider a comprehensive list of 58 fast-to-compute instance metrics (see Appendix B for details) that have been shown to be important in revealing instance hardness. Some of these metrics summarize properties of each edge or node in the graph, such as the standard deviation of normalized edge weights, while others capture global properties such as the degree assortativity[2] of the graph. While most properties are standard graph metrics, some are more creative; for instance, we include the approximate size of the maximum independent set of the graph, as computed by a fast greedy heuristic.

To answer the second question of how well we covered the instance space, we follow a similar procedure to the one used in Smith-Miles et al. (2014) by plotting the distribution of the most significant graph metrics within the set of instances. Figure 5 visually captures that the expanded instance library (gray points) includes a wider range of instances than the standard libraries (blue and red points) across the four most important metrics for predicting instance-specific heuristic performance, as discussed in Section 6. To additionally make a more quantitative comparison, we define the "coverage" of any non-binary metric $f$ by an instance $i \in \mathcal{I}$ as the range

$$c_\epsilon(f,i) = [f(i) - \epsilon, f(i) + \epsilon] \cap [0,1], \tag{1}$$

where $f(i)$ is normalized such that for all instances $i \in \mathcal{I}$ in the expanded instance library, $f(i)$ varies between 0 and 1. We define the coverage of a metric $f$ by an instance library $\mathcal{I}$ as

$$c_\epsilon(f,\mathcal{I}) = \bigcup_{i \in \mathcal{I}} c_\epsilon(f,i) \tag{2}$$

The length of $c_\epsilon(f,\mathcal{I})$ indicates the degree to which instances in $\mathcal{I}$ cover the full range of values of $f$ — lengths close to 1 indicate $\mathcal{I}$ contains a wide range of values for metric $f$, while values close to 0 indicate $\mathcal{I}$ is homogeneous for metric $f$. Limiting to instances with at least 500 nodes, the average length of metric coverage across our 45 non-binary metrics was 0.97 for our expanded instance library and 0.53 for the standard library, with $\epsilon = 0.05$ (see Figure 11 in Appendix B for a comparison of the coverage by instances in the two libraries). This difference cannot be attributed solely to the expanded library's larger size, as the expanded library has an average coverage of 0.84 (the 95% confidence interval is [0.81, 0.87]) when randomly downsampled to the same size as the standard library.

[2]Degree assortativity is the correlation between the degrees of nodes that are linked in the graph (Foster et al. 2010).

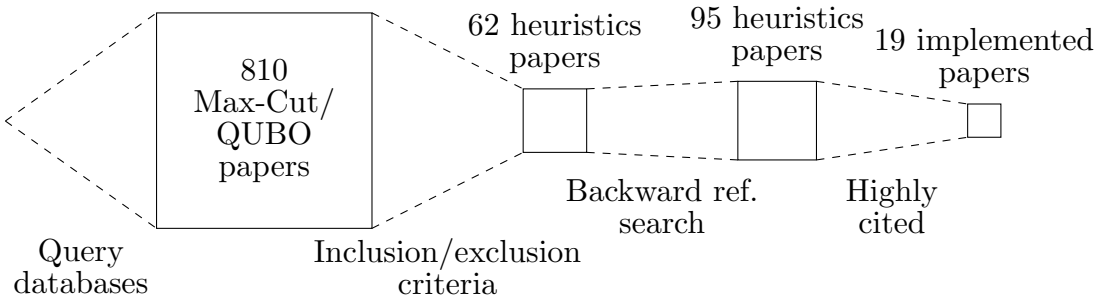


**Figure 6** **Results of the systematic review of the heuristics literature. Squares represent the number of papers at each stage in the review process, and dotted lines represent the actions taken.**

## 3. Heuristics

The next step in our approach is to implement relevant heuristics from the Max-Cut and QUBO literature, which is large and growing. To identify and evaluate various contributions in this body of work we took a *systematic review* approach, which was first popularized in medicine but is now being applied in other areas, including computing (Kitchenham et al. 2009). Key components of a systematic review include identifying a systematic and exhaustive way to search for relevant literature and the use of objective inclusion and exclusion criteria to determine which records are included in the review.

We first constructed a search query to find all English-language publications from 2013 or earlier indexed in the Compendex and Inspec databases that mention either the Max-Cut or QUBO problems. From this initial set of 810 papers, we selected 62 papers that describe at least one heuristic with an element of randomness, don't rely on specialized software (e.g. a semidefinite program solver), and perform empirical testing. We performed a *backward reference search* from this set of papers to obtain a final set of 95 papers, from which we selected 19 of the most highly cited papers that described a total of 37 heuristics (10 for Max-Cut, 27 for QUBO). This process is summarized in Figure 6, and further details about the search process and identified heuristics are presented in Appendix C, along with scaling analyses of their memory consumption.

While code is available for some of these heuristics, the majority are described only in pseudocode in their respective papers. To enable evaluation over the new instance library, we reimplemented all 37 heuristics in a high-performance language (`C++`), sharing code and data structures between the heuristics wherever possible. Examples of shared components include in-memory representation of problem instances and solutions as well as common operations such as one-swap local search and efficient updates of objective values when

changing the value of one node/variable (see Appendix C for details). Heuristic parameters were set to the values used in empirical testing in their respective papers, and we leave parameter tuning for the new instance library (which could be challenging due to its large size) as an area of future research. All the implementations are available as open-source code at `https://github.com/MQLib/MQLib`.

While we tried to implement all heuristics exactly as described in their respective papers, we made some modifications to enable fair heuristic comparison. We use a runtime-based termination criterion to ensure fair comparison, so we require that all algorithms be able to be run indefinitely with *a chance of continuing to make progress*. When run indefinitely, some heuristics may get stuck at a local optimum; for example, a simulated annealing algorithm may eventually cool to the point where no more progress is likely no matter how much longer it is run. Many heuristics address this situation by using mechanisms like random restart whenever no further local improvement is likely or when some degree of convergence is reached in a population of solutions. We modified the remaining heuristics to enable indefinite execution with the chance of making progress by wrapping them in random restarts. These modifications are detailed in Appendix C and more importantly in the code itself, which can be critically reviewed and improved by the community over time thanks to its open-source nature.

## 4. Large-Scale Evaluation

Given a set of implemented heuristics and a collection of instances, we can now move to the empirical testing of each heuristic. Here we describe our testing methodology, which enables fair, reproducible heuristic comparison by evaluating each heuristic with the same termination criteria on the same cloud-computing hardware, and we present comparative results across the 37 heuristics tested.

### 4.1. Evaluation Methodology

As stated in Section 1, more than 80% of heuristic comparisons in the literature do not give each heuristic the same computational budget, often leaving readers unsure if one procedure outperformed another because it is more effective or because it was run for longer. To enable fair comparison, we give each heuristic the same instance-specific runtime limit. For a given instance, the runtime limit was set to the amount of time needed to generate 1,500 random solutions and to repeatedly apply a popular local search procedure

(`AllFirst1Swap`; see Appendix C) until local optimality. Because for most heuristics the majority of computational effort is spent on local search, we believe this runtime limit captures the relative difficulty of different problem instances for the heuristics being evaluated. An advantage of defining the runtime limit in this way is that new heuristics can be evaluated using different hardware by re-computing the runtime limit for the instance on that hardware.

All empirical testing was performed using cloud computing resources, which provides a practical benefit over private computing clusters for some users: A large amount of computing power can be requisitioned on demand from a cloud computing provider without any contention with other users for resources. We performed heuristic evaluation using 60 `m3.medium` Ubuntu machines from the Amazon Elastic Compute Cloud (EC2), which have Intel Xeon E5-2670 processors and 4 GiB of memory; code was compiled with the `g++` compiler using the `-O3` optimization flag. We evaluated each of the 37 heuristics over all 3,296 problem instances in the expanded instance library, consuming 2.0 CPU-years of processing power (20.1 CPU-days per heuristic), taking 12.4 days over the 60 machines (8.0 hours per heuristic), and costing $1,196 ($32.3 per heuristic). In experiments we evaluated each heuristic 5 times for each problem instance with different random seeds.

### 4.2. Comparing heuristic performance

The large-scale evaluation provides us with a solution for each of the 37 heuristics on each of the 3,296 instances in the expanded library. On some instances, particularly small ones, all or nearly all heuristics had the same objective value, meaning these instances cannot be used to differentiate heuristics. We take the view expressed by McGeoch (2012) and others that presenting results on "uninteresting" instances has little utility, and all results in the remainder of this paper were computed using the 2,635 instances for which no more than half of the 37 heuristics attained the best solution. As expected, small instances were much more likely to be uninteresting: 65% of instances with fewer than 100 nodes were uninteresting, compared to 10% of instances with 100 or more nodes.

Given this set of interesting instances, we can define metrics for measuring and comparing heuristic performance. We chose seven natural measures of heuristic performance, which can be computed for a heuristic $h$ in the set of all heuristics $\mathcal{H}$ using objective value $x^k_{h,i}$ for each replicate $k$ of $h$ on every instance $i$ from the set of all 2,635 interesting instances $\mathcal{I}$:

- The **worst-of-5 deviation** of a heuristic $h$, $WD(h) := |\mathcal{I}|^{-1} \sum_{i \in \mathcal{I}} \frac{\min_{k \in \{1,\ldots,5\}} x^k_{h,i}}{\max_{r \in \mathcal{H}, k \in \{1,\ldots,5\}} x^k_{r,i}}$, is the average percentage deviation (averaged across the set of interesting instances) of the worst solution among the five replicates for $h$. Percentage deviation of a solution for instance $i$ is computed using the best solution obtained by any heuristic on $i$.
- The **mean-of-5 deviation** and **best-of-5 deviation** of a heuristic $h$, $MD(h) := |\mathcal{I}|^{-1} \sum_{i \in \mathcal{I}} \frac{\sum_{k=1}^5 x^k_{h,i}/5}{\max_{r \in \mathcal{H}, k \in \{1,\ldots,5\}} x^k_{r,i}}$ and $BD(h) := |\mathcal{I}|^{-1} \sum_{i \in \mathcal{I}} \frac{\max_{k \in \{1,\ldots,5\}} x^k_{h,i}}{\max_{r \in \mathcal{H}, k \in \{1,\ldots,5\}} x^k_{r,i}}$, respectively, are the average percentage deviations of the mean and best solutions of the five replicates of $h$.
- The **first-equal percentage** of a heuristic $h$, $FE(h) := |\{i \in \mathcal{I} : \sum_{k=1}^5 x^k_{h,i} = \max_{r \in \mathcal{H}} \sum_{k=1}^5 x^k_{r,i}\}|/|\mathcal{I}|$, is the percentage of interesting instances for which the mean solution of $h$ across the five replicates was no worse than the mean solution of any of the 36 other heuristics.
- The **first-strict percentage** of a heuristic $h$, $FS(h) := |\{i \in \mathcal{I} : \sum_{k=1}^5 x^k_{h,i} > \max_{r \in \mathcal{H} \setminus \{h\}} \sum_{k=1}^5 x^k_{r,i}\}|/|\mathcal{I}|$, is the percentage of interesting instances for which the mean solution of $h$ across the five replicates outperformed the mean solution of any of the 36 other heuristics.
- The **best achieved percentage** of a heuristic $h$, $BA(h) := |\{i \in \mathcal{I} : \max_{k \in \{1,\ldots,5\}} x^k_{h,i} = \max_{r \in \mathcal{H}, k \in \{1,\ldots,5\}} x^k_{r,i}\}|/|\mathcal{I}|$, is the percentage of interesting instances for which the heuristic achieved the best-known heuristic solution in at least one replicate.
- The **average rank** of a heuristic $h$ is defined as $AR(h) := |\mathcal{I}|^{-1} \sum_{i \in \mathcal{I}} R(h,i)$, where $R(h,i)$ is the rank of the mean-of-5 deviation of heuristic $h$ on instance $i$ among the mean-of-5 deviations of all 37 heuristics in $\mathcal{H}$ on $i$. The mean-of-5 deviation of heuristic $h$ on instance $i$ is defined as $\frac{\sum_{k=1}^5 x^k_{h,i}/5}{\max_{r \in \mathcal{H}, k \in \{1,\ldots,5\}} x^k_{r,i}}$. In the case of ties, all heuristics are given the minimum rank. Rank 1 indicates the best performance among all heuristics, and rank 37 indicates the worst performance.

While it would be preferable to compute the mean deviation from the optimal solution for an instance instead of from the best-known heuristic solution for that instance, this is not practical for our analysis due to the large number of interesting instances and their sizes. However, results from a subset of the instances suggest the best-known heuristic solutions may be of high quality — the best-known heuristic solution was optimal for all 29 QUBO standard test bed problem instances with optimal solutions reported in Rendl et al. (2010) or Boros et al. (2016), which range in size from 100–250 variables.

| | Instances with Top Performance (%) | | | Deviation from Best Solution (%) | | | Avg. Rank |
|---|---|---|---|---|---|---|---|
| Heuristic | First-Equal (Mean-of-5) | First-Strict (Mean-of-5) | Best Achieved | Worst-of-5 Deviation | Mean-of-5 Deviation | Best-of-5 Deviation | |
| BUR02 | **22.9** | **16.2** | 32.7 | **0.5** | **0.3** | **0.2** | 10.7 |
| FES02GVP | 21.3 | 4.1 | 29.9 | 0.6 | 0.5 | 0.4 | 10.1 |
| PAL04T3 | 19.4 | 2.3 | 30.5 | 0.8 | 0.6 | 0.5 | **7.5** |
| FES02GP | 19.3 | 5.5 | 27.1 | 0.9 | 0.7 | 0.6 | 14.2 |
| FES02GV | 18.0 | 1.5 | 26.0 | 0.8 | 0.7 | 0.6 | 11.2 |
| PAL04T2 | 17.0 | 8.1 | **33.3** | 0.9 | 0.6 | 0.3 | 8.5 |
| BEA98TS | 16.4 | 5.0 | 22.4 | 2.6 | 2.4 | 2.1 | 16.6 |
| LU10 | 14.9 | 2.7 | 23.9 | 1.7 | 1.5 | 1.2 | 13.1 |
| FES02G | 14.5 | 2.6 | 19.6 | 1.4 | 1.3 | 1.1 | 18.2 |
| MER04 | 14.4 | 3.2 | 24.4 | 0.7 | 0.6 | 0.5 | 8.6 |
| PAL04T1 | 14.0 | 2.7 | 21.7 | 2.4 | 2.2 | 1.9 | 15.2 |
| MER99LS | 12.8 | 6.9 | 30.9 | 0.7 | 0.5 | 0.3 | 10.0 |
| MER02GRK | 12.6 | 0.6 | 19.2 | 0.8 | 0.6 | 0.5 | 11.9 |
| MER02LSK | 11.3 | 0.4 | 19.3 | 1.0 | 0.8 | 0.7 | 13.3 |
| PAL04T5 | 9.6 | 2.4 | 21.1 | 3.0 | 2.5 | 2.0 | 18.6 |
| PAL06 | 9.4 | 0.9 | 21.9 | 2.4 | 1.9 | 1.4 | 17.3 |
| ALK98 | 7.5 | 2.9 | 15.6 | 0.8 | 0.6 | 0.5 | 12.5 |
| PAL04T4 | 7.3 | 0.2 | 18.4 | 3.8 | 3.2 | 2.6 | 21.7 |
| GLO10 | 6.1 | 0.3 | 21.7 | 2.2 | 1.7 | 1.3 | 16.6 |
| FES02V | 6.0 | 0.3 | 15.1 | 1.3 | 1.1 | 0.9 | 13.8 |
| KAT00 | 6.0 | 0.1 | 16.2 | 1.5 | 1.2 | 0.9 | 16.9 |
| FES02VP | 5.8 | 0.1 | 16.1 | 1.1 | 0.9 | 0.7 | 12.4 |
| PAL04MT | 5.1 | 0.2 | 16.8 | 4.0 | 3.4 | 2.8 | 24.1 |
| MER02GR | 3.5 | 0.0 | 5.7 | 3.2 | 3.0 | 2.8 | 24.3 |
| GLO98 | 2.5 | 0.1 | 8.1 | 2.1 | 1.8 | 1.5 | 23.4 |
| HAS00TS | 2.2 | 0.4 | 7.2 | 1.6 | 1.3 | 1.1 | 20.3 |
| HAS00GA | 2.1 | 0.2 | 10.6 | 2.2 | 1.9 | 1.6 | 20.9 |
| MER02LS1 | 1.8 | 0.0 | 6.3 | 3.0 | 2.8 | 2.6 | 25.1 |
| PAR08 | 1.7 | 0.1 | 11.2 | 3.0 | 2.5 | 2.1 | 17.9 |
| KAT01 | 1.6 | 0.6 | 4.0 | 2.7 | 2.4 | 2.1 | 26.3 |
| DUA05 | 0.8 | 0.2 | 6.1 | 2.1 | 1.7 | 1.4 | 17.4 |
| LOD99 | 0.7 | 0.0 | 3.4 | 5.7 | 5.3 | 4.9 | 30.2 |
| LAG09HCE | 0.4 | 0.3 | 4.1 | 2.3 | 1.9 | 1.6 | 20.4 |
| BEA98SA | 0.4 | 0.0 | 1.3 | 4.8 | 4.0 | 3.2 | 30.8 |
| MER99CR | 0.0 | 0.0 | 0.0 | 15.9 | 14.4 | 12.9 | 34.6 |
| MER99MU | 0.0 | 0.0 | 0.0 | 24.8 | 23.3 | 21.7 | 35.7 |
| LAG09CE | 0.0 | 0.0 | 0.0 | 30.8 | 30.1 | 29.2 | 36.9 |

**Table 1 Summary of results for each heuristic across the 2,635 interesting instances. The best heuristic under each metric is marked in bold.**

The performance of the selected 37 heuristics on these seven measures are presented in Table 1. The best overall heuristic on the expanded instance library with respect to the performance of its mean solution across the five replicates for each instance was Max-Cut heuristic BUR02, which not only matched the best performance on 22.9% of instances but also had strictly better performance than any other heuristic on 16.2% of instances and a mean deviation of only 0.3%. Heuristics showed significant differences in their variability to seed; for instance, QUBO tabu search procedure PAL04T2 nearly matched BUR02's best-of-5 deviation but had significantly higher mean-of-5 and worst-of-5 deviations. As no heuristic matched the best mean-of-5 performance on more than 22.9% of problem instances and 30 of the 37 outperformed all other heuristics on at least one instance, no single heuristic dominated all the others across the instance library. This motivates our approach in Section 6, in which we use machine learning to predict which heuristic will achieve the best solution for a given problem instance.

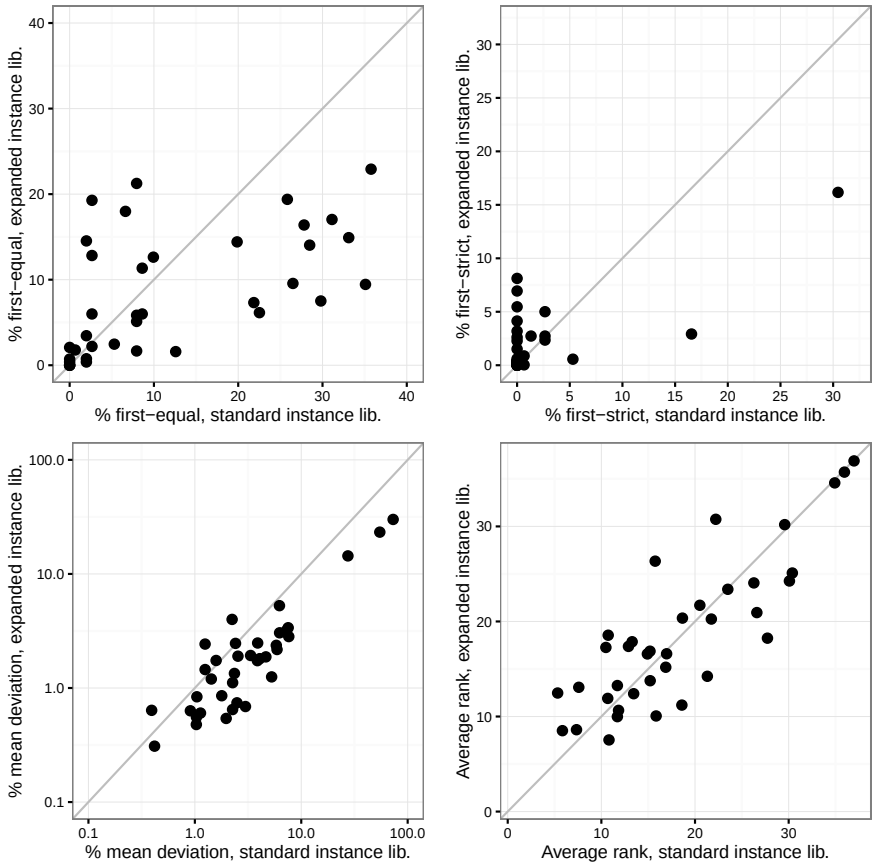


**Figure 7** **Comparison of the performance of the 37 heuristics across the "interesting" instances in the standard and expanded instance libraries. (Top left) Percentage of instances for which each heuristic matches the best solution over the 37 heuristics. (Top right) Percentage of instances for which each heuristic is strictly best (outperforms all other heuristics). (Bottom left) Mean deviation percentage from the best solution. (Bottom right) Average rank of the mean-of-5 deviation of each heuristic.**

As depicted in Figure 7, results on the standard instance library did not always generalize well to the broader set of problem instances used in this work. For instance, MER99LS, a genetic algorithm for QUBO, was the sole best-performing heuristic on 6.9% of instances on the expanded library (the third highest first-strict rate), despite not being the sole best-performing heuristic on a single instance in the standard library and only attaining the best solution on 2.6% of standard library instances (rank 23/37). In total, 23 heuristics had first-strict value 0 on the standard instance library but a positive first-strict value on the expanded library, and three of those heuristics had a first-strict value exceeding 5% on the expanded library.

## 5. Interpreting Heuristic Performance

The results presented to this point represent a large-scale, fair comparison between previously published heuristics, a necessary step to identify state-of-the-art procedures. However, aggregate error measures commonly used in comparative evaluation like mean deviation from the best-known solution provide no insight into where heuristics perform particularly well or poorly, and therefore cannot be used to address the title question "what works best when?" As a result, some have argued that it is equally or more important to evaluate heuristics using scientific testing (Hooker 1995, Johnson 2002, McGeoch 2012), in which a controlled experiment is performed to quantify the impact of one or more problem instance or heuristic characteristics on performance.

In this section, we use techniques that exploit the rich set of instances and heuristics we have collected to identify what works well when. For individual heuristics, we use interpretable machine learning models (regression trees) to identify types of problem instances for which a given heuristic performs particularly well or poorly, which could further be used to generate hypotheses for scientific testing. Further, we identify types of problem instances for which broader classes of heuristics perform particularly well or poorly, gaining insights into the benefits and drawbacks of different heuristic ideas. Such ideas have been recently popular in the literature through the study of "algorithmic footprints," which identify parts of the instance space where an algorithm performs well or poorly (Corne and Reynolds 2010, Smith-Miles and van Hemert 2011). Various studies have employed different techniques for dimension reduction of the feature space (to identify regions of performance visually) like self-organizing maps (Insani et al. 2013, Smith-Miles and van Hemert 2011) and principal component analysis (Smith-Miles et al. 2014). These approaches identify regions in the reduced space where one heuristic performs better than the other. Interpretable models like decision trees have been used to generate rules to decide which heuristic variant of the Lin-Kernighan heuristic works best for a given instance of the traveling salesman problem (Smith-Miles and van Hemert 2011) and for deciding between two heuristics for the single machine scheduling problem (Smith-Miles et al. 2009), to name a few. In this work, we build heuristic-specific models that explain the most significant instance properties that affect heuristic performance. An advantage of this approach is that results are presented in the space of the original instance features instead of in a less interpretable

dimensionally reduced space, though a disadvantage of this approach is that different features may be more important in predicting the performance of different heuristics, making it harder to directly compare the footprints of different heuristics.

### 5.1. Understanding where a heuristic performs well or poorly

To identify types of problem instances for which a heuristic $h$ performs well or poorly, we label its performance on each of the 2,635 interesting instances identified in Section 4 using the rank of its mean-of-5 deviation $R(h, i)$ on each instance $i$. As in Section 4.2, in the case of ties in mean-of-5 deviation we assign the smallest rank for all tied heuristics. As there are 37 heuristics, ranks take values in the range 1–37.

For each heuristic, we fitted a CART model (Breiman et al. 1994) to predict the rank of that heuristic on each instance based on the instance metrics as defined in Section 2, using the `rpart` package in R (Therneau et al. 2015). To ensure simple fitted models, we limited trees to maximum depth 3 and required each leaf to contain at least 100 observations. Fitted trees had average $R^2$ of 0.51 (range: 0.19–0.78).

For any heuristic, we can build a regression tree to gain insight into the relationship between problem instance characteristics and the rank of the heuristic on that instance. While this provides useful information about all heuristics, of particular interest are pairs of heuristics for which the same instance characteristics are predictive of performance, such as a pair of heuristics where the same variables (whatever they might be) are used in splits near the top of the regression tree. For such heuristics, we can visualize areas of the problem instance space where both heuristics perform well, where both perform poorly, or where one performs well while the other performs poorly. Figure 8 presents such a visualization for GLO98 (top) and PAR08 (bottom), both tabu search procedures for QUBO. The two heuristics had the highest $R^2$ amongst any CART models that used the same two variables for their initial splits ($R^2 = 0.54$ for GLO98 and 0.61 for PAR08), excluding pairs of heuristics from the same paper. PAR08 has its best performance on instances with a large number of nodes and a small maximum independent set size (normalized by the number of nodes in the instance). Meanwhile, GLO98 has its worst performance on those problem instances and its best performance on problem instances with a small number of nodes. These differences in the types of instances where each heuristic performs best motivates the algorithm portfolio approach taken in Section 6.

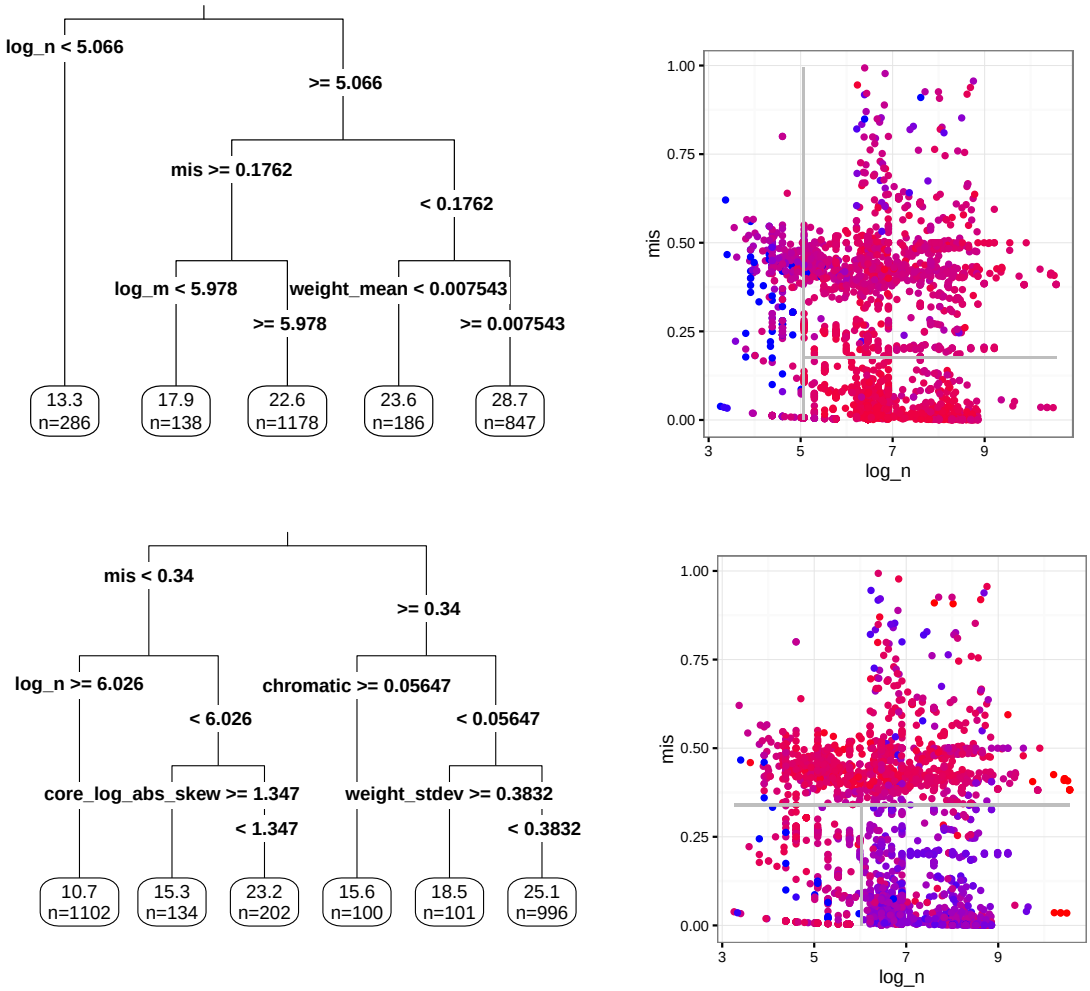


**Figure 8** **Interpretable models identifying the instances on which GLO98 (top) and PAR08 (bottom) perform particularly well or poorly. (Left) A regression tree that predicts the heuristic's rank (range: 1–37) for each of the "interesting" problem instances in the expanded instance library. (Right) The distribution of rank across the two problem instance characteristics identified as most important in the regression trees; blue indicates the heuristic performed well (rank near 1) and red indicates the heuristic performed poorly (rank near 37).**

We include the CART models for the remaining 35 heuristics considered in this study in the supplementary material, and we believe that heuristic papers that develop new heuristics would be enhanced by presenting results of this nature.

Interpretable machine learning models can help solve a key challenge of scientific testing: identifying hypotheses to test (Hooker 1995). For instance, the regression trees in Figure 8 suggest further controlled testing to assess the impact of the maximum independent set size versus the number of nodes on the performances of GLO98 and PAR08.

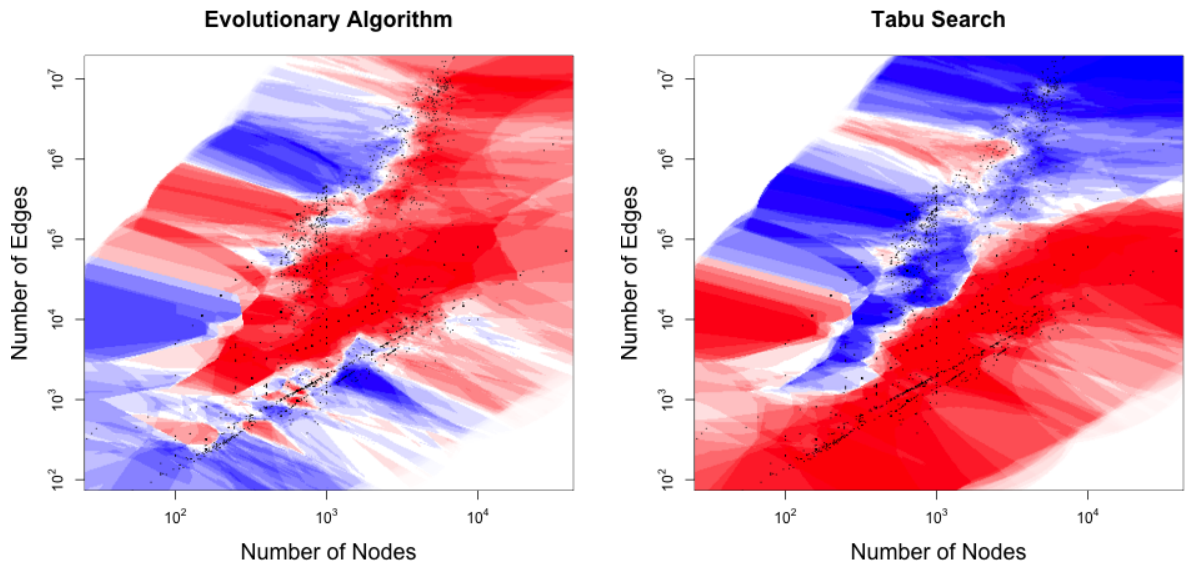


**Figure 9** **Performance of evolutionary algorithms (left) and tabu search procedures (right) on the 1,878 instances for which only one heuristic attains the best solution, broken down by instance node and edge counts. Blue regions indicate instances for which the specified heuristic class performed best, and red regions indicate instances for which the heuristic class did not perform best.**

### 5.2. Understanding where heuristic ideas perform well or poorly

Many ideas have been tested in heuristic development, with inspiration ranging from natural selection to relaxations of exact algorithms to local searches to random restarts. In this section, we consider classes of heuristics that use the same subroutines or idea, for example tabu searches or genetically evolving populations of candidate solutions. Note that this view is different from an algorithm footprint as we consider classes of algorithms (instead of a single algorithm) to identify instances spaces where some styles of heuristics perform better than others. In this section, we identify instance spaces where certain heuristic ideas perform well or poorly by analyzing the results of all heuristics that use a particular idea.

Using the 1,878 instances for which only one heuristic attains the best mean-of-5 deviation, we investigate the sorts of problem instances for which evolutionary algorithms and tabu search procedures outperform all other heuristics. These two classes of heuristics were selected because they are broad, popular classes of heuristics and because they both have a significant number of instances for which they gave the sole best solution. In Figure 9 we capture types of instances for which each class of heuristics tends to perform best, breaking down instances by number of nodes and edges. Points in the plot for each heuristic class are colored on a gradient from blue to red based on the proportion of the 20 nearest

problem instances for which a heuristic from that class was the best-performing heuristic (nearest instances are defined with respect to the Euclidean distance in the 2-dimensional space of the log of the number of nodes and the log of the number of edges in the instance). Dark blue regions of Figure 9 represent instances for which evolutionary algorithms (left) or tabu search procedures (right) are best, dark red regions indicate other types of procedures performed best, and colors in between dark blue and dark red indicate a mix of best-performing heuristic classes. Additionally, colors in the plot lie on a gradient from the blue/red mix to white depending on the proportion of the 20 nearest instances that are within distance 1 in the instance space: Points with no instances within distance 1 are completely white.

Instance density has a profound impact on the performance of tabu search procedures, as they are often the top procedure on dense graphs but rarely for sparse ones. The story is more complex for evolutionary algorithms: They perform best on sparse and dense mid-size instances, but non-evolutionary algorithms seem to succeed on the largest instances.

Many such plots can be produced for various heuristic components and problem instance characteristics, providing insights into the impact of those ideas on heuristic performance.

## 6. Algorithm Portfolio

In Section 4, we saw that no single heuristic outperformed all others across the expanded problem instance library, and in Section 5 we saw that problem instance characteristics can be used to differentiate where a heuristic will likely perform well or poorly. The natural next step is to construct a method that selects the heuristic or set of heuristics to use for a given instance based on that instance's characteristics, termed an *offline learning heuristic selection hyper-heuristic* (Burke et al. 2010) or an algorithm portfolio. This is also referred to in the literature as the *algorithm selection problem*, and has been used to create solvers for satisfiability testing (Xu et al. 2008, Nikolić et al. 2013), quantified Boolean formulas (Pulina and Tacchella 2009), constraint programming (O'Mahony et al. 2008), and planning problems (Howe et al. 2000). Unlike many hyper-heuristics found in the literature, our procedure does not change the heuristic used during the solution process, nor does it construct a heuristic by selecting from a set of heuristic components. We show that our algorithm portfolio outperforms each of the 37 Max-Cut and QUBO heuristics it combines, and we apply it to an important problem in vision: image segmentation.

### 6.1. Constructing an algorithm portfolio

There is significant multi-disciplinary literature on creating algorithm portfolios. In 1976, Rice defined the algorithm selection problem as learning the mapping from instance features to the best algorithm to run on an instance (Rice 1976). This framework is simple yet very general, and multiple studies in the operation research (OR), artificial intelligence, and machine learning literatures have been viewed under the same lens (Smith-Miles 2008). On one hand, the machine learning community views the algorithm selection problem as a learning problem, and various studies have used regression (Leyton-Brown et al. 2002, 2003, Xu et al. 2008), Bayesian networks (Horvitz et al. 2001), k-nearest neighbor rankings (Brazdil et al. 2003), neural networks (Smith et al. 2001), and decision trees (Ali and Smith 2006, Vilalta and Drissi 2002) to predict the performance of a set of algorithms given a set of instance features. Meanwhile, the OR community has focused more on identifying phase transitions and on landscape analysis to develop deeper insights into the relationship between instances and algorithms (Stützle and Fernandes 2004, Smith-Miles 2008). In recent years, however, there has been a greater cross-pollination of ideas, and the best algorithm for a given set of instances for various OR problems like the traveling salesman problem (Smith-Miles and van Hemert 2011), timetabling (Smith-Miles and Lopes 2011), quadratic assignment problem (Smith-Miles et al. 2010), and job shop scheduling problem (Smith-Miles et al. 2009) has been prescribed using machine learning algorithms.

We take the machine learning approach to the algorithm selection problem, attempting to select a heuristic or set of heuristics that attain the best expected solution quality when run in parallel on a given problem instance. To do so, we use the following approach:

- We first build a *random forest model* (Breiman 2001) for each of the 37 heuristics that predicts whether the heuristic will achieve the highest mean-of-5 solution for a given instance, using the 58 instance characteristics from Section 2.2 as predictor variables. Each of these models performs a binary classification. Random forest models for classification are ensemble learning models in which multiple (`ntree`, a parameter) classification trees are built with training sets that are randomly sampled with replacement from the original training set. When building these classification trees, each split is made by randomly sampling a subset of the variables (of size `mtry`, a parameter) and limiting the split to one of those variables. Splitting is terminated by the specified minimum size of a leaf (`nodesize`, a parameter). In practice, random forests often have better out-of-sample predictive performance than classification trees, motivating their use for our algorithm portfolio.

- For a new problem instance, we obtain the predicted probability that each of the 37 heuristics will achieve the best solution using the random forest models. In a random forest for classification, the predicted probability that an observation is of some class is the proportion of trees predicting that class for that observation.
- If one heuristic is desired, we select the heuristic with the highest predicted probability, breaking ties randomly, and return the solution of that heuristic. If $k$ heuristics are desired, we select the $k$ heuristics with the highest predicted probabilities, run them in parallel, and report the best solution obtained among the $k$ heuristics for the instance.

To validate this method works well on unseen instances, we use a standard training-testing split approach in which we build our random forest models using only the training set and evaluate them using only the testing set. We split the 2,635 "interesting" instances into a training set of size 1,844 (70%) and a testing set of size 791 (30%). All random forest models were fitted using the R `randomForest` package (Liaw and Wiener 2002). We selected the number of metrics used in each split when constructing the random forest (the `mtry` parameter) using five-fold cross-validation on the training set, and we used the default parameter values from the `randomForest` package, setting `ntree`, the number of trees to build, to 500 and `nodesize`, the minimum number of observations in leaves, to 1.

Suppose for a given test-set problem instance the algorithm portfolio described above selects a set $\mathcal{H}_s$ of heuristics to run ($\mathcal{H}_s \subseteq \mathcal{H}$). Let $X_h$ for $h \in \mathcal{H}_s$ be a random variable indicating the percentage deviation of $h$ from the best-known solution for that instance; we model $X_h$ to take the percentage deviation of a (uniform) randomly selected replicate of $h$ on the instance. The portfolio returns the best solution returned by any heuristic $h \in \mathcal{H}_s$; the selected solution has percentage deviation $\min_{h\in\mathcal{H}_s} X_h$. We assess the quality of the selected set $\mathcal{H}_s$ using the expected percentage deviation from the best-known heuristic solution, $\mathbb{E}[\min_{h\in\mathcal{H}_s} X_h]$, and the probability that the best-known solution is returned by the algorithm portfolio, $\mathbb{P}[\min_{h\in\mathcal{H}_s} X_h = 0]$.[3] Given the heuristics selected for each test-set

[3] As an illustrative example, assume that on a given problem instance heuristic A's five replicates had percentage deviations {0%, 0%, 0%, 1%, 1%} from the best-known solution, while heuristic B's five replicates had percentage deviations {0%, 0%, 2%, 2%, 2%}. If only heuristic A is run on the instance then there is a 60% probability the best-known solution is returned and a 40% chance a solution with deviation 1% is returned, yielding expected deviation 0.4%. If both heuristics are run in parallel, then the percent deviations of A and B will take values (0%, 0%), respectively, with probability 24%, (0%, 2%) with probability 36%, (1%, 0%) with probability 16%, and (1%, 2%) with probability 24%. This 2-heuristic algorithm portfolio returns the best-known solution 76% of the time and has deviation 1% the remainder of the time, meaning it has expected deviation 0.24%.

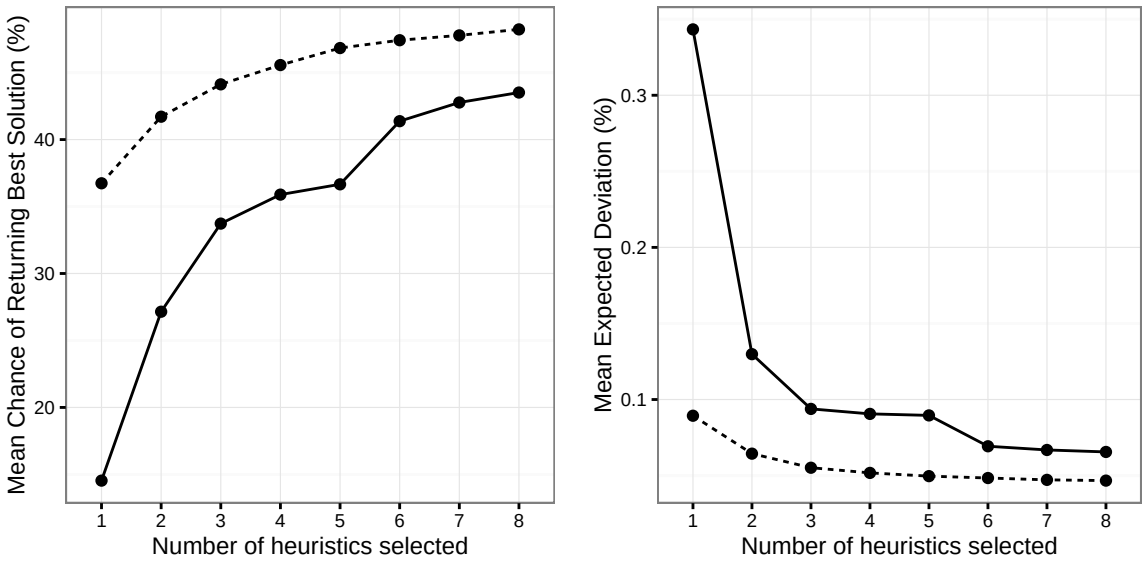


**Figure 10** **A comparison of the performance of $k$ parallel heuristics, either selected by the algorithm portfolio (dashed line) or by selecting the $k$ heuristics with the highest number of best solutions for training-set instances (solid line). (Left) The percentage of test-set instances for which one of the parallel heuristics found the best solution. (Right) The average deviation from the best solution.**

instance by the algorithm portfolio, we use the mean expected deviation and mean probability of achieving the best-known solution to evaluate the performance of the algorithm portfolio over the test set.

When configured to select a single heuristic for each problem instance, the algorithm portfolio selected 27 of the 37 heuristics for at least one test-set problem instance, achieving a mean probability of obtaining the best solution of 37% and a mean expected deviation of 0.09%. This represents an improvement over running BUR02 over all test-set instances, where it achieved a mean probability of obtaining the best solution of 15% and a mean expected deviation of 0.34%. As a result, the algorithm portfolio represents a state-of-the-art procedure, outperforming any of the 37 heuristics it combines. Figure 10 reports the performance of an algorithm portfolio running $k$ heuristics in parallel, compared to the naive approach of always running the $k$ heuristics that attained the highest number of best solutions on the training set. For all $k \in \{1, \ldots, 8\}$, the algorithm portfolio obtains the best solution on a higher proportion of instances and a lower mean deviation. Running with eight heuristics in parallel, the algorithm portfolio achieved a mean probability of obtaining the best solution of 48% and a mean expected deviation of 0.05%.

To assess the importance of each instance feature in identifying where different heuristics perform best, we computed the *mean decrease in Gini impurity* variable importance

measure for every fitted random forest model used in the algorithm portfolio (Breiman et al. 1994). The four most important instance features in terms of the average variable importance rank across the random forest models were: log of the number of nodes, log of the number of edges, log of the normalized second eigenvalue of the graph Laplacian, and log of the ratio of the first eigenvalue of the graph Laplacian to the second eigenvalue. These instance features had average rank 7.9, 8.1, 8.5, and 10.0, respectively, in the ranked lists of variable importance. Meanwhile, 12 features were not in the top 10 most important features for any random forest model, indicating they might be removed without significant loss in algorithm portfolio performance. The full summary of variable importance measures is reported in Appendix D.

### 6.2. Application to image segmentation

Our algorithm portfolio provides practitioners with access to a state-of-the-art procedure for Max-Cut and QUBO, and in this section we investigate whether it could improve the quality of heuristics used in applied work. In recent work, de Sousa et al. (2013) developed a new Max-Cut heuristic for use in an image segmentation application. In this application, an image is divided into regions of adjacent pixels with similar colors, and each region is represented by a node in the Max-Cut input graph. Nodes corresponding to adjacent image regions are connected by edges with edge weights that capture similarity of the average colors in these regions — edges linking regions with very dissimilar average colors are assigned larger weights. After solving Max-Cut on the graph corresponding to an image, regions that are not separated by the cut are merged into a single larger region, yielding an image segmentation.

To test the usefulness of our algorithm portfolio (limited to selecting one heuristic per instance) to practitioners in this application area, we ran it on 100 randomly selected images from the training set of the Berkeley Segmentation Dataset and Benchmark 300 (Martin et al. 2001), the image segmentation database used in de Sousa et al. (2013) to test their procedure. To evaluate how our algorithm portfolio would perform in a fully out-of-sample setting (which is how it would be applied in a new application area), we did not use any of the above-mentioned image segmentation problem instances in the training set of the algorithm portfolio. Instead, we used the same portfolio trained on 1,844 instances that is described in Section 6.1 for all the image segmentation experiments. We compared our algorithm portfolio against the new heuristic proposed in de Sousa et al.

(2013), which was developed specifically for the image segmentation application. For each instance, we ran five replicates of the new procedure and all 37 heuristics to obtain a best-known solution for each problem instance, using the runtime limit specified in Section 4. The algorithm portfolio had mean expected deviation of 1.22% and 0.07% after 10% and 100% of the runtime limit, respectively, while the procedure from de Sousa et al. (2013) had a mean expected deviation of 34.45% and 32.70%, respectively. Computing graph metrics for the algorithm portfolio introduced relatively little overhead, taking an average of just 0.013 seconds per instance (range: 0.001–0.061).

For image segmentation, our algorithm portfolio obtained significantly higher quality solutions than a procedure developed specifically for problem instances from that application area. This highlights the potential benefit of open-source algorithm portfolios to practitioners — these procedures could yield high-quality solutions in the application area of interest, and an open-source implementation made available could save significant implementation effort compared to developing a new heuristic.

## 7. Discussion and Future Work

In this work, we performed a systematic evaluation of heuristics for Max-Cut and QUBO that addresses the irreproducibility, lack of fair comparison, and homogeneous problem instances observed in the empirical testing of heuristics for these problems. We evaluated heuristics for these problems by assembling a heterogeneous problem instance library, constructing an open-source implementation of many previously published heuristics, and evaluating these heuristics using cloud computing resources. We demonstrate that this enables reproducible, fair comparison of heuristics at large scale. Further benefits include providing better insights into when heuristics perform well or poorly and enabling the construction of a state-of-the-art algorithm portfolio that could be useful to practitioners.

Our systematic heuristic evaluation for Max-Cut and QUBO has several limitations. In the context of instance generation, there have been a number of studies that explore instance generation by genetically evolving a population of instances to find hard instances for a set of algorithms (Smith-Miles and van Hemert 2011). It would be interesting to see such instance space exploration techniques integrated into this work. Next, our work relies on the reimplementation of previously published heuristics, which may differ from the original authors' implementations either due to mistakes/underspecification in the

published pseudocode describing that procedure or due to programming errors. However, making the code base open source reduces this problem since implementation errors can be more easily identified and addressed. Secondly, we were only able to implement heuristics from 19 of the 95 previously published heuristic papers for Max-Cut and QUBO, so we may have missed the best-performing procedure for a subset of the problem instances. Further, this work does not include heuristics that rely on external libraries or on complex subroutines such as a semidefinite program or mixed-integer linear program solver. Given that our code base has been made open source and easily accessible to the community, additional important heuristics (that may use external libraries) can be added to the code base. Next, the data structures that we use in this work are well suited for sparse representation of graphs. It might be beneficial to explore specialized data structures that are more suitable for very dense graphs. There have been a number of studies that consider a class of heuristics indexed by parameters and have used an algorithm selection approach to perform parameter tuning. We believe this to be an interesting direction of future work. Finally, we do not consider the performance of algorithm portfolios that run the same heuristic more than once in Section 6. This is because we simply select the most promising set of $k$ heuristics while constructing the portfolio, a design decision that could be changed in future work.

We hope that the source code and evaluation performed in this work will be useful in improving the quality of future Max-Cut and QUBO evaluation. The runtime limit defined in Section 4 enables other researchers to compare their heuristics to this work without rerunning all the 37 heuristics again on the expanded instance library. Such testing can also be performed using the same cloud computing hardware that we used, which is accessible to anybody. We provide a script in our code base to enable parallel testing on the cloud. Testing a new Max-Cut or QUBO heuristic with the interesting instances from our expanded instance library would take approximately 24.0 wall-clock hours (using 20 parallel cloud computing machines) and cost $32.5 on the Amazon cloud.

Given the importance of heuristics in applied areas and the rapidly growing literature on empirical testing to evaluate heuristics, we believe it is important to get the most value possible from computational evaluation. Though in this work we focused on heuristics for Max-Cut and QUBO, there are still a large number of problems where there do not exist projects that provide access to the state-of-the-art heuristics. Many of the weaknesses we

noted in Section 1 also apply to experimental evaluation of heuristics for other problems. Among the first 12 heuristics papers published in the *INFORMS Journal of Computing* since 2013 that compared to previously published heuristics, only five used the same computer for all comparisons and only two allocated the same computational budget to all heuristics. Further, only three of the 12 released the source code or an executable for their heuristic, eight evaluated with fewer than 100 problem instances (often randomly generated), and none of the published works compared to more than eight prior heuristics. We hope that open-source implementation and large-scale evaluation will be applied widely to heuristic evaluation to (i) make empirical testing of heuristics more fair, broad, and reproducible, (ii) provide better insights into where heuristics perform well or poorly, and (iii) make state-of-the-art heuristics accessible to practitioners.

## Acknowledgments

The authors would like to thank Dimitris Bertsimas for funding assistance and discussions, Roman Kim for his contributions to the implementation of two of the heuristics, and MIT librarian Amy Stout for assistance in the systematic literature review.

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 1122374, NSF Grant Nos. CCF-1115849 and OR-1029603, and ONR Grant Nos. N00014-14-1-0072, N00014-12-1-0999 and N00014-15-1-2083. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation and the Office of Naval Research.

### Appendix A: Standard and Expanded Instance Libraries

We consider the *standard instance library* for Max-Cut and QUBO to be the set of instances used for computational experiments by at least four papers out of the 95 papers identified in Section 3; the six sources that make up this library are described in Table 2. All QUBO instances were converted to Max-Cut instances using the reduction from Hammer (1965), and node counts and densities in Table 2 are for the resulting Max-Cut instances.

| Problem | Name | Count | Description | | Reference |
|---|---|---|---|---|---|
| | | | Nodes | Density | |
| Max-Cut | G-set | 71 | 800–20,000 | 0.0%–6.0% | Helmberg and Rendl (2000) |
| | Spin Glass | 30 | 125–2,744 | 0.2%–4.8% | Burer et al. (2002) |
| | Torus | 4 | 512–3,375 | 0.2%–1.2% | $7^{th}$ DIMACS Implementation Challenge |
| QUBO | GKA | 45 | 21–501 | 8.0%–99.0% | Glover et al. (1998) |
| | Beasley | 60 | 51–2,501 | 9.9%–14.7% | Beasley (1998) |
| | P3-7 | 21 | 3,001–7,001 | 49.8%–99.5% | Palubeckis (2006) |

**Table 2 Sources of instances considered to be the standard instance library for Max-Cut and QUBO.**

The expanded instance library includes the standard library but also draws from a wide collection of additional sources, including ORLib (Beasley 1990), SteinLib (Koch et al. 2001), the 2nd, 7th, and 11th DIMACS implementation challenges, TSPLib (Reinelt 1991), and additional instances from the Biq Mac library that are not used extensively in testing Max-Cut and QUBO heuristics (Wiegele 2007). We also generated a number of random graphs using Culberson random graph generators (Culberson et al. 1995), the Python NetworkX library (Hagberg et al. 2008), and the machine-independent random graph generator `rudy` (Rinaldi 1996). Many different types of graphs were constructed, including Erdös-Rényi graphs (Erdös and Rényi 1960), Barabási-Albert graphs (Barabási and Albert 1999), Watts-Strogatz graphs (Watts and Strogatz 1998), and their variants. To add another dimension to our test instances, we considered edge weights sampled from 65 different probability distributions including uniform, triangular, beta, and exponential. We provide the full set of parameters for our database in the instance library at `https://github.com/MQLib`.

## Appendix B: Graph Metrics

We calculated 58 metrics describing each instance in our library. Ten of these metrics are "global" metrics that consider the graph as a whole, and the remainder are "local" metrics that are summary statistics of various node- and edge-specific quantities. Figure 11 shows the complete list of metrics and how well they are covered by our expanded instance library versus the standard library. Each of the metrics in the figure is referred to by a short name, which we provide in the following descriptions.

**Global metrics** The first four metrics are simple to calculate: the logarithm[4] of the number of nodes and edges in the graph (**log_n** and **log_m**, respectively), the proportion of edges with a positive weight (**percent_pos**), and a zero-one indicator for whether the graph is disconnected or not (**disconnected**). The next three metrics are all more complex measures of structure: the approximate chromatic number of the graph as computed by the Welsh-Powell heuristic (Welsh and Powell 1967), normalized by the number of nodes (**chromatic**); the degree assortativity of the graph, a measure of correlation in the degree of linked nodes[5] (**assortativity**); and the approximate size of the maximum independent set of the graph calculated by a fast heuristic,[6] normalized by the number of nodes (**mis**). The final three metrics are all calculated from the weighted graph Laplacian matrix: the logarithm of the first and second largest eigenvalues normalized by the average node degree (**log_norm_ev1** and **log_norm_ev2**, respectively) and the logarithm of the ratio of the two largest eigenvalues (**log_ev_ratio**).

**Local metrics** Six different local quantities were calculated, and for each quantity eight statistical measures were calculated to give a total of 48 local metrics. The first quantity is simply the weight of each edge, normalized by the maximum absolute value across all the edge weights (**weight**). Next, we computed the average degree connectivity (**avg_deg_conn**), where the degree connectivity of degree $k$ is the average degree of the neighbors of nodes with degree $k$. If no nodes have a particular degree value that degree is ignored in the computation. The last four local quantities were specific to nodes and their neighbors:

[4] It is a standard practice to take the logarithm for some features with highly varied values to reduce the effect of outliers.

[5] Degree assortativity is computed as $\frac{\sum_{(i,j)\in E} 2(d(i)-\bar{d})(d(j)-\bar{d})}{\sum_{(i,j)\in E}(d(i)-\bar{d})^2+(d(j)-\bar{d})^2}$, where $\bar{d}=\sum_{(i,j)\in E}\frac{d(i)+d(j)}{2|E|}$.

[6] We use a greedy heuristic for finding an approximate maximum independent set by at each step choosing a vertex with the minimum degree and removing its neighbors.

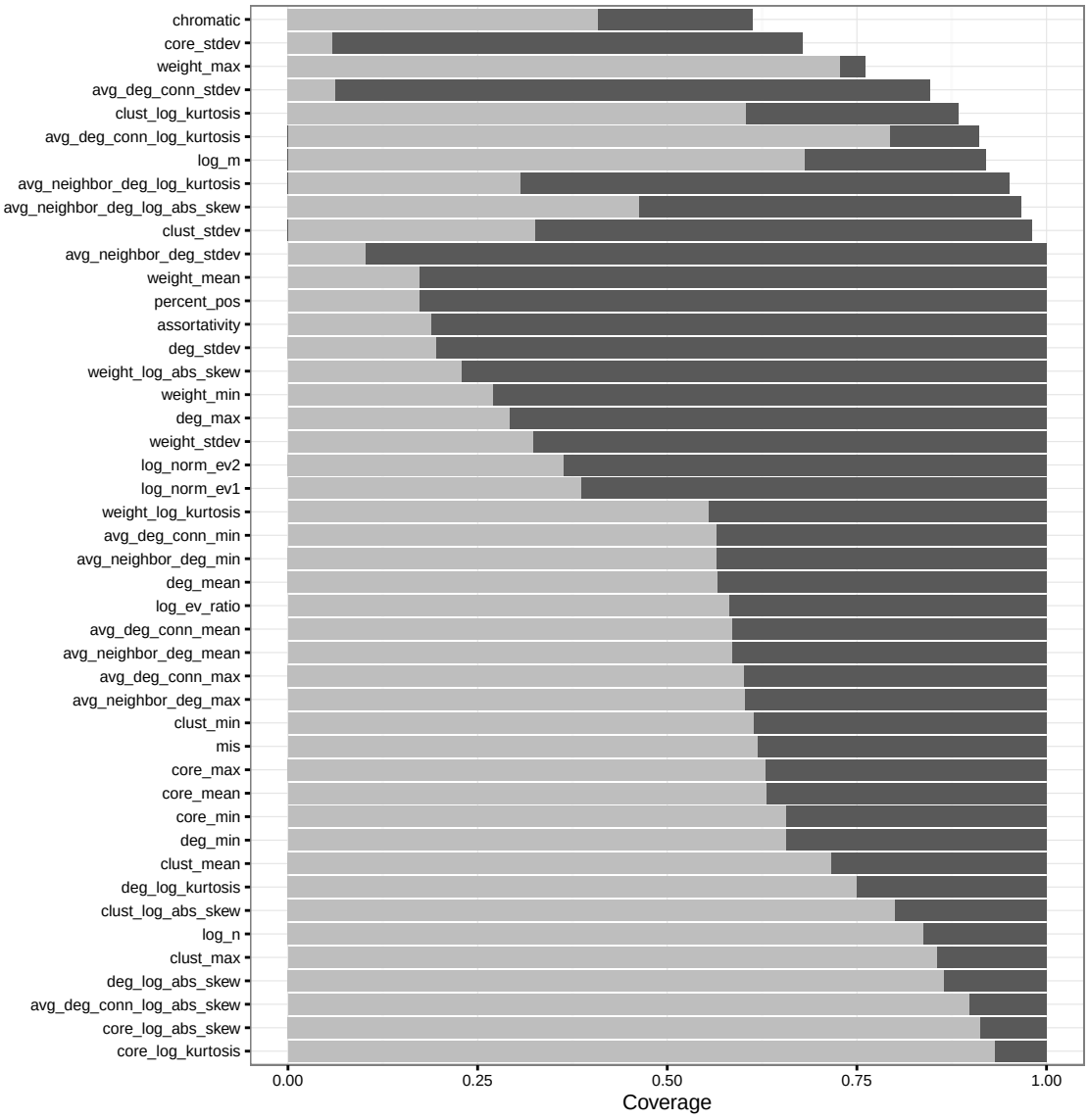


**Figure 11** **Coverage of each metric by instances with 500 or more nodes in the standard instance library (gray) versus the expanded instance library (gray and black together). Binary metrics are not included.**

- the degree of each node (**deg**), normalized by the maximum possible degree.
- the local clustering coefficient of a random subset of size $3\lceil \ln(n) \rceil$ out of the $n$ nodes (**clust**), computed as the number of triangles a node forms with its neighbors divided by the total number of possible triangles. Nodes with fewer than two neighbors are not included.
- the average degree of the neighbors of each node, (**avg_neighbor_deg**), normalized by the maximum possible degree. Nodes with no neighbors are not included.
- the core number for each node in the graph (**core**), normalized by the the number of nodes minus one.[7]

[7] A $k$-core of the graph is a maximal subgraph in which all nodes in the subgraph have degree $k$ or more. The maximum $k$ such that the graph has a $k$-core is called its core number. The core number for a node is the maximum $k$ such that the node lies in a $k$-core.

The first four statistical measures are simply the minimum (**_min**), maximum (**_max**), mean (**_mean**), and standard deviation (**_stdev**) of each of the vectors of local quantities. We also considered the skewness of the quantities using the logarithm of one plus the absolute value of skewness (**_log_abs_skew**) and a zero-one indicator if the skewness was non-negative (**_skew_positive**). Further, we captured the logarithm of four plus the excess kurtosis of each vector of local quantities (**_log_kurtosis**). Finally we included a zero-one indicator to capture whether all values for a quantity were the same (**_const**).

## Appendix C: Identification and Implementation of Heuristics

As described in Section 1, one of the steps of our process is to identify and implement heuristics described in the literature. Here we elaborate on the description in Section 3 of our methodology for identifying heuristics in a systematic way, our approach to implementing them, the modifications made to include some heuristics, and the memory scaling of the heuristics.

**Selection of heuristics** We searched all English-language publications indexed in the Compendex and Inspec databases published in 2013 or earlier with the query

$$\text{maxcut} \vee \text{max-cut} \vee \text{“max cut”} \vee (\text{quadratic} \wedge \text{binary} \wedge \text{unconstrained})$$

which identified an initial set of 810 results. For a paper to be considered relevant it had to satisfy the following criteria:

1. The paper must describe at least one new heuristic for Max-Cut or QUBO.
2. The paper must have computational experiments for the new heuristic(s) it describes.
3. Heuristics must contain an element of randomness. For example, a deterministic greedy heuristic to construct a solution would not be accepted. We added this criterion as deterministic heuristics do not support arbitrary running times, making head-to-head comparisons with a pre-determined runtime limit challenging.
4. Heuristics must not rely on external libraries or rely on complex subroutines such as a semidefinite program or mixed-integer linear program solver.

Table 3 lists the final set of 19 papers selected and the 37 heuristics they describe, including their asymptotic memory consumption in terms of the number of nodes/variables $n$, the number of edges/non-zero matrix elements $m$ and various heuristic-specific parameters. The asymptotic memory consumption displayed in Table 3 assumes only the most recent best-known solution is stored for each heuristic. Our software library also gives an option to store the history of all best-known solutions encountered throughout the run of a heuristic. In general, heuristics may encounter any number of best-known solutions as they are run, and each requires $O(n)$ memory if they are stored.

**Implementation of heuristics** While implementing all the heuristics in `C++` is important to reduce the impact of implementation details on observed performance, perhaps even more important are the shared components used across heuristics.

*Instance* We define classes `MaxCutInstance` (stores the instance as a graph adjacency list) and `QUBOInstance` (stores the instance as a sparse matrix and dense diagonal vector), which both share file loading code. Examples of functionality provided by the instance classes include efficient iterators, shuffled edge lists, and

| Paper | Type | Short name | Description | Space Req. |
|---|---|---|---|---|
| Alkhamis et al. (1998) | Q | ALK98 | Simulated annealing | $O(n+m)$ |
| Beasley (1998) | Q | BEA98SA | Simulated annealing | $O(n+m)$ |
| | | BEA98TS | Tabu search | $O(n+m)$ |
| Burer et al. (2002) | M | BUR02 | Non-linear optimization with local search | $O(n+m)$ |
| Duarte et al. (2005) | M | DUA05 | Genetic algorithm with VNS as local search | $O(pn+m)$ |
| Festa et al. (2002) | M | FES02G | GRASP with local search | $O(n+m)$ |
| | | FES02GP | GRASP with path-relinking | $O(\lambda n+m)$ |
| | | FES02V | VNS | $O(n+m)$ |
| | | FES02VP | VNS with path-relinking | $O(\lambda n+m)$ |
| | | FES02GV | GRASP with VNS local search | $O(n+m)$ |
| | | FES02GVP | GRASP & VNS with path-relinking | $O(\lambda n+m)$ |
| Glover et al. (1998) | Q | GLO98 | Tabu search | $O(tn+m)$ |
| Glover et al. (2010) | Q | GLO10 | Tabu search with long-term memory | $O(\lambda n+m)$ |
| Hasan et al. (2000) | Q | HAS00GA | Genetic algorithm | $O((p+\tau)n+m)$ |
| | | HAS00TS | Tabu search | $O(n+m)$ |
| Katayama et al. (2000) | Q | KAT00 | Genetic algorithm with $k$-opt local search | $O(pn+m)$ |
| Katayama and Narihisa (2001) | Q | KAT01 | Simulated annealing | $O(n+m)$ |
| Laguna et al. (2009) | M | LAG09CE | Cross-entropy method | $O(5.87\rho n^2+m)$ |
| | | LAG09HCE | Cross-entropy method with local search | $O(0.031n^2+m)$ |
| Lodi et al. (1999) | Q | LOD99 | Genetic algorithm | $O(pn+m)$ |
| Lü et al. (2010) | Q | LU10 | Genetic algorithm with tabu search | $O(pn+m)$ |
| Merz and Freisleben (1999) | Q | MER99LS | Genetic algorithm, with crossover and local search | $O(\beta n+m)$ |
| | | MER99MU | Genetic algorithm, with mutation only | $O(\beta n+m)$ |
| | | MER99CR | Genetic algorithm, with crossover only | $O(\beta n+m)$ |
| Merz and Freisleben (2002) | Q | MER02GR | GRASP without local search | $O(n+m)$ |
| | | MER02LS1 | 1-opt local search with random restarts | $O(n+m)$ |
| | | MER02LSK | $k$-opt local search with random restarts | $O(n+m)$ |
| | | MER02GRK | $k$-opt local search with GRASP | $O(n+m)$ |
| Merz and Katayama (2004) | Q | MER04 | Genetic algorithm, with $k$-opt local search | $O(\alpha n+m)$ |
| Palubeckis (2004) | Q | PAL04T1 | Tabu search | $O(n+m)$ |
| | | PAL04T2 | Iterated tabu search | $O(n+m)$ |
| | | PAL04T3 | Tabu search with GRASP | $O(n+m)$ |
| | | PAL04T4 | Tabu search with long-term memory | $O((\lambda+\gamma)n+m)$ |
| | | PAL04T5 | Iterated tabu search | $O(n+m)$ |
| | | PAL04MT | Tabu search | $O(n+m)$ |
| Palubeckis (2006) | Q | PAL06 | Iterated tabu search | $O(n+m)$ |
| Pardalos et al. (2008) | Q | PAR08 | Global equilibrium search | ** |

**Table 3 Implemented heuristics for the Max-Cut ("M") and QUBO ("Q") problems with their asymptotic space requirements. For the space requirements, $n$ is the number of nodes, $m$ is the number of edges (Max-Cut) or number of non-zero matrix entries (QUBO), $p$ is the initial population size, $\lambda$ is the maximum number of elite solutions maintained, $\gamma$ is the maximum size of the set of solutions to be avoided that the algorithm maintains, $t$ is the number of tabu neighborhoods, $\tau$ is the number of tournaments, $\rho$ is the fraction of the population retained in each iteration, $\beta$ is the recombination rate, and $\alpha$ is the crossover rate. For Pardalos et al. (2008), the memory requirement is at least $O(\lambda n+m)$.**

vertex degree counts. One caveat with our implementation is that if an instance is dense then the use of sparse storage formats may introduce some inefficiency. As the internal instance data structure is hidden from the heuristic code, dynamic selection of the correct storage format on a per-instance basis is a possible extension.

*Solution* The `MaxCutSolution` and `QUBOSolution` classes both inherit from an abstract `ExtendedSolution` class that consists of the solution vector, the objective value, and a vector quantifying how much the objective will change from flipping a vertex across the cut (Max-Cut) or flipping a variable's value between 0 and 1 (QUBO). This is essential to the efficient implementation of a local search, as not only can the best swap be found in $O(n)$ time for an instance with $n$ nodes/variables, but the vector of objective value changes can also be updated efficiently. This data structure is described in some of the implemented papers, e.g. Pardalos et al. (2008).

*Operations* The base solution class also implements commonly used subroutines. For example, we implement the local search subroutines `AllBest1Swap` (repeatedly flip the node/variable that best improves the solution until no improving moves exist), `AllFirst1Swap` (repeatedly flip the lexicographically first occurring node/variable that improves the solution until no improving moves exist), and `AllShuffle1Swap` (shuffle the nodes/variables and then repeatedly flip the lexicographically first occurring node/variable that improves the solution until no improving moves exist). An additional example is `UpdateCutValues`, which efficiently flips a vertex across the cut (Max-Cut) or flips a variable's value between 0 and 1 (QUBO).

In our work all heuristics need to be able to run indefinitely with a chance of continuing to make progress (if not already optimal), as mentioned in Section 3. Many heuristics satisfy this property without modification: BUR02, FES02*, KAT00, LOD99, MER99*, MER02*, MER04, PAR08, and half of the PAL04* variants. We wrapped the remaining heuristics in random restarts.

**Memory Scaling of Heuristics** To complement the asymptotic memory consumption of each heuristic displayed in Table 3, we also performed scaling analysis of the memory consumption of each heuristic. Because in this work all heuristics are provided the same runtime limit, no scaling analysis was performed of heuristic runtimes.

First, we tested each heuristic on randomly generated dense Max-Cut instances consisting of complete graphs with each edge weight randomly selected with equal probability from the set {-1, 1}. To observe memory scaling behavior on these dense problem instances, we randomly generated instances of size 100, 300, 1,000, and 3,000 nodes and tested each heuristic on each instance. For each instance, all heuristics were given the same runtime limit, as described in Section 4.1. The peak memory consumption of each heuristic for different dense instance sizes is plotted in Figure 12 (left). The observed scaling confirms an $O(n^2)$ memory consumption for all heuristics on dense problem instances, as expected from the asymptotic analysis. Heuristic PAL04T5 stands out as having more memory usage than the other heuristics, which is because that heuristic constructs perturbed versions of the problem instance, a memory-intensive procedure.

Further, we tested each heuristic on randomly generated sparse Max-Cut instances. To construct an instance with $n$ nodes, we randomly generated an Erdős-Rényi graph with $n$ nodes and edge probability $5/(n-1)$, meaning each node in the graph has expected degree 5. Edge weights for selected edges were again randomly selected with equal probability from the set {-1, 1}. To observe memory scaling behavior on these sparse problem instances, we randomly generated instances of size 1,000, 3,000, 10,000, and 30,000 nodes and tested each heuristic on each instance. Again, we gave each heuristic the same runtime limit, as described in Section 4.1. The peak memory consumption of each heuristic for different sparse instance

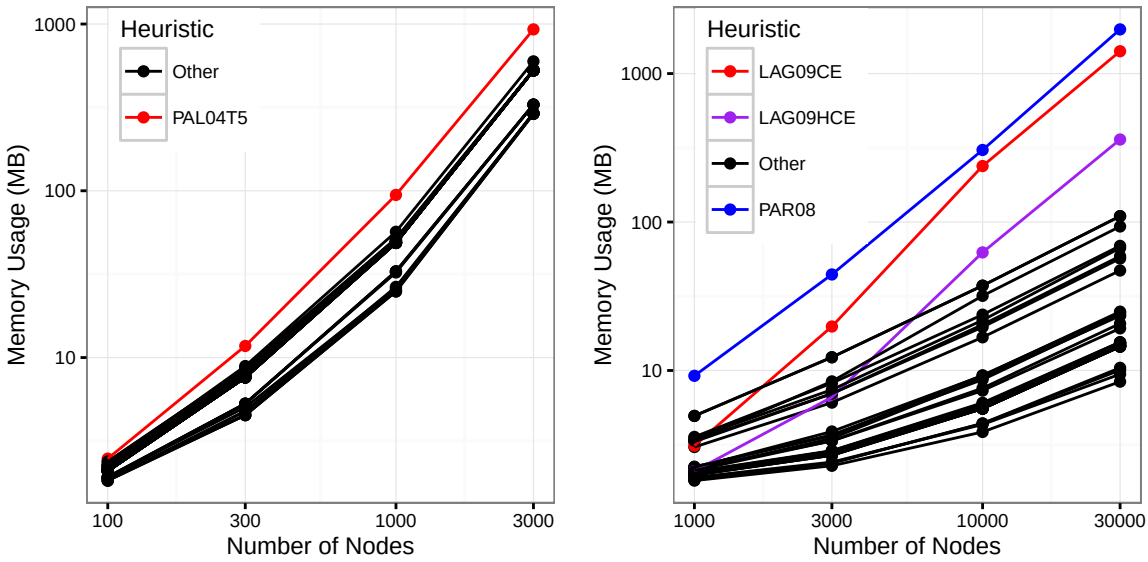


**Figure 12** **A comparison of the memory scaling behavior of the heuristics (note the log-transformed axes). (Left) Complete graphs with randomly selected edge weights. (Right) Sparse Erdős-Rényi graphs with expected degree 5 and randomly selected edge weights.**

sizes is plotted in Figure 12 (right). The observed scaling confirms an $O(n)$ memory consumption for all heuristics except LAG09CE, LAG09HCE, and PAR08 on sparse problem instances, as expected from the asymptotic analysis. Further, LAG09CE and LAG09HCE exhibit expected $O(n^2)$ memory consumption on sparse problem instances. Finally, heuristic PAR08 consumes more memory than any other heuristic. This is because PAR08 stores all locally optimal solutions encountered during its tabu search procedure, which can be quite memory intensive.

### Appendix D: Random Forest Variable Importance

Table 4 lists the variable importance values for all the 58 metrics used in this work, averaged over the random forest models for the 37 heuristics.

## References

Ali, Shawkat, Kate A Smith. 2006. On learning algorithm selection for classification. *Applied Soft Computing* **6** 119–138.

Alkhamis, Talal M, Merza Hasan, Mohamed A Ahmed. 1998. Simulated annealing for the unconstrained quadratic pseudo-Boolean function. *European Journal of Operational Research* **108** 641–652.

Barabási, Albert-László, Réka Albert. 1999. Emergence of scaling in random networks. *Science* **286** 509–512.

Barahona, Francisco. 1996. Network design using cut inequalities. *SIAM Journal of Optimization* **6** 823–837.

Barahona, Francisco, Martin Grötschel, Michael Jünger, Gerhard Reinelt. 1988. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research* **36** 493–513.

| variable | MDGr | MDG | pct | variable | MDGr | MDG | pct |
|---|---|---|---|---|---|---|---|
| log_n | 7.90 | 23.00 | 74.00 | log_m | 8.10 | 13.00 | 74.00 |
| log_norm_ev2 | 8.50 | 12.40 | 68.00 | log_ev_ratio | 10.00 | 10.10 | 71.00 |
| log_norm_ev1 | 10.90 | 9.00 | 65.00 | weight_log_kurtosis | 11.40 | 11.30 | 53.00 |
| weight_min | 13.00 | 10.80 | 53.00 | weight_mean | 13.40 | 9.90 | 47.00 |
| weight_stdev | 13.40 | 8.40 | 47.00 | deg_stdev | 14.40 | 7.60 | 47.00 |
| weight_log_abs_skew | 16.60 | 6.80 | 32.00 | mis | 18.60 | 8.90 | 29.00 |
| assortativity | 18.70 | 7.60 | 26.00 | avg_deg_conn_min | 19.90 | 6.30 | 21.00 |
| avg_deg_conn_stdev | 20.20 | 5.30 | 26.00 | deg_log_abs_skew | 20.40 | 5.60 | 21.00 |
| deg_max | 21.50 | 6.80 | 15.00 | avg_deg_conn_log_kurtosis | 21.80 | 6.20 | 26.00 |
| avg_neighbor_deg_min | 22.30 | 5.80 | 15.00 | chromatic | 23.10 | 5.50 | 6.00 |
| deg_log_kurtosis | 23.60 | 5.10 | 12.00 | avg_neighbor_deg_stdev | 24.70 | 5.30 | 24.00 |
| avg_deg_conn_max | 25.40 | 3.50 | 12.00 | avg_deg_conn_mean | 25.40 | 3.80 | 6.00 |
| avg_neighbor_deg_mean | 25.40 | 3.90 | 12.00 | core_mean | 25.50 | 4.80 | 18.00 |
| avg_neighbor_deg_max | 26.10 | 3.80 | 15.00 | deg_mean | 26.40 | 4.20 | 12.00 |
| core_stdev | 26.60 | 4.50 | 6.00 | deg_min | 26.80 | 3.60 | 9.00 |
| avg_neighbor_deg_log_abs_skew | 27.30 | 4.20 | 0.00 | avg_neighbor_deg_log_kurtosis | 27.40 | 4.10 | 6.00 |
| core_log_kurtosis | 27.90 | 4.40 | 9.00 | clust_stdev | 28.00 | 4.60 | 6.00 |
| core_min | 28.40 | 3.50 | 6.00 | clust_mean | 28.60 | 5.10 | 3.00 |
| avg_deg_conn_log_abs_skew | 29.10 | 3.90 | 9.00 | core_log_abs_skew | 30.50 | 3.90 | 3.00 |
| clust_max | 31.10 | 6.00 | 6.00 | core_max | 33.10 | 2.90 | 0.00 |
| clust_log_abs_skew | 33.40 | 3.60 | 0.00 | percent_pos | 34.80 | 3.60 | 3.00 |
| clust_log_kurtosis | 35.30 | 3.50 | 0.00 | clust_min | 35.40 | 3.60 | 6.00 |
| avg_neighbor_deg_skew_positive | 44.40 | 1.90 | 3.00 | deg_skew_positive | 44.90 | 1.50 | 3.00 |
| weight_skew_positive | 47.50 | 0.50 | 0.00 | avg_deg_conn_skew_positive | 48.00 | 0.60 | 0.00 |
| clust_skew_positive | 49.70 | 0.40 | 0.00 | weight_const | 50.80 | 0.30 | 0.00 |
| weight_max | 51.10 | 0.40 | 0.00 | core_skew_positive | 51.40 | 0.30 | 0.00 |
| clust_const | 51.60 | 0.40 | 0.00 | core_const | 51.70 | 0.20 | 0.00 |
| deg_const | 54.50 | 0.10 | 0.00 | avg_deg_conn_const | 54.80 | 0.10 | 0.00 |
| avg_neighbor_deg_const | 55.10 | 0.10 | 0.00 | disconnected | 55.40 | 0.10 | 0.00 |

**Table 4 The variable importance of each feature averaged over all the heuristic-specific random forest models, showing overall importance in predicting heuristic performance for any given instance, as described in Section 6.1. Here, MDGr is the mean decrease in Gini rank, MDG is the mean decrease in Gini, and pct is the percentage of random forest models for which this feature was in the top 10 most important variables. Variables are sorted in increasing order by the mean decrease in Gini rank, with ties broken by the mean decrease in Gini.**

Barr, Richard S., Bruce L. Golden, James P. Kelly, Mauricio G.C. Resende, William R. Stewart, Jr. 1995. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics* **1** 9–32.

Bartz-Beielstein, Thomas. 2006. *Experimental Research in Evolutionary Computation: The New Experimentalism*. Springer-Verlag Berlin Heidelberg.

Beasley, John E. 1990. OR-Library: distributing test problems by electronic mail. *Journal of the operational research society* **41** 1069–1072.

Beasley, John E. 1998. Heuristic algorithms for the unconstrained binary quadratic programming problem. Tech. rep., Imperial College, London, UK.

Boros, Endre, Peter L. Hammer, Gabriel Tavares. 2016. Psuedo-boolean optimization web-project. `http://rutcor.rutgers.edu/~pbo/`.

Brazdil, Pavel B, Carlos Soares, Joaquim Pinto Da Costa. 2003. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. *Machine Learning* **50** 251–277.

Breiman, Leo. 2001. Random forests. *Machine learning* **45** 5–32.

Breiman, Leo, Jerome Friedman, Richard A. Olshen, Charles J. Stone. 1994. *Classification and Regression Trees*. Chapman & Hall, New York.

Burer, Samuel, Renato DC Monteiro, Yin Zhang. 2002. Rank-two relaxation heuristics for Max-Cut and other binary quadratic programs. *SIAM Journal on Optimization* **12** 503–521.

Burke, Edmund, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, John R. Woodward. 2010. A classification of hyper-heuristic approaches. Michel Gendreau, Jean-Yves Potvin, eds., *Handbook of Metaheuristics*. Springer, 449–468.

Cheeseman, Peter, Bob Kanefsky, William M Taylor. 1991. Where the really hard problems are. *IJCAI*, vol. 91. 331–337.

Corne, David W, Alan P Reynolds. 2010. Optimisation and generalisation: footprints in instance space. *International Conference on Parallel Problem Solving from Nature*. Springer, 22–31.

Culberson, Joseph, Adam Beacham, Denis Papp. 1995. Hiding our colors. *CP95 Workshop on Studying and Solving Really Hard Problems*. Citeseer, 31–42.

de Sousa, Samuel, Yll Haxhimusa, Walter G. Kropatsch. 2013. Estimation of distribution algorithm for the Max-Cut problem. Walter G. Kropatsch, Nicole M. Artner, Yll Haxhimusa, Xiaoyi Jiang, eds., *Graph-Based Representations in Pattern Recognition*, *Lecture Notes in Computer Science*, vol. 7877. Springer Berlin Heidelberg, 244–253.

Duarte, Abraham, Ángel Sánchez, Felipe Fernández, Raúl Cabido. 2005. A low-level hybridization between memetic algorithm and VNS for the Max-Cut problem. *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. ACM, 999–1006.

Eiben, A.E., Márk Jelasity. 2002. A critical note on experimental research in EC. *Proceedings of the 2002 Congress on Evolutionary Computation*. IEEE Press, Piscataway, NJ, 582–587.

Erdös, Paul, A Rényi. 1960. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci* **5** 17–61.

Festa, Paola, Panos M Pardalos, Mauricio GC Resende, Celso C Ribeiro. 2002. Randomized heuristics for the MAX-CUT problem. *Optimization methods and software* **17** 1033–1058.

Foster, Jacob G, David V Foster, Peter Grassberger, Maya Paczuski. 2010. Edge direction and the structure of networks. *Proceedings of the National Academy of Sciences* **107** 10815–10820.

Gent, Ian P, Toby Walsh. 1996. The TSP phase transition. *Artificial Intelligence* **88** 349–358.

Glover, Fred, Gary A Kochenberger, Bahram Alidaee. 1998. Adaptive memory tabu search for binary quadratic programs. *Management Science* **44** 336–345.

Glover, Fred, Zhipeng Lü, Jin-Kao Hao. 2010. Diversification-driven tabu search for unconstrained binary quadratic problems. *4OR* **8** 239–253.

Hagberg, Aric A., Daniel A. Schult, Pieter J. Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. Gaël Varoquaux, Travis Vaught, Jarrod Millman, eds., *Proceedings of the 7th Python in Science Conference*. Pasadena, CA, USA, 11–15.

Hammer, P.L. 1965. Some network flow problems solved with pseudo-boolean programming. *Operations Research* **13** 388–399.

Hartmann, Alexander K, Martin Weigt. 2003. Statistical mechanics of the vertex-cover problem. *Journal of Physics A: Mathematical and General* **36** 11069.

Hartmann, Alexander K, Martin Weigt. 2006. *Phase transitions in combinatorial optimization problems: basics, algorithms and statistical mechanics*. John Wiley & Sons.

Hasan, Merza, T AlKhamis, J Ali. 2000. A comparison between simulated annealing, genetic algorithm and tabu search methods for the unconstrained quadratic Pseudo-Boolean function. *Computers & Industrial Engineering* **38** 323–340.

Helmberg, Christoph, Franz Rendl. 2000. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization* **10** 673–696.

Hill, Raymond R, Charles H Reilly. 2000. The effects of coefficient correlation structure in two-dimensional knapsack problems on solution procedure performance. *Management Science* **46** 302–317.

Hooker, J.N. 1995. Testing heuristics: We have it all wrong. *Journal of Heuristics* **1** 33–42.

Horvitz, Eric, Yongshao Ruan, Carla Gomes, Henry Kautz, Bart Selman, Max Chickering. 2001. A Bayesian approach to tackling hard computational problems. *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 235–244.

Howe, Adele E, Eric Dahlman, Christopher Hansen, Michael Scheetz, Anneliese Von Mayrhauser. 2000. Exploiting competitive planner performance. *Recent Advances in AI Planning*. Springer, 62–72.

Insani, Nur, Kate Smith-Miles, Davaatseren Baatar. 2013. Selecting suitable solution strategies for classes of graph coloring instances using data mining. *Information Technology and Electrical Engineering (ICITEE), 2013 International Conference on*. IEEE, 208–215.

Johnson, David S. 2002. A theoretician's guide to the experimental evaluation of algorithms. Michael H. Goldwasser, David S. Johnson, Catherine C. McGeoch, eds., *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*. American Mathematical Society, 215–250.

Karp, Richard M. 1972. Reducibility among combinatorial problems. Raymond E. Miller, James W. Thatcher, Jean D. Bohlinger, eds., *Complexity of Computer Computations*. The IBM Research Symposia Series, Springer US, 85–103.

Katayama, Kengo, Hiroyuki Narihisa. 2001. Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. *European Journal of Operational Research* **134** 103–119.

Katayama, Kengo, Masafumi Tani, Hiroyuki Narihisa. 2000. Solving large binary quadratic programming problems by effective genetic local search algorithm. *GECCO*. 643–650.

Kitchenham, Barbara, O Pearl Brereton, David Budgen, Mark Turner, John Bailey, Stephen Linkman. 2009. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology* **51** 7–15.

Koch, Thorsten, Alexander Martin, Stefan Voß. 2001. SteinLib: An updated library on Steiner tree problems in graphs. Xiu Zhen Cheng, Ding-Zhu Du, eds., *Steiner Trees in Industry*, *Combinatorial Optimization*, vol. 11. Springer US, 285–325.

Krzkakała, Florent, Andrea Pagnani, Martin Weigt. 2004. Threshold values, stability analysis, and high-q asymptotics for the coloring problem on random graphs. *Physical Review E* **70** 046705.

Laguna, Manuel, Abraham Duarte, Rafael Martí. 2009. Hybridizing the cross-entropy method: An application to the max-cut problem. *Computers & Operations Research* **36** 487–498.

Leyton-Brown, Kevin, Eugene Nudelman, Galen Andrew, Jim McFadden, Yoav Shoham. 2003. A portfolio approach to algorithm selection. *IJCAI*, vol. 1543. 2003.

Leyton-Brown, Kevin, Eugene Nudelman, Yoav Shoham. 2002. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. *International Conference on Principles and Practice of Constraint Programming*. Springer, 556–572.

Liaw, Andy, Matthew Wiener. 2002. Classification and regression by randomForest. *R News* **2** 18–22. URL `http://CRAN.R-project.org/doc/Rnews/`.

Lodi, Andrea, Kim Allemand, Thomas M Liebling. 1999. An evolutionary heuristic for quadratic 0–1 programming. *European Journal of Operational Research* **119** 662–670.

Lü, Zhipeng, Fred Glover, Jin-Kao Hao. 2010. A hybrid metaheuristic approach to solving the UBQP problem. *European Journal of Operational Research* **207** 1254–1262.

Martí, Rafael, Abraham Duarte, Manuel Laguna. 2009. Advanced scatter search for the Max-Cut problem. *INFORMS Journal on Computing* **21** 26–38.

Martin, D., C. Fowlkes, D. Tal, J. Malik. 2001. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. *Proc. 8th Int'l Conf. computer Vision*, vol. 2. 416–423.

McGeoch, Catherine C. 2012. *A guide to experimental algorithmics*. Cambridge University Press, New York, NY.

Merz, Peter, Bernd Freisleben. 1999. Genetic algorithms for binary quadratic programming. *Proceedings of the genetic and evolutionary computation conference*, vol. 1. Citeseer, 417–424.

Merz, Peter, Bernd Freisleben. 2002. Greedy and local search heuristics for unconstrained binary quadratic programming. *Journal of Heuristics* **8** 197–213.

Merz, Peter, Kengo Katayama. 2004. Memetic algorithms for the unconstrained binary quadratic programming problem. *BioSystems* **78** 99–118.

Newman, Mark EJ, Steven H Strogatz, Duncan J Watts. 2001. Random graphs with arbitrary degree distributions and their applications. *Physical review E* **64** 026118.

Newman, Mark EJ, Duncan J Watts, Steven H Strogatz. 2002. Random graph models of social networks. *Proceedings of the National Academy of Sciences* **99** 2566–2572.

Nikolić, Mladen, Filip Marić, Predrag Janičić. 2013. Simple algorithm portfolio for SAT. *Artificial Intelligence Review* **40** 457–465.

O'Mahony, Eoin, Emmanuel Hebrard, Alan Holland, Conor Nugent, Barry O'Sullivan. 2008. Using case-based reasoning in an algorithm portfolio for constraint solving. *Irish Conference on Artificial Intelligence and Cognitive Science*. 210–216.

Palubeckis, Gintaras. 2004. Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. *Annals of Operations Research* **131** 259–282.

Palubeckis, Gintaras. 2006. Iterated tabu search for the unconstrained binary quadratic optimization problem. *Informatica* **17** 279–296.

Pardalos, Panos M, Oleg A Prokopyev, Oleg V Shylo, Vladimir P Shylo. 2008. Global equilibrium search applied to the unconstrained binary quadratic optimization problem. *Optimisation Methods and Software* **23** 129–140.

Pulina, Luca, Armando Tacchella. 2009. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints* **14** 80–116.

Rardin, Ronald L., Reha Uzsoy. 2001. Experimental evaluation of heuristic optimization algorithms: A tutorial. *Journal of Heuristics* **7** 261–304.

Reinelt, Gerhard. 1991. TSPLIB—a traveling salesman problem library. *ORSA Journal on Computing* **3** 376–384.

Rendl, Franz, Giovanni Rinaldi, Angelika Wiegele. 2010. Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. *Mathematical Programming* **121** 307–335.

Rice, John R. 1976. The algorithm selection problem. *Advances in computers* **15** 65–118.

Rinaldi, G. 1996. RUDY: A generator for random graphs. `http://web.stanford.edu/~yyye/yyye/Gset/rudy.c`. Accessed: 2014-09-30.

Silberholz, John, Bruce Golden. 2010. Comparison of metaheuristics. Michel Gendreau, Jean-Yves Potvin, eds., *Handbook of Metaheuristics*. Springer, 625–640.

Smith, Kate A, Frederick Woo, Vic Ciesielski, Remzi Ibrahim. 2001. Modelling the relationship between problem characteristics and data mining algorithm performance using neural networks. Cihan Dagli, Etin Akay, C.L. Philip Chen, Benito R. Fernandez, Joydeep Ghosh, eds., *Intelligent Engineering Systems Through Artificial Neural Networks*. 356–362.

Smith-Miles, K, R James, J Giffin, Yiqing Tu. 2009. Understanding the relationship between scheduling problem structure and heuristic performance using knowledge discovery. *Learning and Intelligent Optimization, LION* **3**.

Smith-Miles, Kate. 2008. Towards insightful algorithm selection for optimization using meta-learning concepts. *WCCI 2008: IEEE World Congress on Computational Intelligence*. IEEE, 4118–4124.

Smith-Miles, Kate, Davaatseren Baatar, Brendan Wreford, Rhyd Lewis. 2014. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research* **45** 12–24.

Smith-Miles, Kate, Leo Lopes. 2011. Generalising algorithm performance in instance space: a timetabling case study. *International Conference on Learning and Intelligent Optimization*. Springer, 524–538.

Smith-Miles, Kate, Leo Lopes. 2012. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research* **39** 875–889.

Smith-Miles, Kate, Jano van Hemert. 2011. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence* **61** 87–104.

Smith-Miles, Kate, Jano van Hemert, Xin Yu Lim. 2010. Understanding TSP difficulty by learning from evolved instances. *International Conference on Learning and Intelligent Optimization*. Springer, 266–280.

Stadler, Peter F, Wolfgang Schnabl. 1992. The landscape of the traveling salesman problem. *Physics Letters A* **161** 337–344.

Stützle, Thomas, Susana Fernandes. 2004. New benchmark instances for the QAP and the experimental analysis of algorithms. *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, 199–209.

Therneau, Terry, Beth Atkinson, Brian Ripley. 2015. *rpart: Recursive Partitioning and Regression Trees*. URL `https://CRAN.R-project.org/package=rpart`. R package version 4.1-10.

Vilalta, Ricardo, Youssef Drissi. 2002. A perspective view and survey of meta-learning. *Artificial Intelligence Review* **18** 77–95.

Vollmann, Thomas E, Elwood S Buffa. 1966. The facilities layout problem in perspective. *Management Science* **12** B–450.

Wang, Zhe, Tong Zhang, Yuhao Zhang. 2013. Distinguish hard instances of an np-hard problem using machine learning. `http://cs229.stanford.edu/proj2013/WangZhangZhang-DistinguishHardInstancesOfAnNPHardProblem.pdf`.

Watts, Duncan J, Steven H Strogatz. 1998. Collective dynamics of 'small-world' networks. *Nature* **393** 440–442.

Weigt, Martin, Alexander K Hartmann. 2000. Number of guards needed by a museum: A phase transition in vertex covering of random graphs. *Physical review letters* **84** 6118.

Welsh, D. J. A., M. B. Powell. 1967. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal* **10** 85–86.

Wiegele, Angelika. 2007. Biq Mac library — a collection of Max-Cut and quadratic 0-1 programming instances of medium size. Tech. rep., Alpen-Adria-Universität Klagenfurt.

Xu, Lin, Frank Hutter, Holger H Hoos, Kevin Leyton-Brown. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* **32** 565–606.