Journal of Heuristics (2021) 27:497–547
https://doi.org/10.1007/s10732-020-09465-7

Augmented intuition: a bridge between theory and practice

Pablo Moscato[1] · Luke Mathieson[2] · Mohammad Nazmul Haque[1]

Received: 14 July 2019 / Revised: 31 August 2020 / Accepted: 23 December 2020 /
Published online: 31 January 2021

Abstract
Motivated by the celebrated paper of Hooker (J Heuristics 1(1): 33–42, 1995) published in the first issue of this journal, and by the relative lack of progress of both approximation algorithms and fixed-parameter algorithms for the classical decision and optimization problems related to covering edges by vertices, we aimed at developing an approach centered in augmenting our intuition about what is indeed needed. We present a case study of a novel design methodology by which algorithm weaknesses will be identified by computer-based and fixed-parameter tractable algorithmic challenges on their performance. Comprehensive benchmarkings on all instances of small size then become an integral part of the design process. Subsequent analyses of cases where human intuition "fails", supported by computational testing, will then lead to the development of new methods by avoiding the traps of relying only on human perspicacity and ultimately will improve the quality of the results. Consequently, the computer-aided design process is seen as a tool to augment human intuition. It aims at accelerating and foster theory development in areas such as graph theory and combinatorial optimization since some safe reduction rules for pre-processing can be mathematically proved via theorems. This approach can also lead to the generation of new interesting heuristics. We test our ideas with a fundamental problem in graph theory that has attracted the attention of many researchers over decades, but for which seems it seems to be that a certain stagnation has occurred. The lessons learned are certainly beneficial, suggesting that we can bridge the increasing gap between theory and practice by a more concerted approach that would fuel human imagination from a data-driven discovery perspective.

Keywords Vertex cover · Augmented intelligence · Human–computer data-driven discovery · Heuristics · Algorithms · Kernelization

P.M. acknowledges a generous donation from the Maitland Cancer Appeal and previous support from the Australian Research Council Future Fellowship FT120100060 and Australian Research Council Discovery Projects DP120102576, DP140104183 and DP200102364.

Extended author information available on the last page of the article

1 Introduction

"It should be one's sole endeavour to see everything afresh and create it anew."
- Gustav Mahler

Nobel Prize winner in Physics, Richard Feynman, was considered one of the quintessential problem solvers of the twentieth century. In a Chicago Tribune interview[1] his colleague at Caltech, Murray Gell-Mann, another Nobel Prize awardee, who also was a world class problem solver in Physics and a champion for the study of complexity in science (Gell-Mann 1995), half-jokingly specified *'Feynman's Algorithm'* as a simple three-step procedure:

1. Write down the problem.
2. Think really hard.
3. Write down the solution.

Puckish as this may seem, this "algorithm" rightfully models at a very high level the conduct of much of the fundamental practice of Algorithmics and Theory of Computer Science. When faced with a computational problem, algorithmic designers and complexity theorists typically approach a white board (or a blank page), *think really hard* for a while, sketch particular "complex instances of small size" that may come to their mind, *think really hard* (again) to try to identify the worst-case scenario, and they finally *write down the solution* in the form of an algorithm (or a complexity proof that shows membership or some other type of tight result).

While this activity is accompanied by a toolkit of techniques and experience obtained during the past decades, it is most certainly ad hoc. There is no consensus of what "writing down the problem" is. Ideally, computational benchmarking on all instances of small size can help to uncover the reasons why a heuristic may fail to obtain the optimal solution. Systematic benchmarking on small instances is rarely seen in practice, with most benchmarking being done a posteriori, on larger instances. Hooker already pointed at these issues a quarter of a century ago both in Hooker (1995) and in when he appealed for the establishment of a new "empirical science of algorithms" (Hooker 1994).

Undoubtedly, it is also clear that the algorithmics approach of using human intuition alone to generate worst-case instances has had its successes. For certain problems, this standard approach is sufficient and relying on human intuition is all what is needed; a near trivial example is the sorting of a list of numbers where most of the "bad" algorithms are more complex than the "good" ones (cf. *Bubblesort* and *Mergesort*). However, a proposal for using evolutionary algorithms to augment human intuition via "evolutionary attacks" has already been proposed for those algorithms (Cotta and Moscato 2003).

In spite of the success of just using human intuition alone, the core difficulties with the approach run deeply and propagate into the future, for instance, by teaching our young peers. Indeed, we routinely train the next generation of computer scientists and applied discrete mathematicians by handpicking "the best examples" of this

[1] https://www.chicagotribune.com/news/ct-xpm-1992-11-17-9204150260-story.html.

paradigm based on the successes of this approach, and we praise the beauty and simplicity of some proofs. When considering harder problems, however, this leads to the development of algorithms with obscure weaknesses and limitations and/or lack of insights into their empirical success or applicable range. For many problems and many researchers "algorithmic success" then becomes a one-off hit during the lifetime of their careers. We can read in a famous textbook: *"The Algorithm Design Manual"* by S. Skiena: *"Algorithms are the ideas behind computer programs"* (Skiena 2008), a partial definition that algorithms share with heuristics. Very little is said about where to find *intuition* for these ideas, or how we can systematize a process for algorithm design. The early-on take home for students from the book is *"Searching for counterexamples is the best way to disprove the correctness of a heuristic"* ... but he also recognizes that this search *"relies more on inspiration than exhaustion"* (Skiena 2008, Sec. 1.3.3).

What happens when we run out of inspiration? Would then be possible that we could globally and collectively be 'running out of ideas' for a given problem? Could we augment out intuition by employing exhaustive testing on instances of smaller sizes? Can we make a methodological bridge between algorithm and heuristic design by bringing benchmarking a priori instead of a posteriori? We aim at providing a case study in one of combinatorial optimization's most studied problems as an illustrative example.

1.1 The vertex cover problem: much more than a case in point

The history of the well-known combinatorial problem k- Vertex Cover demonstrates the problem in an incisively way. Given an undirected graph $G(V, E)$, a vertex cover V' is a subset of V such that for all edges in E at least one of the endpoints is in V'. The problem of deciding if a graph G can have a vertex cover with at most $k > 0$ vertices is one of the most well-recognized "classical" examples of an NP-complete problem. In the optimization version (Min Vertex Cover), we aim at finding the cover of the minimum cardinality.

The proof of the existence of a "straightforward algorithm" (quoting Garey and Johnson 1979) with a guaranteed performance ratio of 2 is both simple and elegant and, consequently, it has become standard practice to teach this proof in many undergraduate courses around the world. However, the weakness and limitations of this paradigm for algorithm design, which is based on a performance ratio and simplicity of proofs, and which certainly involves "think [ing] really hard", are neither discussed nor questioned. In fact, it comes as a surprise to our students that very little other progress in the development of approximation algorithms for Min Vertex Cover has occurred since Gavril proved this result (the existence of a 2-approximation ratio) in 1974 [known as a private communication to D.S. Johnson according to a well-known textbook (Garey and Johnson 1979)]. The best known approximation ratio for this problem is $2 - \Theta(\frac{1}{\sqrt{log|V|}})$, a result obtained approximately 30 years after Gavil's proof Karakostas (2005). Progress exists on trying to demonstrate the hardness of approximability, and only three years before, Dinur and Safra (2002) proved that it is NP-hard to approximate the cardinality of the minimum vertex cover within any factor smaller than 1.36.

1.2 Fixed-parameterized tractability: a new hope and a new stagnation

A breath of fresh air also happened for this problem; a string of exact algorithms for the Min Vertex Cover has appeared over the past 30 years, each improving on the previous results. These are called *fixed-parameter tractable* algorithms. The problem then quickly became the "superstar" of this new wave of algorithmic design hope. Today it is perhaps the most well-known problem of a computational complexity class known as FPT (i.e., the class of problems for which a fixed-parameter tractable algorithm exists).

This type of approach brought an interesting design strategy; it contained systematic theoretical analyses of instances of small size for the purpose of *kernelization*. They bring, in some sense, an "implicit benchmarking" process via mathematically proven results on *safe reduction rules*. These ideas are explained in detail in Sect. 2. We give an example of a safe reduction rule in Sect. 2.1 and all the reduction rules used in this manuscript are given in Appendix A. They have proven useful to solve to optimality any instance of vertex cover with up to 7 vertices. However, after a very successful 15 years, also in this case we observe a stagnation in the development of fixed-parameter tractable algorithms since no significant further progress occurred since 2005, i.e. fifteen years ago.

1.3 Is lack of progress "natural"?

How to track progress? Unfortunately, there is no generic repository containing all theoretical results in computer science as a semantic database, but there is a very useful online compendium on approximation algorithms maintained by Viggo Kann since 1992.[2] From it we can see that many well-known problems in that compendium also have had very sporadic successes in terms of improving the approximability status over decades. It is also remarkable that vertex cover, the problem that in some sense initiated and help championed the theory of fixed-parameter tractability, also has the same stagnation for graphs of size equal or greater than 8 vertices, a situation that has changed little after nearly 15 years of research.

Should we calmly accept that "this is natural"? Would it be possible that some combinatorial problems are "just like that"? Or, we dare to ask, is it the case that the progress in theoretical computer science, and in this case graph theory, is indeed limited by the human cognitive capacity for identifying structure in complex objects?

We point that Thorup (1998) structured programs (i.e. goto-free programs), "including, for example, short circuit evaluations and multiple exits from loops, have tree width at most 6", so perhaps that is linked to human cognitive capacity for control flow of algorithms. However, today the route to justify the lack of progress over decades is to blame the "inherent hardness" of these problems, their "inapproximability" in some

[2] http://www.csc.kth.se/~viggo/problemlist/.

cases, or, in the case of fixed-parameter algorithms, the increasingly more difficult mathematical proofs required to bring new advances. Of course, we also have another tempting escape route; we can also blame the problem. Usually, a conjecture is open, in this case it was that there does not exist an algorithm with a fixed approximation ratio better than 2 (Feige 2003; Hochbaum 1983).

Both routes are directions for an intellectual escapism and neither is addressing the basic difficulty. Few researchers dare to challenge the common core of these highly entrenched methodological practices in mathematics and computer science, in particular, its lack of comprehensive experimentation with instances of small size. After all, nothing seems wrong with "Fenyman's Algorithm". How can we blame ourselves for "Thinking really hard"? Well, in fact, perhaps we can blame the part that says that we should start with a blank page or a perfectly clean whiteboard. While we acknowledge that we do not yet have a final solution, we offer an alternative for consideration and this manuscript is, humbly speaking, perhaps the first step in that new direction.

1.4 Structure of this paper

This paper is structured as follows: Section 2 discusses *kernelization*, an important concept to understand the paper (while Appendix A presents the set of reduction rules that have been implemented and tested in this study). In Appendix B we point at some key surveys on current methods and practices of automated heuristics designs. In Sect. 2.2 we present the results of these reduction rules in large scale vertex cover benchmarking datasets. Section 3 lays out an augmented intelligence methodology for moving beyond the traditional approach for developing reduction rules and provides some partial conclusions. Section 4 details the design of three new heuristic algorithms built thanks to the insights obtained thanks to the approach of Sect. 3. Section 5 presents the results of these heuristics on the benchmarking algorithms and statistical performance tests are provided. In addition to those, we presented the results of other metaheursitics used for BHOSLIB and DIMACS datasets in Appendices C.1, C.2 and a comparison of our heuristic with the Isolation Algorithm (IA) (Ugurlu 2012) in Appendix C.4. Section 6 discuss possible new reduction rules for the problem; and Sects. 7 and 8 discuss the results and limitations of the study, and present the conclusions and future directions.

2 Fundamentals of kernelization

We will center the discussion on the k- VERTEX COVER decision problem: *"Given a simple undirected graph $G(V, E)$, is there a $V' \subseteq V$ such that $|V'| \leq k$?"* Then, from the point of view of parameterized complexity, the instance is the graph and a parameter (i.e., (G, k)).

2.1 Preprocessing and reduction rules

In the area of Operations Research some polynomial-time techniques have been used to deal with large instances of problems that have some "real-world origin" and have been used in practice. They are called *"safe data reductions"*. Based on them, some *pre-processing techniques* often make seemingly intractable, large instances, such as those arising in machine learning or bioinformatics, small and tractable. A great example of this is Karsten Weihe's "covering trains by stations" story (see Moscato (2019) for a detailed account of it). An example of a reduction rule for the k- Vertex Cover is the following:

Lemma 1 Shared neighborhoods of two adjacent vertices - *Let* (G, k) *be an instance of* k- Vertex Cover *with* $u, v \in V(G)$ *where* $N(u) \subseteq N(v) - u$ *and* $uv \in E(G)$. *Let* $G'(V', E')$ *be the graph obtained by deleting vertex* v *from* G. *Then* (G, k) *is a* Yes *instance of* k- Vertex Cover *iff* $(G - v, k - 1)$ *is also a* Yes *instance.*

Proof

($\Rightarrow$) Let V' be a vertex cover of size k for G. We have two cases: $v \in V'$ and $v \notin V'$. If $v \in V'$, then $V' - v$ is a vertex cover for $G - v$ of size $k - 1$. If $v \notin V'$, we must have $u \in V'$ to cover the edge uv. However as $v \notin V'$, the edges wv for any $w \in N(v) - u$ must be covered by w. In particular this implies that $N(u) \subset V'$. Therefore we can take the alternate vertex cover $V'' = (V' - u) \cup \{v\}$ and we again have the first case.

($\Leftarrow$) Let V' be a vertex cover of size $k - 1$ for G'. V' covers all edges of G except those edges wv for $w \in N(v)$. Thus $V' \cup \{v\}$ forms a vertex cover of size k for G.

□

We note that this reduction rule is not saying that vertex v has to be in a vertex cover of minimum cardinality (but we are saying that it is in one of size k). If we keep on reducing the graph until a vertex cover is found, we can then revisit this vertex and if all vertices in $N(v)$ are already in the vertex cover then v does not need to be. The reduction rule is still valid as a mechanism to answer the decision version for the value of k, and if v is actually no needed, then a vertex cover of cardinality $k - 1$ may actually exist.

2.1.1 Kernelization and fixed-parameter tractability

Formal proofs rules like the one above were previously known as "preprocessing" in the area of Operations Research. It was noted nearly three decades ago that they can become powerful tools to reduce the dimensionality of an instance. The design of *"fixed-parameter algorithms"* depends on identifying these rules and prove their correctness. In general, in parameterized computational complexity the input is a pair (x, k) where x corresponds to an instance of a decision problem and $k > 0$ is a parameter which represents some characteristic of the instance. It is also assumed that k is independent of the size of the instance (i.e., it is assumed to be a fixed value or “fixed parameter”). The overall aim of the application of reduction rules is then generally to transform, in polynomial time, an instance (x, k) of a decision problem P into an equivalent instance (x', k') such that, $|x'| \leq |x'|$ and $k < k'$ and $|x'|+k' \leq f(k)$ for some fixed computable function $f(k)$.

These transformations are obtained via the application of these ‘reduction rules’ and they provide an effective tool for pre-processing instances of NP-hard problems.

This type of technique is known as *kernelization*. For some problems it is possible to find a function $f(k)$, depending only on k, bounding the size of a maximal reduced instance (x', k') (Abu-Khzam et al. 2004).

If such a function exists, then the problem is said to be ‘kernelizable’, and this proves that it is in the computational complexity class called ‘FPT’ (the class of fixed-parameter tractable problems).

2.1.2 Strict kernelization

When a parameterized problem is in class FPT, it is often said that the combinatorial running-time “explosion” (currently conjectured to be something inherently unavoidable to most exact algorithms for NP-hard problems) has now been “confined to the parameters”.

More recently, the concept of *“diminishable problems”* and *“strict polynomial kernelization”* has been introduced Fernau et al. (2018). In this case it is required that $|x'| \leq f(k)$, for some function $f(k)$ in $k^{O(1)}$ and $k' \leq k + c$ for some constant c. This basically means that a “strict kernelization is a kernelization that does not increase the output parameter k' by more than an additive constant” (Fernau et al. 2018).

2.2 Are reduction rules identified for kernelization purposes useful in practice?

This is the a fundamental question for practitioners who develop heuristics. Following Hooker’s classic *“Testing Heuristics: We have it all wrong”* (Hooker 1995), we stop to analyze the result of this computational experiment. We aim at getting some insights of whether the best set of techniques from fixed-parameter algorithmics fails at producing both feasible solutions and to find the optimum for a large number of instances used for benchmarking.

Are these safe rules coming from fixed-parameter tractability useful to reduce the size of the instances? To address the question, we decided to benchmark the algorithm that was is recognized as the best of its class. Since it is based on 10 reductions rules we call it ‘the *The 10-R* approach’ and we provide an explicit description in Appendix A.

To test the effectiveness of *The 10-R*, we employed the widely known benchmarking instances from the Second DIMACS Implementation Challenge. The DIMACS datasets consist of collections of graphs initially proposed to challenge algorithmic designers in the areas of *maximum clique* and *graph coloring* problems. For each type of problem, we have two sets of graphs: *benchmark* and *challenge*. We have applied our implemented reduction rules to reduce the number of vertices of these graphs. We used the following set of instances:

– DIMACS Benchmark Clique Instances

Table 1 Number of successful applications of reduction rules on the DIMACS color graph's benchmark and volume instances over all instances

Reduction rule	Successful application count	
	Benchmark	Volume
Degree Zero	0	0
Degree One	3	5
Degree *k*	0	0
Degree two adjacent	4	8
Degree two non-adjacent	4	4
Complete neighborhood	13	40
Struction	52	78
Crown	0	0
Linear programming	0	5
General fold	7	13

– DIMACS Benchmark Graph Coloring Instances
– DIMACS Volume Graph Coloring Instances
– DIMACS Volume Clique Instances.

These instances go from approximately 200 nodes to 4000 and they have roughly between 4000 and 4 million edges (e.g C4000.5).

To understand if some of the rules in *The 10-R* have more success than others we have counted the number of times each of the reduction rules was able to safely reduce the problem. We present the results in Table 1. We found that the *Struction* rule is the most dominating (the one that more frequently is able to reduce the size of the graph). The next successful reduction rule is *Complete neighborhood*. The *Degree Zero, Degree k* and *Crown* rules have never been able to successfully be applied to reduce any graph used in this experiment, and the *Linear Programming* one is only able to do it five times. None of the reduction rules was able to reduce the size of the instance of the Clique type. Overall, in the light of the amount of theoretical interest and time dedicated to them over several decades, it is a rather disappointing, although not entirely unexpected result. Current benchmarking challenge instances are still very difficult in practice to established kernelization approaches. This motivates the next step of our investigation.

3 Augmented intelligence discovery of new reduction rules

We now look at the following hypothesis: there exists another "yet to be discovered" reduction rule that could be better suited than the ones already known. We consider that it would be very unproductive to search for them by analyzing benchmarking data. The instances that we have used in the previous experiment come from a variety of sources, present different (yet unknown) structural properties, and we would not get inspiration and/or insight due to the large size and density. An alternative approach is subsequently offered.

3.1 First step: test the current tools

We have implemented in a single piece of software all the reduction rules in Abu-Khzam et al. (2004) and Chen et al. (2010) (i.e., *The 10-R* algorithm) and we extended it to be an exact Min Vertex Cover solver. We do this by the application in tandem of the reduction rules *The 10-R* followed by a reduction to an Integer Programming model for Min Vertex Cover which uses CPLEX[3] as a subroutine.

We then used this solver on the complete enumerated sets of non-isomorphic graphs ranging from 2 to 7 vertices respectively. The union of the reduction rules of the two papers cited above (Abu-Khzam et al. 2004; Chen et al. 2010) were able to solve all instances with 7 vertices, as expected. This confirms that our code reduces all of them in agreement with the existing theory.

3.2 Second step: build intuition by working at the interface

The next step is to increase the size of the instance slightly, to appreciate what may be escaping our intuition, yet providing a relatively small instance of the basic problem. Things became really interesting this way. After application of the *The 10-R* algorithm, only 5 irreducible graphs of size 8 (i.e., irreducible by the rules *The 10-R*) exist from the grand total of all non-isomorphic graphs of size 8 (which is 12,346). This means that only five required to be solved by the Integer Programming model using CPLEX that follows. All of these 5 irreducible graphs of size 8 have a vertex cover of size five. They are shown in Fig. 1.

This has been an inspiring and informative experiment. We know that the *The 10-R* are able to reduce any instance of size 7, and "almost" does the same job for all instances of size 8. The existence of a relatively small subset of irreducible instances inspires the quest of identifying one or two other rules that may reduce these instances as well, something which at present is an open problem. This provides the first outcome for further theoretical development of new fixed-parameter algorithms.

3.3 Third step: look ahead

It is then perhaps logical to look one step ahead to see what are we facing. If we do not find a reduction (or reductions) for size 8, these graphs will remain as subgraphs of non-reducible graphs of size 9. How many of them are non-reducible by *The 10-R*? We did this analysis as well. Of the 274,668 non-isomorphic graphs of size 9, the reduction rules only failed to solve 118 instances, showing that the number of irreducible instances has only increased by 23 times.

We think that this is also important for theoretical development. Many inductive proofs would require to specify that a graph is free of some property, which may

[3] IBM ILOG CPLEX Optimization Studio at www.cplex.com.

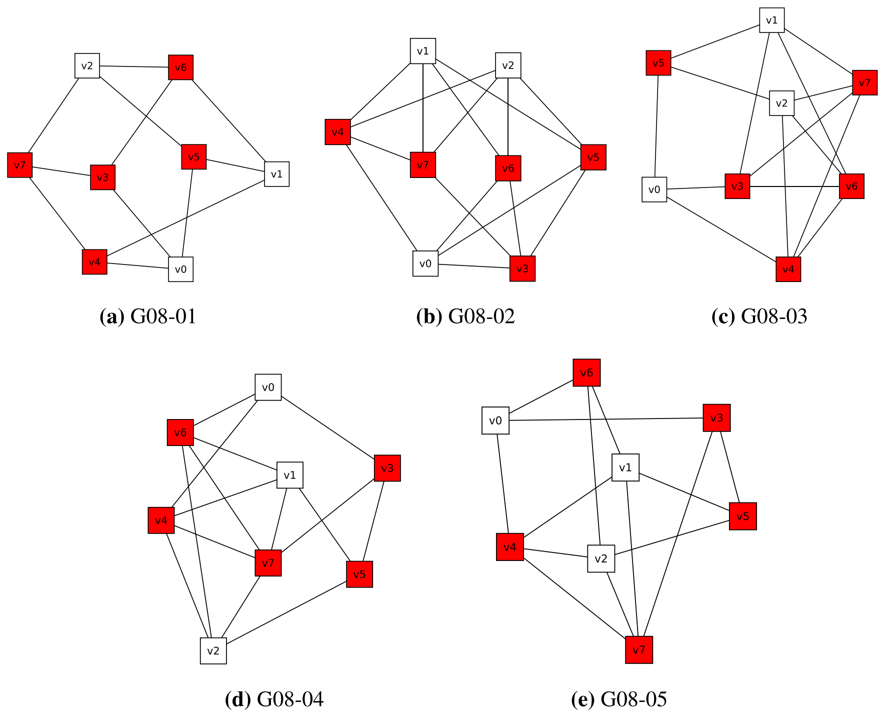


Fig. 1 Irreducible instances of size 8 graphs ($n = 8$ and $k = 5$) after application of all the reductions rules of references Abu-Khzam et al. (2004) and Chen et al. (2010). The vertices in red indicate a minimum cardinality vertex cover. Several vertex covers of size five exist for each of these instances (Color figure online)

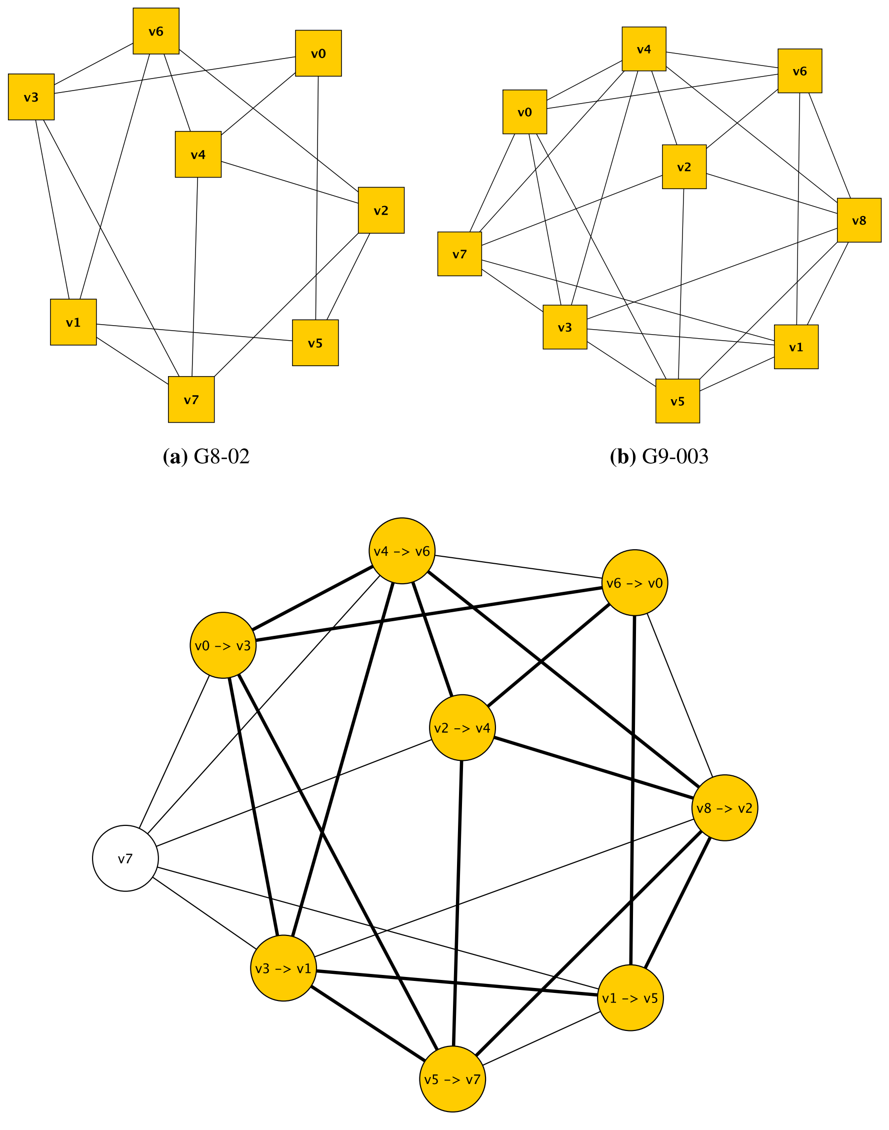
proved to be the case after a reduction rule has been executed. We thus are interested in collecting a database of graphs that are non-reducible by the existing reductions rules and then can be compared with others of smaller size.

3.4 Fourth step: identify common structure

We then investigated if there was some sort of common structure (i.e., subgraph present in some pairs of these sets). We employed a highly effective memetic algorithm for the network alignment problem (Mathieson et al. 2019). This software has been very useful since it is a fast heuristic and allows exploratory insights and scales well for the tasks at hand.

Also, in this case, the results were very interesting and they promise to be informative for the theoretical development of fixed-parameter tractable algorithms. For instance, the 5 non-reduced instances of size 8 do not seem to have a significant common isomorphic subgraph between them when taken in pairs, indicating that perhaps several new reduction rules would be needed to fully reduce these instances (Fig. 2).

The memetic algorithm for network alignment tries to maximize one objective function, the so-called *Edge Correctness (EC)* score. Given an alignment of two networks,



(a) G8-02 **(b)** G9-003

(c) Matching Graph after *NetworkAlignment*(G8-02, G9-003)

Fig. 2 A visual illustration of outcome from the Network Alignment of two graphs. Here, **a** G08-02 graph is aligned with **b** G9-003. **c** The matching graph of $NetworkAlignment$(G8-02, G9-003) is shown. In the matching graph, the vertex labels are formed with the labels from bigger graph to smaller graph, hence $V(G9) \rightarrow V(G8)$. The vertex without any matching will only have the vertex label from the bigger graph, G9. Common or matched edges are shown in bold

the EC score is the ratio of the number of edges from the smaller network mapped to edges of the larger network and the number of edges in the smaller network in total. To understand if there is some common structure between irreducible instances of size 8 (G8) and those irreducible of size 9 (G9), we computed the matrix of the EC scores obtained by our memetic algorithm for all 5 (G8) $\times$ 118 (G9) graphs. Figure 7 shows an example of Network alignment between a pair of G8 and G9 graphs.

We have executed the Network Alignment of G8 with G9 for 20 independent runs of the memetic algorithm we have used. We have used the best alignment result (i.e., the one that maximizes the *Edge Correctness* score in these 20 runs). To make sense of the results we have used the algorithm presented in Moscato et al. (2007) to produce a permutation of rows and columns that maximizes the correlations observed. See Fig. 3 for the result.

3.5 Results of the structural analysis

In the heat map (shown in Fig. 3) the red color represents that the *Edge Correctness* between a pair of graphs is 1.0 (i.e., the graph of eight vertices is a subgraph of the graph of nine). The minimum value of observed *Edge correctness* is 0.75 and it is represented by a Black colour. The average *Edge Correctness* is 0.96 and it is shown using green.

3.5.1 Several non-reducible graphs of size nine have all the non-reducible graphs of size eight as subgraphs

After executing the Network Alignment program by their types of vertex-cover match, we found a relatively large number of (25) size nine non-reducible graphs G9 which that have all the non-reducible graphs of size eight as subgraphs. An example of them is **G09-003** and it is shown in Fig. 7. We also show the network alignment output for the G09-003 which has perfect alignment with all of the G8's in Fig. 1.

3.5.2 One graph in G8 is a subgraph of 114 out of 118 graphs of G9

Since G8-01 is a graph which aligned perfectly (EC = 1.0) with most of the non-reducible graphs of size 9 (114 out of 118) its study may be prioritized among the five non-reducible of the same size. A reduction rule that may safely reduce G8-01 could potentially lead to reducing almost all those of G9.

If this proves successful, we would also have to find potential reduction rules for these four graphs. In the search for common structure, we search for the best pair alignment of these four graphs. The resulting six network alignments are shown in Fig. 4. We have used the yED software editor[4] to visualize the graph, with the 'Hierarchical Layout' with the 'BFS Layering' (see[5] for details of the method). It is clear that all these graphs have an independent set of size three and a vertex cover that includes all the nodes, not in the independent set.

[4] https://www.yworks.com/yed.

[5] http://docs.yworks.com/yfiles/doc/developers-guide/incremental_hierarchical_layouter.html.

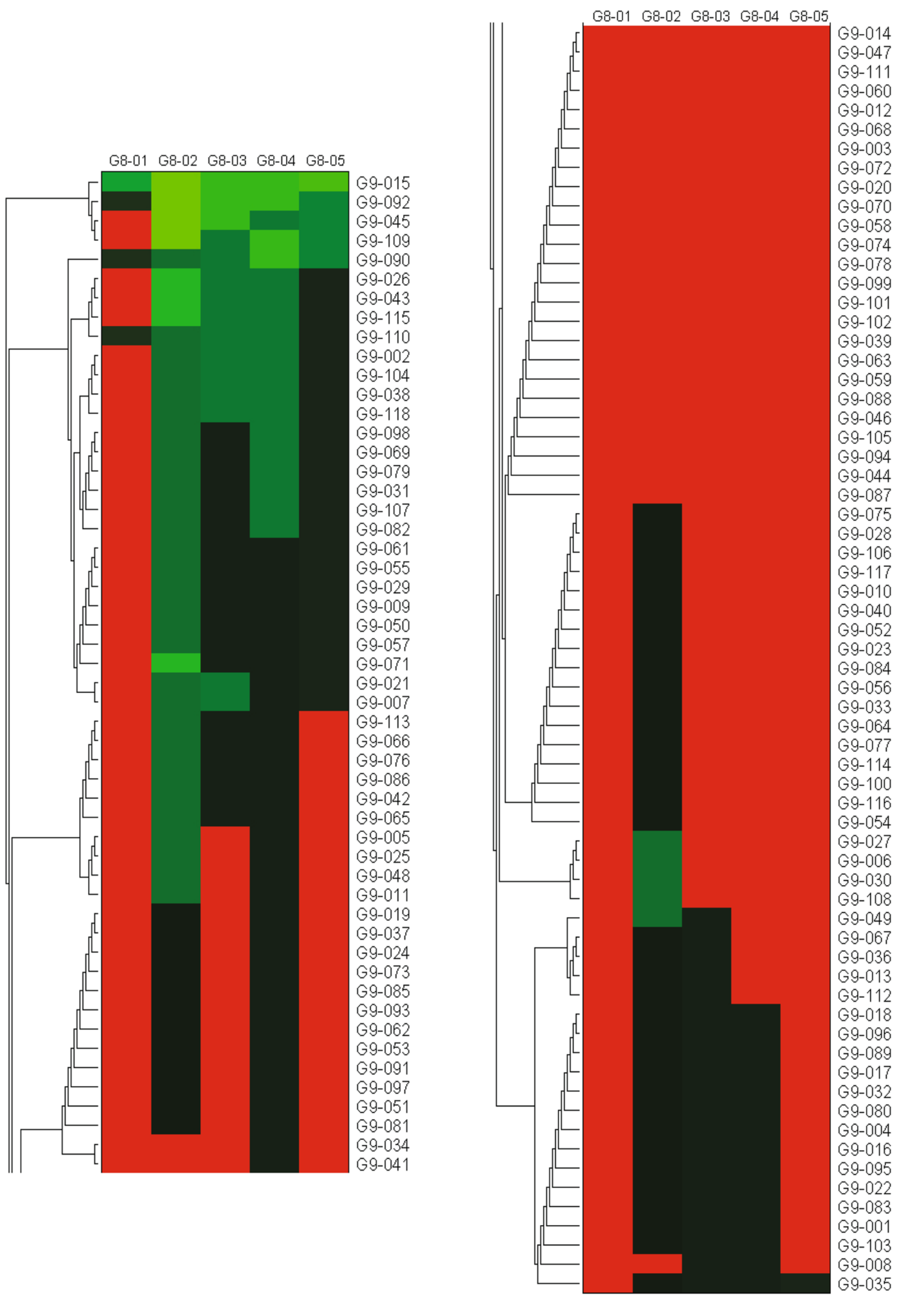


Fig. 3 Heatmap of the *Edge Correctness* scores obtained thanks to the best alignments of G8 with G9. The heatmap has been chopped into two pieces to fit into the page width. The left part is placed at the bottom, while the right part is on the top. The scores are sorted using the memetic algorithm of reference Moscato et al. (2007) using the Euclidean distance as a measure of similarity between rows and columns (Color figure online)

3.5.3 Other alignments

Among 118 G9 graphs, G8-02 aligned perfectly with 28 of them, G8-03 with 64, G8-04 with 51 and G8-05 with 89 of the G9 graphs.

3.5.4 Other characteristics of the graphs of size 9

There are other characteristics of the graphs introduced in Sect. 3.5.2. From Fig. 4 we can observe that `G9-015` has the best alignment with `G9-090` and `G9-092`, only having two unmapped edges. On the other hand, `G9-015` has the worst alignment with `G9-110`, which contains four unmatched edges. We can see `G9-090` achieved the best alignment of having only two unmapped edges with each of the `G9-092` and `G9-110`. Here, `G9-092` have the worst alignment with `G9-110` of having seven unmatched edges.

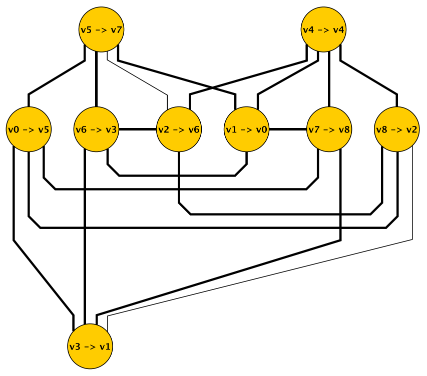
3.6 Partial conclusions

At this stage, it is perhaps relevant to highlight how the computationally-supported process we introduced in this section could augment our intuition in searching for structures that can be exploited to develop reduction rules and fixed-parameter tractable algorithms. We have previously shown in the manuscript that computational results on a large dataset of well-known benchmark instances helped us to illustrate that a set of the 10 reductions rules that build a kernel with the current best fixed-parameter algorithm (in the sense that it has the best theoretical bounds) was actually not very useful in practice. We also questioned what other useful reductions rules have yet to be added to the existing repertoire. Utilising a computational method to augment our intuition, we have identified, using exhaustive enumeration of graphs of orders up to nine, and using powerful methods for network alignment, several small instances which contain structures that the employed reduction rules cannot deal with. Conserved structures can give us some clues of which type of reduction rules are still needed to solve all instances up to size 9.

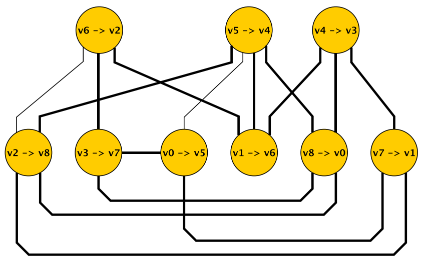
It is now time to turn our attention to what these structures can also give us in terms of the design of heuristics for this problem. In the next section we will look at how other properties of these graphs can give us the necessary intuition to produce heuristics for the Min Vertex Cover problem that find the optimal solutions of all these instances on graphs up to size 9. Later we will show how these new heuristics perform on the same dataset of benchmarking instances used before.

4 Three new heuristics for finding minimum vertex covers based on *k*-class edge scores

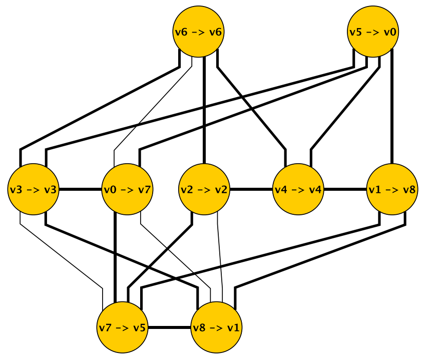
In this section we present three new heuristics based on the intuition we have got from the computational experiments conducted so far. We start by introducing some basic concepts.



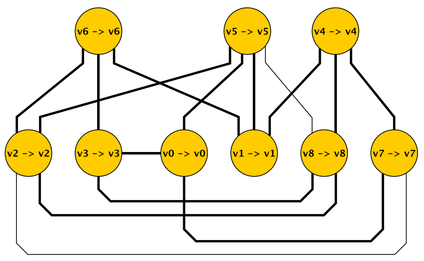
(a) NA(G9-015, G9-090)



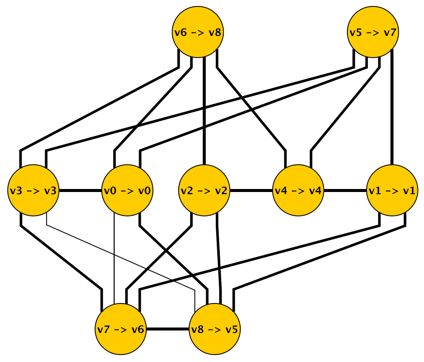
(b) NA(G9-015, G9-092)



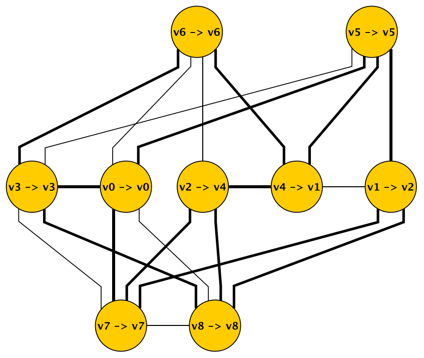
(c) NA(G9-015, G9-110)



(d) NA(G9-090, G9-092)



(e) NA(G9-090, G9-110)



(f) NA(G9-092, G9-110)

Fig. 4 Network Alignment outcome of Four G9 graphs. Vertex label in the aligned graph formed as "vertex label of first graph → vertex label of second graph". Matched edges are marked with bold edges

4.1 "Hierarchical optimal" layering and triangles

Another alternative offered by the software *yED* to layout graphs is called *"Hierarchical Optimal Layering"*. An illustrative example of it running for the same four graphs aligned and implicitly presented in Fig. 4 can be found in Fig. 5. Without having an explicit algorithmic definition of what the software is doing, we have noticed that, in general, the method allocates a "layer" to each node of the graph. Nodes in the same layer seem to constitute an independent set.

If such a layered arrangement can be found in polynomial time by a well-specified algorithm, it may be possible to come up with another polynomial-time algorithm that exploits this structure as a precondition and safely reduces the instance size.

We can also observe in Fig. 5 that the graphs have connections between layers and that they have a relatively large number of *triangles*, i.e., cliques of size three in the graph. These extra insights have motivated new heuristic approaches. Before we discuss them we need to introduce some definitions for clarity.

4.2 The truss decomposition of a graph and other necessary definitions

With the new intuition gained from this analysis, we are now proposing several heuristic approaches. We will also use the information of a *triangle-based decomposition* of the graph via the *k-truss* (Wang and Cheng 2012).

Definition 1 (k-truss of a graph) The *k-truss of a graph* $G = (V, E)$ is the largest subgraph of G in which every edge is contained in at least $(k-2)$ triangles within the subgraph.

Consequently, the *k-truss decomposition of a graph* G is the set of all non-empty k-trusses of G for all $k \geq 2$.

Definition 2 (Truss number): The *truss number* of an edge $e \in E$ of a graph $G(V, E)$ is the number of triangles in G that contain e.

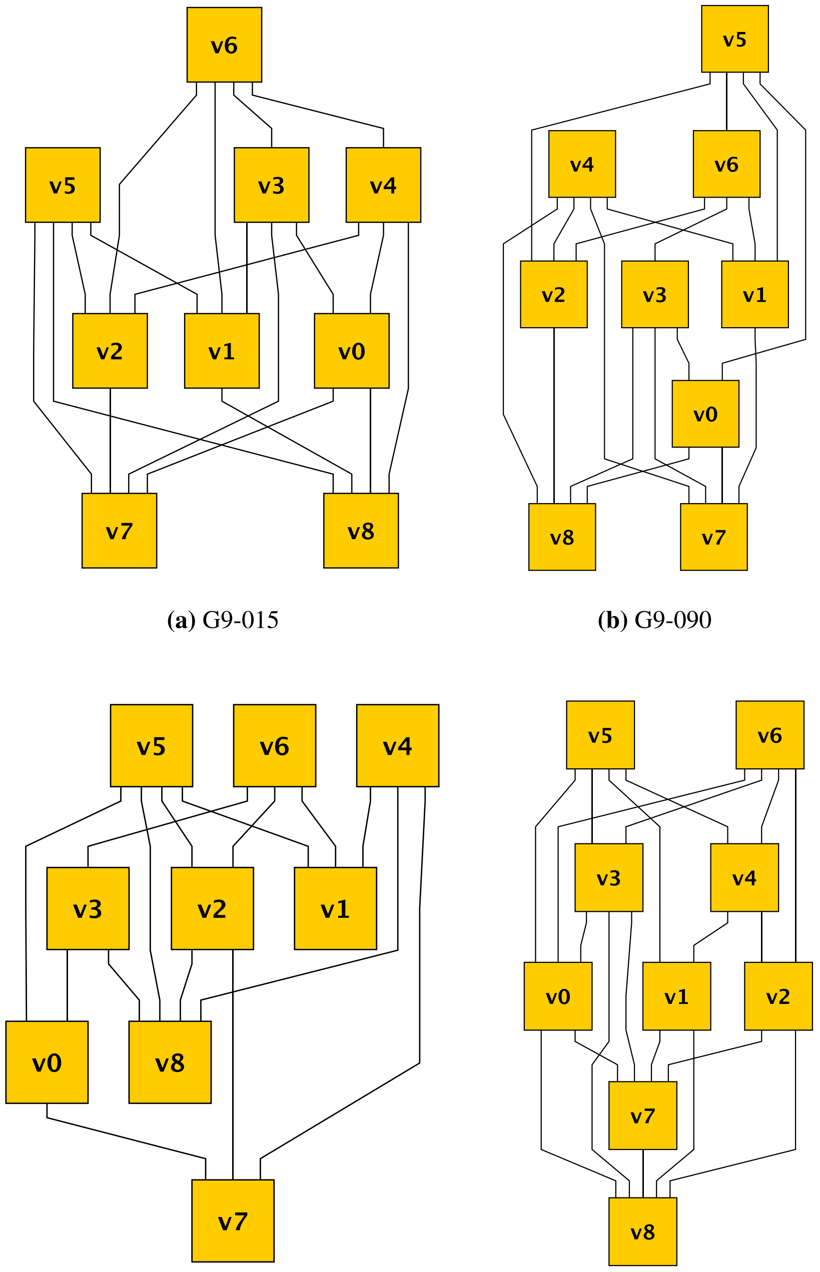
We can then define the *k-class score of an edge* as a function of the truss numbers as:

Definition 3 (k -class edge score): The *k-class score of an edge* $e \in E$, denoted as $kcEdgeScore(e)$, is the maximum truss number (k) of the edge obtained from the k-truss decomposition of the graph.

We also need to define another score that will be useful later on when proposing some heuristics in Sect. 4.6.

Definition 4 (k-class score sum of a vertex): The *k-class score sum of a vertex* in a k-truss decomposition of a graph G is defined as

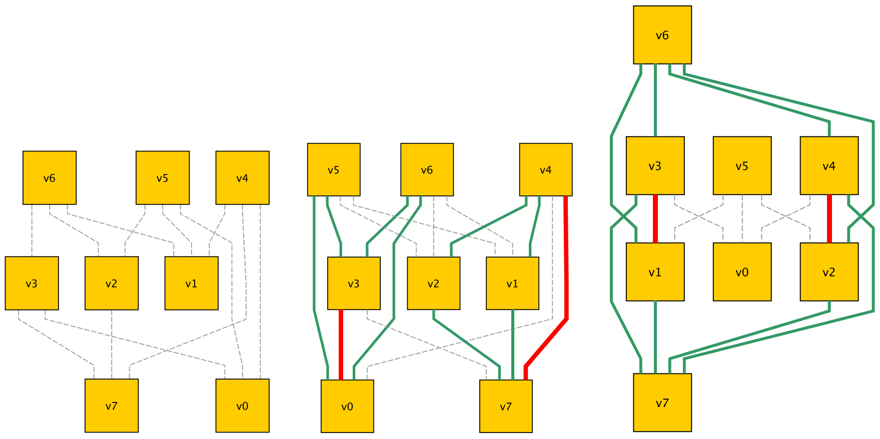
$$kcVertexScore(u) = \sum_{v \in N(u)} kcEdgeScore((uv)) \qquad (1)$$



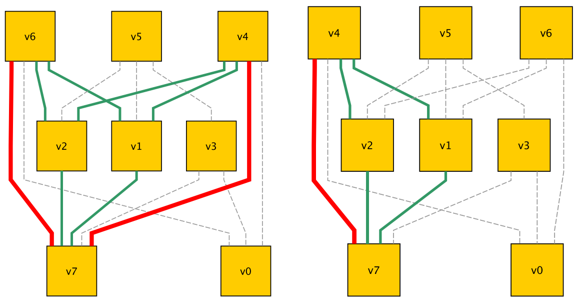
(a) G9-015 **(b)** G9-090

(c) G9-092 **(d)** G9-110

Fig. 5 Four *The 10-R*-irreducible G9 graphs displayed using the in 'Hierarchical Optimal Layering' option of the *yED* software, revealing their edge density and an interesting structure of layers of independent sets and a relative high density of triangles



(a) k-class edges of G8-01 **(b)** k-class edges of G8-02 **(c)** k-class edges of G8-03



(d) k-class edges of G8-04 **(e)** k-class edges of G8-05

Fig. 6 The G8 graphs and their k-class edges ($3 \le k \le 4$). The dashed-edges in G8 graphs correspond to the non k-class edges (≤ 2). In the G8 graphs, we have two values of k-classes (3 and 4). Here 3-class edges are coloured with green and 4-class edges are with red (Color figure online)

for all $v \in N(u)$, i.e., the sum of the k-class edge scores for all of the incident edges of node u.

We show the k-truss decomposition of the G8 graphs in Fig. 6. Here, the graph G8-01 does not have any triangles, so it has no k-truss decomposition available. Here, an edge with k-class score of 4 denotes that it is incident with a maximum of $(4-2) = 2$ triangles in the graph. Now we describe the algorithm in detail.

4.3 The Extended Berra "Lemma"

Although we are not aware of any contributions to the mathematics of professional baseball player and coach Mr. Lawrence Peter "Yogi" Berra, he is well known due to many of his quotes, many of them hilarious, which are nowadays famous in popular culture. One that in particular motivates the name to our branching strategy:

"When you come to a fork in the road, take it."

so in the context of celebrating this quote we can write:

Lemma 2 (Berra "Lemma") *Let* (G, k) *be a reduced instance of* k- VERTEX COVER *and* $uv \in E(G)$. *If* $(G - v - u, k - 2)$ *is completely reducible, take it.*

In fact, Gavril's 2-approximation ratio can be seen as a randomized heuristic that finds at random "a fork in the road" (an edge) and includes the nodes that it connects in the vertex cover. However, it does it without any guarantee that the remaining graph is reducible. There is also no guarantee that the endpoints should be both in an optimal vertex cover.

An extension into a three-way branching seems natural:

Lemma 3 (Extended Berra "Lemma") *Let* (G, k) *be a reduced instance of* k- VERTEX COVER *and* $uv \in E(G)$. *If* $(G - u, k - 1)$ *or* $(G - v, k - 1)$, *or* $(G - v - u, k - 2)$ *is completely reducible, take it.*

meaning that, when considering an edge (u, v), we will search in parallel both parameterized problem alternatives (i.e., either u or v can be part of a vertex cover).

4.4 Final check for redundancies: the necessary final step

Take any one of the non-reducible graphs of size 8 (G8) and apply the Extended Berra "Lemma" to an arbitrarily chosen edge. We are in one of the three-way splits; assume that u is assigned to be in the vertex cover. The resulting graph is reducible by the *The 10-R* algorithm because it is of size 7, and we know that any graph of that size is reducible. Still, we need to check that the nodes of this vertex cover of this graph of size 7 do not contain the neighborhood of u in the original graph of size 8 as a subset. If that is the case, it is not necessary to include u. This means that the decisions after the application of the Berra "Lemma" necessarily need to be revisited. Our heuristic will then use a final pass to check that no vertex in the cover is "redundant" since all its neighbors are already in the cover.

4.5 Greediness guided by a function of the *k*-class edge scores

Note also that an arbitrary edge of a graph in G8 would lead to a reducible graph, so the Berra "Lemma" only needs to be applied once. However, for a larger graph, we may need to have a guiding function to decide on which edge to "fork". Several heuristics for Min Vertex Cover have used a number of strategies, e.g. selecting a node of maximum degree and, when there are ties, the node u that minimizes the number of edges among $N(u)$. While other more complicated branching rules exist (see for instance Akiba and Iwata 2016), we present one here based on triangles and the k-class edge scores which is novel in the area. More importantly, it is directly motivated by our computational experiment, which has clearly augmented our intuition about the problem, and this is a main research theme of our study.

4.6 Three proposed heuristics

To illustrate the results of the experimental process, we present three heuristics motivated specifically by empirical observation of the structure of these instances. Later, in Sect. 5 we demonstrate their effectiveness in comparison to the prior results of the exact reduction algorithms on the benchmarking datasets.

4.6.1 Heuristic 1

The first heuristic employs the k-class score sum of the vertex (defined in Eq. 4.2) to prioritise vertices for inclusion in the vertex cover. Given a graph $G = (V, E)$ as input, the heuristic first sorts the vertices into a list L by k-class score sum of vertex (highest first), then by highest degree among vertices with the same k-class score sum and finally with ties broken uniformly at random.

The heuristic then proceeds to build a vertex cover by successively selecting the next available vertex v from the list and examining its neighborhood. If v has the highest k-class score sum among its neighbors, it is added to the cover and removed from the graph (along with its incident edges, of course). If there is at least one other vertex with the same k-class score sum in its neighborhood, all such neighbors are collected into a set N_v^k and the heuristic branches on either adding v to the cover and removing it and all its incident edges, or adding a randomly selected $w \in N_v^k$ to the cover and removing it and all its incident edges. In all cases, the removed vertex is also deleted from L, the degrees and k-class score sum of the remaining vertices are updated and L is resorted.

A branch of the heuristic terminates successfully when either a cover is found in some sub-branch (i.e., when all edges in G have been removed), or unsuccessfully when the size of the cover exceeds the parameter k in all sub-branches. Finally, we do a final check to remove any vertices where all of its neighbors are already in cover and return the smallest cover C.

The pseudocode is given in Heuristic 1.

4.6.2 Heuristic 2

The second heuristic for Min Vertex Cover is based on k-class edge score (defined in Eq. 3), the maximum degree of the vertices and the number of edges among neighbors of a vertex. Heuristic 2 gives the pseudocode.

The heuristic receives the graph $G = (V, E)$ as the input. It iteratively adds a vertex, v, into the cover C and removes the vertex and all its incident edges from G. This process repeats until the graph is completely disconnected. To chose a vertex to add to the cover C, the heuristic considers several scores. First, it selects the set of edges E' with the maximum k-class edge score and chooses the endpoint with the maximum degree from those edges. If there are multiple vertices with the maximum degree, then we select the vertex whose neighbors have the minimum number of common edges. Any further tie is broken by choosing a vertex uniformly at random. The chosen vertex v is added to the cover C. Then we remove v and its incident edges from the graph, and update associated scores (k-class edge score, degree etc.).

Heuristic 1:

```
Data: A graph $G = (V, E)$
Result: A vertex cover $C \subseteq V$ of size at most $k$ for $G$
1  initialization: Compute the $k$-class score sum for each vertex
2  $C := \emptyset$
3  while $E \neq \emptyset$ and $|C| \leq k$ do
4      $L := V$ sorted first by highest $k$-class score sum, then by highest degree
5      $v :=$ first element of $L$
6      $N_v^k := \{u \in N(v) \mid kcVertexScore(u) = kcVertexScore(v)\}$
7      if $N_v^k = \emptyset$ then
8          $C := C \cup \{v\}$
9          $G := G \setminus \{v\}$
10     else
11         Select $w \in N_v^k$ uniformly at random
12         Branch: on adding $v$ to $C$ or adding $w$ to $C$
13             case $v$
14                 $C := C \cup \{v\}$
15                 $G := G \setminus \{v\}$
16             case $w$
17                 $C := C \cup \{w\}$
18                 $G := G \setminus \{w\}$
19 if All branches give $|C| > k$ then
20     return NO COVER
21 else
       // Final Check to remove vertices already covered
22     Remove all Vertex $c$ from $C$ where $N(c) \in C$
23     return Smallest $C$
```

After the main loop is complete the C is checked for vertices whose entire neighborhood is also in C. Such redundant vertices are removed. The cover C is then removed.

4.6.3 Heuristic 3

The third heuristic is based on considering the degree of the vertices, the degree imbalance of edge end-points, the k-class edge score. The whole process is illustrated by the pseudocode of Heuristic 3.

This heuristic receives the graph $G = (V, E)$ as the input. It iterates by adding a vertex, v, at a time into the cover C and by removing the vertex from G. This process repeats until all edges are being removed. To chose a vertex in the cover C, the heuristic considers several scores. Initially it finds the set of vertices V_m with minimum degree. Form V_m, we compute the set of edges $E' = \{(u, v) \in E\}$ where at least one vertex of the edge is at V_m and we compute the absolute degree imbalance ($|d(u) - d(v)|$) between the two endpoints of the edges in E'. If $|E'| = \{e\}$ (i.e., if there is a unique edge with the maximum degree imbalance), the endpoint of e with the highest degree is chosen as the vertex to be added in the cover. If $|E'| > 1$, we consider the k-class edge score. From E' we select the set of edges E'' with the maximum k-class edge score and we add the end-point with the maximum degree into the cover C. Any

Heuristic 2:

```
   Data: A graph $G = (V, E)$
   Result: A vertex cover $C \subseteq V$
1  initialization: $k$-class edge score and degree of Vertices
2  $C := \emptyset$
3  while $E \neq \emptyset$ do
       // Consider the $k$-class edge scores
4      $E' := \{(u, w) \in E' \mid \forall (x, y) \in E', kcEdgeScore((u, w)) \geq kcEdgeScore((x, y))\}$
       // Consider Max Deg vertices from $k$-class edges
5      $V' := \{w \in V(E') \mid \forall w \in V(E'), d(w) = \max_{\{u \in V(E')\}} d(u)\}$
6      if $V' = \{w\}$ then
7          $v := w$
8      else
           // Consider vertex with Min #common edges among neighbors
9          $V'' := \{w \in V' \mid \forall u \in V', \min_{(w,u) \in V'(E')} \{\mathbf{C}(w), \mathbf{C}(u)\},$ where $\mathbf{C}(z) =$
           count the number of edges $(x, y) \in N(z)\}$
10         if $V'' = \{w\}$ then
11             $v := w$
12         else
13             Choose vertex $v$ from $V''$ uniformly at random
           // add $v$ to cover, update graph and scores
14         $C := C \cup \{v\}$
15         $G := G \setminus \{v\}$
16         Update: $k$-Class Edge Score and degree for both $v$ and $N(v)$
   // Final Check to remove vertices already covered
17 Remove all Vertex $c$ from $C$ where $N(c) \subseteq C$
18 Return $C$
```

remaining tie is broken uniformly at random. The chosen vertex, v, is then added into the cover C. We then remove v from G, and update associated scores (k-class edge score, degree imbalance, etc.). We continue this process until the graph is completely disconnected. Finally, if there exists any vertex $v \in C$ where $N(v) \subset C$ such that the deletion of it from cover does not leave an edge uncovered, then we remove v from C. We then return the cover C.

5 Results of the proposed heuristics on benchmark datasets

We have applied the three proposed heuristics for Min Vertex Cover problem on the Complement graphs of Clique Instances form DIMACS challenge datasets. We executed the algorithms on the Research Compute Grid (RCG)[6] allocated with 2 CPU cores, 8GB memory and 100 hours of CPU wall time per instances. We report the best performance of the proposed heuristics over the branch-and-reduce based exact algorithm (FPT) implemented in Akiba and Iwata (2016). For some instances, the FPT-based exact solution method or the heuristics exhausted the maximum allocated

[6] The description and configuration of the RCG of University of Newcastle, Australia can be accessed at: https://www.newcastle.edu.au/research-and-innovation/resources/research-computing-services/advanced-computing.

Heuristic 3:

```
Data: A graph $G = (V, E)$
Result: A vertex cover $C \subseteq V$
1  $C := \emptyset$
2  while $E \neq \emptyset$ do
       // Find the vertices with minimum degree
3      $V_m = \{v \in V : d(v) = \min_{u \in V} d(u)\}$
       // Candidate Edges for Maximum Degree Imbalance Score with at
          least one vertex of the edge has a vertex with min degree
4      $E' := \{(u, w) \in E : \forall(x, y) \in E : (x \in V_m) \vee (y \in V_m), |d(u) - d(w)| \geq |d(x) - d(y)|\}$
5      if $E' = \{(u, w)\}$ then
6          if $d(u) > d(w)$ then
7              $v := u$
8          else
9              $v := w$
10     else
           // Consider Cand. Edge with Max $k$-class edge score
11         $E'' := \{(u, w) \in E' : \forall(x, y) \in E', kcEdgeScore((u, w)) \geq kcEdgeScore((x, y))\}$
           // Consider Max Deg Vertex from Cand. Edges
12         $V' := \{w \in V(E'') : \forall w \in V(E''), d(w) = \max_{\{u \in V(E'')\}} d(u)\}$
13         if $V' = \{w\}$ then
14             $v := w$
15         else
16             Choose a vertex $v$ uniformly at random from $V'$
       // add $v$ to cover, update graph and scores
17     $C := C \cup \{v\}$
18     $G := G \setminus \{v\}$
19     Update: $k$-Class Edge Score and degree for both $v$ and $N(v)$
   // Final Check to remove vertices already covered
20 Remove all Vertex $v$ from $C$ where $N(v) \subseteq C$
21 Return $C$
```

CPU-time before it reached any solution. In that case, we put a dash '–' mark in a cell of the reported table.

5.1 Reduction outcome by the proposed heuristics on BHOSLIB Dataset

Ke Xu from the Beihang University, Beijing, China maintains a benchmark dataset for graph problems, named BHOSLIB,[7] which contains 40 instances for Min Vertex Cover problem. We have used those instances to compare the performance of our proposed Heuristics in Table 2. In the comparison of the best results obtained by the three proposed heuristics, the Heuristic 1 ($h1$) achieved the best performances in 8, Heuristic 2 ($h2$) in 21 and Heuristic 3 ($h3$) in 19 instances from the BHOLIB dataset. The runtime requirements for t_{h3} are significantly lower than that of t_{h1} and t_{h2}.

[7] BHOSLIB: "Benchmarks with Hidden Optimum Solutions for Graph Problems" can be accessed at: http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm.

Table 2 Execution outcome of proposed heuristics for Min Vertex Cover problem on the BHOSLIB instances

Instance	|V|	|E|	C	C_{h1}	t_{h1}	C_{h2}	t_{h2}	C_{h3}	t_{h3}
frb30-15-1	450	17827	420	424	387.29	**422**	2.904	425	0.042
frb30-15-2	450	17874	420	**425**	388.685	427	2.867	426	0.037
frb30-15-3	450	17809	420	426	324.272	**423**	2.513	425	0.033
frb30-15-4	450	17831	420	**422**	331.084	428	2.695	425	0.041
frb30-15-5	450	17794	420	**424**	328.129	427	4.496	**424**	0.041
frb35-17-1	595	27856	560	567	1058.915	**566**	7.57	**566**	0.053
frb35-17-2	595	27847	560	567	1056.902	568	9.304	**565**	0.053
frb35-17-3	595	27931	560	**565**	1031.078	566	8.997	566	0.049
frb35-17-4	595	27842	560	**565**	1043.462	566	7.335	567	0.057
frb35-17-5	595	28143	560	**566**	1070.425	569	6.988	**566**	0.06
frb40-19-1	760	41314	720	**727**	2673.72	729	17.369	728	0.073
frb40-19-2	760	41263	720	**726**	2209.123	729	16.624	727	0.067
frb40-19-3	760	41095	720	728	2322.429	728	13.349	**727**	0.071
frb40-19-4	760	41605	720	728	2569.759	729	18.048	**727**	0.062
frb40-19-5	760	41619	720	727	2451.046	732	16.304	**726**	0.071
frb45-21-1	945	59186	900	909	5636.558	**904**	24.452	910	0.089
frb45-21-2	945	58624	900	908	5381.902	**904**	27.533	907	0.093
frb45-21-3	945	58245	900	910	6972.466	**908**	30.093	909	0.103
frb45-21-4	945	58549	900	910	6596.334	**906**	24.871	907	0.079
frb45-21-5	945	58579	900	909	6934.587	**908**	25.622	**908**	0.091
frb50-23-1	1150	80072	1100	1110	12756.98	**1109**	47.294	1111	0.112
frb50-23-2	1150	80851	1100	1110	12771.76	**1108**	40.975	1109	0.109
frb50-23-3	1150	81068	1100	1109	14931.49	1114	39.817	**1108**	0.119
frb50-23-4	1150	80258	1100	1110	12410.98	**1108**	44.708	**1108**	0.108
frb50-23-5	1150	80035	1100	1110	13259.71	**1101**	41.368	1109	0.292
frb53-24-1	1272	94227	1219	1230	23487.67	**1227**	62.186	**1227**	0.137
frb53-24-2	1272	94289	1219	1230	19483.75	**1227**	45.042	1229	0.146
frb53-24-3	1272	94127	1219	1229	22736.39	**1227**	60.461	**1227**	0.131
frb53-24-4	1272	94308	1219	1229	22992.61	**1227**	59.624	**1227**	0.135
frb53-24-5	1272	94226	1219	1232	28264.63	**1227**	46.994	1229	0.132
frb56-25-1	1400	109676	1344	1355	36604.91	1355	82.179	**1353**	0.138
frb56-25-2	1400	109401	1344	1356	32926.01	1354	65.224	**1353**	0.148
frb56-25-3	1400	109379	1344	1356	30681.63	1355	67.709	**1353**	0.151
frb56-25-4	1400	110038	1344	1357	32807.77	**1348**	71.012	1354	0.154
frb56-25-5	1400	109601	1344	1356	29201	1356	68.598	**1353**	0.149

Table 2 continued

Instance	|V|	|E|	C	C_{h1}	t_{h1}	C_{h2}	t_{h2}	C_{h3}	t_{h3}
frb59-26-1	1534	126555	1475	1488	41895.53	**1479**	89.848	1487	0.189
frb59-26-2	1534	126163	1475	1488	40358.87	**1484**	87.702	1487	0.187
frb59-26-3	1534	126082	1475	1489	60933.59	1486	80.33	**1485**	0.372
frb59-26-4	1534	127011	1475	1488	52556.86	**1480**	80.023	1485	0.203
frb59-26-5	1534	125982	1475	1487	68168.12	1487	71.455	**1484**	0.16

Here we show the size of Exact cover (**C**) and the running time (t_{h_i}) required to find the Cover size (C_{h_i}) found by our heuristics h_i (where $i = 1, \ldots, 3$)

5.2 Reduction outcome by the proposed heuristics on the DIMACS graphs for clique (complement instances) problem

We have applied the heuristics on complement graph of the Maximum Clique problem of DIMACS Challenge. The results are shown in Table 3. In the comparison of the best results obtained by the three proposed heuristics, Heuristic 1 ($h1$) achieved the best performances for 27 instances, Heuristic 2 ($h2$) in 26 and Heuristic 3 ($h3$) in 59 instances from the DIMACS dataset. Both $h1$ and $h2$ were unable to complete the execution for 10 and 1 instances respectively, in the allocated time of 360,000 s. In contrast, $h3$ was able to produce the results within a fraction of the time.

5.3 Statistical test of the performances of proposed three heuristics

We used the vertex cover size obtained for each of the heuristics ($h1$, $h2$, $h3$) presented in Tables 2 and 3. We consolidated only results of those instances where all three heuristics were able to complete the execution within the allocated CPU times for the statistical test (since it is already clear that $h3$ was the most successful in delivering a cover for all instances). Now we want to find if there are any significant differences in the performances of finding the size of a vertex cover by the heuristics on the consolidated 108 instances from BHOSLIB and DIMACS datasets. A Friedman rank-sum test on those data returned the Friedman chi-squared statistic = 28.155556 with a p-value = $7.693054\ 10^{-7}$. This p-value rejects the omnibus null hypothesis, hence, "*Significant Differences exist in the ranking of the heuristics*".

The omnibus p-value is way below the respectable critical threshold of 0.05, so we conduct a post-hoc test to find which of the pairs have significant differences in their performances. The p-value obtained for the pairwise comparisons using Nemenyi post hoc test is presented in heatmap shown in Fig. 7a. We also computed a critical difference plot [as proposed in Demsar (2006)] to show the significant differences of the heuristics for their ranking in Fig. 7b.

Both the heatmap and critical difference plot revealed that there exist no significant differences between $h1$ and $h2$. However, the $h3$ shows significant differences in the performance than both $h1$ and $h2$. The critical plot illustrates the median ranking of both heuristics $h1$ and $h2$ are within the rank of 2 and 3. The median ranking of the $h3$

Table 3 Execution outcome of proposed heuristics for Min Vertex Cover problem on the complement graphs of Clique instances from DIMACS challenge dataset

Instance	|V|	|E|	C	C_{h1}	t_{h1}	C_{h2}	t_{h2}	C_{h3}	t_{h3}
brock200_1	200	5066	179	182	11.8	182	0.6	**181**	0.5
brock200_2	200	10024	188	**191**	43.4	192	4.5	**191**	0.6
brock200_3	200	7852	185	188	26.4	190	1.4	**187**	0.5
brock200_4	200	6811	183	187	25.5	188	1.4	**185**	0.6
brock400_1	400	20077	373	**377**	389.3	380	5.2	378	1.7
brock400_2	400	20014	371	379	385.2	383	3.2	**378**	1.4
brock400_3	400	20119	369	380	402.4	380	3.5	**379**	0.9
brock400_4	400	20035	367	378	369.3	382	3	**377**	1
brock800_1	800	112095	777	784	22490.3	785	259.1	**782**	4.9
brock800_2	800	111434	776	782	18583.1	**779**	246.2	782	5
brock800_3	800	112267	775	784	63186.5	**778**	278.6	783	5.6
brock800_4	800	111957	774	785	27004.7	**780**	252.1	783	7.6
C1000.9	1000	49421	–	948	6109.9	948	12.8	**941**	3.3
C125.9	125	787	91	94	1.6	94	1	**93**	0.3
C2000.5	2000	999164	1984	–	–	**1986**	73305.8	1,987	64.2
C2000.9	2000	199468	–	–	–	**1931**	163.5	**1931**	14.6
C250.9	250	3141	206	**210**	11.3	213	1.2	211	0.5
C4000.5	4000	3997732	–	–	–	–	–	**3985**	493.9
C500.9	500	12418	–	451	233	455	1.9	**448**	1.3

Table 3 continued

Instance	|V|	|E|	C	C_{h1}	t_{h1}	C_{h2}	t_{h2}	C_{h3}	t_{h3}
c-fat200-1	200	18366	188	**188**	115.4	**188**	16	**188**	4.9
c-fat200-2	200	16665	176	**176**	95.8	178	14.5	**176**	3.4
c-fat200-5	200	11427	142	**142**	47.8	**142**	79.2	**142**	4.1
c-fat500-1	500	120291	486	**486**	12870	**486**	1462.3	**486**	59.6
c-fat500-10	500	78123	374	**374**	4869	**374**	265.2	**374**	140.9
c-fat500-2	500	115611	474	**474**	13148.1	476	1325.8	**474**	32.2
c-fat500-5	500	101559	436	**436**	7249	438	707.3	**436**	72.8
gen200_p0.9_44	200	1990	156	165	4.8	166	1.1	**163**	0.4
gen200_p0.9_55	200	1990	145	**156**	6.1	163	0.7	162	0.3
gen400_p0.9_55	400	7980	345	**353**	81.1	356	1.1	**353**	0.7
gen400_p0.9_65	400	7980	335	**353**	88.4	356	1.4	355	0.7
gen400_p0.9_75	400	7980	325	**335**	85.7	343	1	352	0.7
hamming10-2	1024	5120	512	577	98042.2	556	270.2	**512**	1.1
hamming10-4	1024	89600	–	**992**	17729.9	993	45058.3	**992**	16.2
hamming6-2	64	192	32	34	0.4	**32**	0.5	**32**	0.1
hamming6-4	64	1312	60	**60**	1.2	**60**	2	**60**	0.5
hamming8-2	256	1024	128	140	66.9	**128**	3.6	**128**	0.3
hamming8-4	256	11776	240	**240**	79.1	241	140.3	**240**	1.8

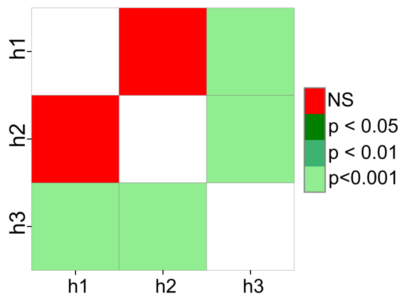
Table 3 continued

Instance	|V|	|E|	C	C_{h1}	t_{h1}	C_{h2}	t_{h2}	C_{h3}	t_{h3}
johnson16-2-4	120	1680	112	**112**	3.2	**112**	9.2	**112**	0.9
johnson32-2-4	496	14880	480	**480**	781.9	483	4052	**480**	19.6
johnson8-2-4	28	168	24	**24**	0.1	**24**	0.5	**24**	0.1
johnson8-4-4	70	560	56	57	0.5	**56**	1	**56**	0.2
keller4	171	5100	160	162	11.2	**161**	5.5	163	1
keller5	776	74710	749	758	6926.2	**757**	91.5	760	5.7
keller6	3361	1026582	–	–	–	**3233**	23328	3329	107.7
MANN_a27	378	702	252	261	524.3	260	6.7	**253**	0.9
MANN_a45	1035	1980	690	705	106162.2	705	139.5	**693**	2
MANN_a81	3321	6480	–	–	–	2241	7072.6	**2225**	8.7
MANN_a9	45	72	29	**29**	0.2	**29**	0.7	**29**	0.1
p_hat1000-1	1000	377247	990	–	–	**991**	13267.4	**991**	13.4
p_hat1000-2	1000	254701	954	–	–	959	3421	**955**	7.1
p_hat1000-3	1000	127754	–	941	25142.3	941	314.6	**938**	4
p_hat1500-1	1500	839327	1488	–	–	1492	101585.9	**1490**	35.3
p_hat1500-2	1500	555290	1435	–	–	1441	26743.3	**1438**	22.3
p_hat1500-3	1500	277006	–	–	–	1415	2165.4	**1413**	11.4
p_hat300-1	300	33917	292	**293**	547.7	294	53.2	**293**	1.4
p_hat300-2	300	22922	275	276	245.1	278	15.9	**275**	1.1
p_hat300-3	300	11460	264	**266**	76.8	271	1.8	**266**	0.6
p_hat500-1	500	93181	491	**492**	6973.2	493	544.2	**492**	2.5
p_hat500-2	500	61804	464	**464**	2528.2	467	149.2	466	1.8
p_hat500-3	500	30950	450	**452**	827.5	458	14	455	1.3
p_hat700-1	700	183651	689	**692**	38469.1	693	2250	**692**	5.2

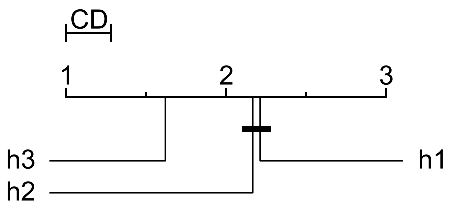
Table 3 continued

Instance	|V|	|E|	C	C_{h1}	t_{h1}	C_{h2}	t_{h2}	C_{h3}	t_{h3}
p_hat700-2	700	122922	656	**657**	13448.2	662	711.9	**657**	3.5
p_hat700-3	700	61640	638	642	4243.1	644	54.3	**641**	1.9
san1000	1000	249000	985	992	108694	992	2184.8	**990**	88.9
san200_0.7_1	200	5970	170	185	15.5	**184**	0.7	**184**	0.6
san200_0.7_2	200	5970	182	188	14	187	0.8	**185**	0.8
san200_0.9_1	200	1990	130	155	6.4	**151**	0.7	152	0.3
san200_0.9_2	200	1990	140	165	6.5	163	0.5	**159**	0.3
san200_0.9_3	200	1990	156	167	5	166	0.5	**165**	0.4
san400_0.5_1	400	39900	387	393	850.8	**392**	29.3	**392**	3.7
san400_0.7_1	400	23940	360	380	404.6	**362**	11	378	1.3
san400_0.7_2	400	23940	370	385	346.2	**376**	5.6	382	1.7
san400_0.7_3	400	23940	378	388	508.2	**381**	7	385	2.2
san400_0.9_1	400	7980	300	**315**	86.8	328	1.3	318	0.8
sanr200_0.7	200	6032	182	184	15.4	185	0.7	**183**	0.4
sanr200_0.9	200	2037	158	162	5.6	165	1.1	**160**	0.3
sanr400_0.5	400	39816	387	390	1161.6	389	28.3	**388**	1.4
sanr400_0.7	400	23931	379	383	388.9	383	4.4	**382**	1.1

Here we show the size of Exact cover (C) and the running time (t_{hi}) required to find the Cover size (C_{hi}) found by our heuristics hi (where $i = 1, \ldots, 3$)



(a) Heatmap of p-values



(b) Critical Difference Plot

Fig. 7 Visual illustration of statistical significance test for the rankings of the proposed heuristics ($h1$, $h2$, $h3$) on BHOSLIB (in Table 2) and DIMACS (in Table 3) instances. Here, **a** The Heatmap showing the p-value obtained for Nemenyi post hoc test. **b** Critical Difference (CD) plot showing the heuristics ordered on a line for their median ranking and statistically non-significant heuristics are connected by horizontal line where Critical difference = 0.279282

is between 1 and 2. Form these statistical tests, it is evident that both of the $h1$ and $h2$ are outperformed by $h3$. So, we will use the proposed heuristic $h3$ in the remaining sections of the paper.

5.4 Reduction outcome by the proposed *h*3 heuristic on a small subset of DIMACS Datasets used by Asgeirsson and Stein (2005)

A small subset form DIMACS instances and the Clique complements are used in Asgeirsson and Stein (2005). We have compared the outcome of our $h3$ with their results in Table 4. We refer to Asgeirsson and Stein's method as hp. The name of clique, coloring and clique-complement instances are ended with .clq, .col and .clq$'$, respectively.

5.4.1 Statistical test of results obtained for datasets used by Asgeirsson and Stein (2005)

Wilcoxon signed-ranks test (Wilcoxon 1992), is a non-parametric statistical hypothesis test, is being suggested as a useful statistical test to compare two algorithms over multiple datasets (Demsar 2006). We used the Wilcoxon signed-ranks test[8] to find if there any difference exists between hp and $h3$ for the results presented in Table 4.

We have calculated both the W-value and z-value and we are reporting them here. However, since we have less than 20 pairs of samples to compare in Table 4, we will use the W-value to evaluate the hypothesis. We found three cases of the tie, which yields the number of samples $N = 12$. For those samples with a confidence level $\alpha = 0.05$, the Two-sided Wilcoxon Signed-Rank test revealed the W-value=9.5. According to the table of exact critical values for the Wilcoxon's test, the critical value for W at $N = 12$ (for $p < 0.05$) is smaller or equal to 13. Hence, the result is significant at

8 https://www.socscistatistics.com/tests/signedranks/.

Table 4 Execution outcome of proposed heuristic 3 for Min Vertex Cover problem on a small subset of DIMACS datasets used by Asgeirsson and Stein (2005)

Instance	C	C_{hp}	C_{h3}	t_{h3}
brock800_3.clq	789	794	**791**	7.7
c-fat500-2.clq	480	481	**480**	2.9
C4000.5.clq	–	3989	**3985**	555.2
johnson32-2-4.clq	465	**465**	**465**	341.2
MANN_a81.clq	**3318**	3318	3319	19971.5
p_hat1500-2.clq	–	1473	**1439**	21
C1000.9.clq′	–	952	**940**	1.5
hamming10-2.clq′	–	**512**	**512**	0.7
keller6.clq′	–	3330	**3328**	107.7
MANN_a45.clq′	–	990	**705**	1.2
DSJC1000.1.col	–	945	**943**	3.8
flat1000_50_0.col	980	**980**	987	10.3
le450_25a.col	359	370	**360**	0.4
R250.1.col	180	190	**184**	0.1
zeroin.i.2.col	84	**84**	**84**	0.1

Here we show the known size of Exact cover (C) and the running time (t_{h3} in seconds) required to find the Cover size (C_{h3}) by our heuristics $h3$ compared with the cover size (C_{hp}) mentioned in their paper

$p < 0.05$ and *"the difference between the rankings of the methods* (hp *and* $h3$) *is significant"*.

Now we used a form of binomial test, known as the sign test (Salzberg 1997; Sheskin 2000), to find which method is significantly better. To do so, we counted the number of wins, losses and ties for $h3$ over hp (win $= 10$, loss $= 2$, tie $= 2$). The 10 times win of $h3$ over hp (in the $N = 12$ datasets, not considering the ties) supports with 95% confidence that "$h3$ *is significantly better than* hp".

6 New reduction rules for *k*-Vertex Cover?

We now return briefly to the augmented intuition approach for the development of reduction rules for vertex cover proposed in Sect. 4. The exploration of the graphs of size 9 remaining after reduction and their analysis using network alignment techniques to help identify common substructure now suggests the following two rules:

Lemma 4 *Let* (G, k) *be an instance of* k- Vertex Cover *with* $\alpha(G) = 2$. (G, k) *is a* Yes *instance iff* $k \geq |V(G)| - 2$.

Proof This result is a simple corollary of the classical k- Vertex Cover/Independent Set NP-completeness reduction. □

Lemma 5 *Let (G, k) be a reduced*[9] *instance of* k- VERTEX COVER *with a vertex v not in any independent set of size at least* 3*, then (G, k) is a* YES *instance iff $(G - v, k - 1)$ is a* YES *instance.*

Proof As $\alpha(G) \geq 3$, v cannot be in the complement of the minimum vertex cover, i.e., it must be in the minimum vertex cover. Therefore it is safe to include this vertex in any vertex cover. □

While these rules are only a small step for the field of fixed-parameter tractability, they offer a proof-of-concept for the augmented intuition approach for algorithm design in this context. What is particularly interesting is to consider the more advanced version whereby we employ computational approaches to not only hint at common substructure, but to propose reduction rules directly, and perhaps even prove them. While there are reduction rules that would evade this approach (certainly there are reduction rules which require at least first order logic to be expressed), many rules can be expressed in logics that are amenable to computer aided proof techniques.

7 Discussion and limitations of the study

The first limitation we would like to discuss is an interesting one since it has been self-imposed. We got an intuition of what may be needed to design a heuristic that would reduce to optimality every single graph of the *The 10-R*-non-reducible ones of sizes 8 and 9. This means that we could have empowered our three heuristics in the following way. Whenever the iterative application of the heuristic disconnects the graph by creating a subgraph of size 7, we could have resorted to the *The 10-R* set to get the exact solution for this component (since we know that any graph of size 7). In terms of our Berra "Lemma", we could have redesigned the three heuristics so that it will try to find a connected component of size 7 and safely reduce it via *The 10-R*. Therefore, the results of such a combined approach could have then been better than the one presented, but we abstained to include them in the heuristic because we wanted to evaluate the power of the new heuristics without resorting to the existing previous practice (the *The 10-R* set). Isolating our heuristic gave us some confidence on its merits by experimentally evaluating it in absence of the more sophisticated complex methods involved in the *The 10-R*.

Another reason for only presenting the results of the new heuristics by running them iteratively without the use of the wide set of reduction rules is due to the possible gains that can be obtained by an adequate permutation in the order they are applied. This situation is well illustrated in the study of Asgeirsson and Stein (2005) which also is based on the use of a triangle-approach (although highly different from us). Their claim is that: *"If vertices v, u and w form a triangle, then we can include all three vertices in a vertex cover for a* 3/2*-approximation"*. One entire section of their paper is just dedicated to discussing the relationship between this procedure to eliminate triangles and the order in which they are executed interspaced with other reductions. They say: *"After much experimentation, we settled on using the following order of*

[9] In particular with the crown rule, struction rule, the LP reduction and the rule implicit in Lemma 4.

reductions to automate the approximation process." (Sec. 3 of Asgeirsson and Stein (2005), p 552), thus recognizing that their results may be been the result of manual tuning with either a smaller set of instances (or with the same they are reporting results with). Without knowing the details of this experimentation, and because the use of the other reductions rules may be a confounding factor, we opted not to follow that approach. In addition, for some instances, and for some of the reduction rules they used, the choice of reductions may lead to a situation in which the algorithm gets "stuck" and can not finish (see Table 2 of Asgeirsson and Stein 2005). We decided to avoid this experimental confounder and just present the results of our heuristics in isolation from safe reductions rules. It is, however, an open area for research how to best intersperse the heuristics presented in this paper with other reduction rules existing in the literature [such as those in Akiba and Iwata (2016), Asgeirsson and Stein (2005), Stege (2000) and/or the extended network flow approach Akiba and Iwata (2016), Iwata et al. (2014)].

We also note that our approach is not undermining any of the current practices, but instead calls for a confluence of techniques in a common purpose. Benchmarking instances, for example, have been able to reveal the practical limitations of the *The 10-R* set (Table 1) in spite of its theoretical interest as a worst-case analysis. However, the relatively large size of these instances does not allow us to benefit from them. We argue that perhaps evolutionary algorithms can be used to find small yet hard to solve instances for particular algorithms such as in Cotta and Moscato (2003), Ahammed and Moscato (2011), or by construction using specific graph grammars (such as done for the TSP Mariano et al. 1995; Moscato and Norman 1998). Automation may also play a role here by generating instances via folding and mirroring, an approach that goes all the way back to pioneering work by David Hilbert (Norman and Moscato 1995). The automated co-evolution of the generation of the worst-case scenarios of the small size of instances for the existing safe-reduction reduction rules (and approximation algorithms) seems to guarantee future research due to its core role in augmenting our intuition. Again, we envision a man-machine approach as *"Thinking really hard"* will always have a role to generate difficult instances for benchmarking Hougardy and Zhong (2020).

8 Conclusions

It is obvious that the quest of presenting a compelling case of why an augmented intuition computational-based approach to algorithm and heuristic design is far from being completed. However, we have some interesting conclusions and also some lessons from this study that are worth sharing. We will summarize here some of the most relevant ones at this stage of our research endeavour.

First, as shown in Sect. 3.2, we tested the reduction rules from one of the best fixed-parameter algorithms for the k- VERTEX COVER as of 2006 [Abu-Khzam et al. (2004) and Chen et al. (2010)], with some of the tightest worst-case analysis. We thus scientifically tested Hooker (1995) both our implementations and the theory behind what we called '*The 10-R*', the ten safe reductions rules in Abu-Khzam et al. (2004) and Chen et al. (2010) (we refer to Sect. 3.1). By a comprehensive approach that tested all non-isomorphic graphs of size 8, we created an intuition of what may be the core problem to develop further fixed-parameter algorithms based on kernelization. We observed that only five graphs of medium density can not be reduced by *The 10-R*. Without automation, we conjecture that the task of identifying such a small subset of non-reducible instances in the large number of non-isomorphic graphs of size 8 (a ratio of $5/12346 \approx 0.000405$) could have not happened by human intuition alone or, alternatively, it would have possibly taken several decades until all five are finally identified. Computer-based scientific enquire clearly plays a big role here. In addition, "looking ahead", also supported by automation (Sect. 3.3), shows that the ratio of non-reducible graphs by the set *The 10-R* is just slightly higher ($118/274668 \approx 0.00043$). Here is then our first claim for an augmented intelligence approach to algorithm design.

Second, the identification of new properties of graphs may require, in turn, the identification of common structures in existing graphs which are conjectured to be of the same type. Again, this seems to be another call for innovative automation if we wish to augment the mathematical intuition of the human designer. In that sense our analysis presented in Sect. 3.4 is based on the evidence obtained by the inductive step of Sect. 3.3 (incrementing the size of the graphs by just one node). The use of algorithms for subgraph isomorphism and network alignment algorithms seems to be required; we were well positioned to do this by having a highly-effective memetic algorithm for this NP-complete and W[1]-complete problem (Mathieson et al. 2019). On the positive side for theory formation, only one of the five graphs of size eight (G8-01) is a subgraph of many *The 10-R*-non-reducible subgraphs of size 9 (114 out of 118), there are only four types of new "structures" of interest in *The 10-R*-non-reducible graphs of size 9. This may indicate that only one or two new reduction rules may be required to reduce all these 114 graphs.

Third, we obviously are well aware that the approach of systematically employing the generation of non-isomorphic graphs in the quest of obtaining non-reducible graphs can continue "ad infinitum" (or we should say, an arbitrarily large order). It also poses a known consideration for kernelization algorithms; a similar situation will likely occur in all FPT-algorithm design tasks for problems in class FPT. This is hardly being discussed but "hitting the wall" is likely to be the most common issue with the paradigm of kernelization for some instance size. We propose that an alternative scientific approach needs to be created. On the positive side, we showed that our study was feasible to do with our available university computing systems and we have obtained some new leads for further investigation. Notably, not only our work took us to the final line, we are leveraging on the expertise and work of others, for instance, Brendan McKay from ANU and his collection of non-isomorphic graphs available online. Then, our collection of non-reducible graphs gave us the intuition that the use of a truss decomposition could guide new types of heuristics, and we presented our tests in this manuscript. Following Hooker's guidelines (Hooker 1995), three heuristics were designed that optimally solved to optimality all *The 10-R*-non-reducible graphs of size 8 and 9; to give context to our heuristics we presented the results of tests with challenging instances used in the literature. Once again, the process we proposed has been backed by the use of automation to gain new intuition and the results also look very promising.

There is another intuitive insight we have obtained, which we would like to share, and that we leave for further exploration. With the exception of graph G08-01, all the other four *The 10-R*-non-reducible graphs of size 8 have a connected vertex cover (Escoffier et al. 2010), so the "price of connectivity" (Camby et al. 2014) is 1 for these graphs. This leads us to conjecture that some kind of extremal graph theory argument can be applied for graphs having some property that these graphs share (aside of being *The 10-R*-non-reducible), leading to perhaps to a new heuristic via the computation of a spanning tree subgraph that maximizes the number of leaves while minimizing the maximum degree observed in the leaf in the original graphs [e.g. a parameterized generalization of the problem studied in Binkele-Raible and Fernau (2014)]. If this new problem is also in class FPT, perhaps it can provide interesting new reduction rules applicable to k- Vertex Cover (if we can characterize those for which the prize of connectivity is equal to 1). Both from exact and heuristic perspectives, this is an interesting area that guarantees further research and we have obtained a new intuition thanks to the proposed method. We thus claim that the approach could also trigger new theory formation.

Before we conclude our observations we may also say that the use of graph layout algorithms (Sect. 4.1) provides another type of insights. It may be the case that one of the results of applying the *The 10-R* set of reduction rules is the creation of graphs which can be arranged in "layers" of nodes such that each layer contains an independent set, and nodes in one layer are connected to other nodes "only a few layers apart" (perhaps an emergent new parameter for parameterized complexity analysis). This seems to be an indirect result of the relatively large number of triangles present in these graphs but also about its global structure, which in turn motivated the introduction of our heuristics. The outputs of the *yEd* software shown in Figs. 5 and 6 show this characteristic. In addition, from Fig. 1 the reader can observe that for all these graphs the vertices in the minimal vertex cover can be located in the same layer that is connected to another layer which is composed entirely of vertices in an independent set. From this figure, as well as Fig. 4 we got the intuition that perhaps another reduction rule, similar in spirit to the crown reduction (see Appendix A), which is part of the *The 10-R* set but which may not been "covered" by the network flow-based procedure of Iwata et al. (2014). This is another area that would benefit from future theoretical research, particularly since the recent results on "layered graphs" (Chitturi 2017; Chitturi et al. 2018). If we can test in polynomial time (or if an FPT-algorithm can be developed that can test if the precondition is true), perhaps a dynamic programming approach, such as the one proposed in Chitturi et al. (2018), can be used in tandem thus leading to a combined set of reduction rules of great power. A tight complexity result would then be required to prove that all non-reducible graphs (of some set of reduction rules such as *The 10-R*) is guaranteed to produce a graph having the required precondition. Once again, this makes a bridge to theory formation by collecting evidence of required structure in small, yet hard to identify, instances of the problem.

Finally, to develop a research enterprise such as that requested almost a quarter of a century ago by Hooker (1995), and in *"Needed: An Empirical Science of Algorithms"* (Hooker 1994), we are convinced that extramural and international cooperation is going to be necessary. While we think there is still a role for benchmarking, particularly as a component of the cooperative design of challenges to the field, this "empirical science" would need to collect evidence by generating large amounts of well annotated datasets. It is "as if" algorithm design may soon be entering a field such as high-energy particle physics, with international collaboration in generating data, interpretations and new theory. We base our belief in the relatively complex set of tools we have used and developed for this research project which involved using integer programming solvers, develop state-of-the-art network alignment algorithms, use of libraries of annotated non-isomorphic graphs, employ visualization and layout algorithms, coding efficient implementations of kernelization algorithms, etc. An international collaborative effort in developing the set of tools needed for automating parts of algorithmic design across the globe would be essential for this global virtual lab of collaboration. We thus hope that our work will stir the interest both of theoreticians and practitioners and perhaps be catalytic for the confluence of a new international research alliance and a novel scientific paradigm for algorithm and heuristic design based on augmented intuition and through international collaboration.

Acknowledgements We are very thankful to Yoichi Iwata who kindly gave access to the code developed with Takuya Akiba for reference Akiba and Iwata (2016). The authors thank Michael Fellows to point out some safe reduction rules in Stege (2000) which are often overlooked. This also pointed at the need of an international cooperation for the annotation of complexity results and useful implementations in code. P.M. wants to dedicate his work to the memory of the late Jon Borwein (1951–2016), a world-class champion for an experimental approach in mathematics, with fond memories of the work they were planning to do together in this area.

A The 10 reduction rules used to test the kernelization performance on benchmark instances

In parameterized complexity, for a problem that is in class FPT, it is possible to obtain, under some circumstances and after "thinking really hard", an algorithm for which a tight upper bound on the complexity can be found.

This algorithm is obtained after the combined effort of the repeated application of the ten reduction rules for k- VERTEX COVER which concentrate on different properties of the graph. We have implemented all of them and we used the reduction rules one after another on the output from the previously applied reduction rule. We will apply these rules repeatedly until no more reduction possible. We will call the resulting method *The 10-R* and we will analyze their performance on some sets of instances.

The ten reduction rules of the *The 10-R* are:

- Degree Zero
- Degree One
- Degree k
- Degree Two Adjacent
- Degree Two Non-adjacent
- Complete neighborhood
- Crown
- Linear-Programming (LP)
- Struction
- General Fold

In this section we will describe those reduction rules for a graph $G = (V, E)$ where we have $n = |V|$ and $m = |E|$. It is then useful to explicitly define them here as follows:

Reduction Rule 1 **(Degree Zero)** *If G contains a vertex u such that $|N(u)| = 0$, then remove u.*

An isolated vertex is not incident on any edge. Hence, it can not be in a optimal vertex cover. We can eliminate a degree zero vertices form G and reduce the problem size n by one, but the parameter size remains at k.

Reduction Rule 2 *(***Degree One** *Balasubramanian et al. (1998)) If G contains a vertex u such that $N(u) = \{v\}$, then add v to the vertex cover and remove u and v from the graph.*

A vertex with single degree can be removed form the graph if and only if its unique neighbor is included in the vertex cover. Removing u from G and adding v to the vertex cover will reduce size of the instance n by two and parameter to $k - 1$.

Reduction Rule 3 *(***Degree** *k Buss and Goldsmith (1993)) If G contains a vertex u with $|N(u)| \geq k$, then add u to the vertex cover and remove it from the graph.*

Selecting u to the vertex cover of the graph reduces the problem size to $n - 1$ and the parameter size to $k - 1$.

Reduction Rule 4 *(***Degree Two Adjacent** *Balasubramanian et al. (1998)) If G contains a vertex u such that $N(u) = \{v, w\}$ and $(vw) \in E$, then add v and w to the vertex cover and remove u, v and w from the graph.*

At least two of three vertices u, v, w of a triangle should be selected to the vertex cover, and any optimal solution that does not choose v and w can be changed to one that does. It will reduce the problem size to $n - 3$ and the parameter size reduces to $k - 2$.

Reduction Rule 5 *(***Degree Two Non-adjacent** *Balasubramanian et al. (1998)) If G contains a vertex u such that $N(u) = \{v, w\}$ and $(vw) \notin E$, then we can fold u by contracting edges uv and uw. We can achieve this by replacing u, v, w with a new vertex u', where $N(u') = N(v) \cup N(w)$.*

Either u is chosen to cover the edges (uv) and (uw) and neither of v or w are chosen, or at least one of v or w is chosen, in which case it is sufficient to choose both and not u to be in the vertex cover Chen et al. (2001). If the new vertex u' is chosen to be in the vertex cover, this corresponds to the case where v and w are chosen, if not, u is chosen. It will reduce the problem size to $n - 2$ and the parameter size to $k - 1$.

Reduction Rule 6 *(***Complete neighborhood***) If G contains a vertex u such that $N(u)$ is a clique (i.e. for every $v, w \in N(u)$ we have $vw \in E$), add $N(u)$ to the vertex cover and remove u and $N(u)$ from the graph.*

This is an extension of the Degree Two Adjacent rule above. A complete graph requires all but one of its vertices to cover its edges, and as u is only adjacent to no other vertices, an optimal solution will never require it to cover anything. It will reduce the problem size to $n - |N(u)| - 1$ and the parameter size to $k - 1$.

Reduction Rule 7 *(***Crown Rule** *Abu-Khzam et al. (2004, 2007)) If G is a graph with a crown (I, H), then there is a vertex cover of G of minimum size that contains all the vertices in H and none of the vertices in I.*

A crown decomposition (I, H, B) of a graph G is a partition of V into sets I, B and H such that

- the crown I is a non-empty independent set,
- the head $H = N(I)$, and
- the rest of the graph body $B = V \setminus (I \cup H)$, and
- there is a matching of size $|H|$ in $G[H \cup I]$.

Hence, we can remove all vertices in I and H from the graph G. The problem size is reduced to $n - |I| - |H|$, and parameter to $k - |H|$ after adding H to the vertex cover.

Reduction Rule 8 *(***Linear-Programming** *Chen et al. (2001))* ***Theorem 1*** *If P, Q and R are defined as below, there is an optimal vertex cover that is a superset of P and that is disjoint from R.*

We can formulate the vertex cover problems as an Integer Linear-Programming in the following manner.

For each vertex $u \in V$ we assign a value $X_u \in \{0, 1\}$ such that the following conditions hold:

- Minimize $\sum_u X_u$
- Satisfy $X_u + X_v \geq 1$ whenever $uv \in E$

As solving integer linear programming is NP-Hard, we use Linear Programming (LP) to approximate the optimal solution for the problem. We can relax constraint $X_u \in \{0, 1\}$ to $X_u \in [0, 1]$, which can further simplified as $X_u \geq 0, \forall u \in V$. Hence, the LP formulation for Min Vertex Cover is:

- Minimize $\sum_u X_u$
- Satisfy $X_u + X_v \geq 1$ whenever $uv \in E$ and $X_u \geq 0$.

To simplify this LP problem, let assume

- $N(S)$ denote the neighborhood of S,
- $P = \{u \in V | X_u > 0.5\}$,
- $Q = \{u \in V | X_u = 0.5\}$ and
- $R = \{u \in V | X_u < 0.5\}$.

Finally, we remove P, R and their adjacent edges from the graph G. The problem size is reduced to $n - |P| - |R|$, and the parameter size becomes $k - |P|$ after adding P to the vertex cover.

Reduction Rule 9 (**Struction** *Chen et al. (2010)) Given a vertex v with neighborhood $N(v) = \{u_1, \ldots, u_p\}$ and with at most $p - 1$ non-edges between its neighbors. For every pair u_i, u_j of non-adjacent vertices in $N(v)$ add a new vertex u_iu_j with edges to every vertex in $N(u_i) \cup N(u_j) \setminus \{v\}$. Remove v and $N(v)$ from the graph.*

This rule is a generalization of the Degree Two Non-adjacent rule, and reduces the size of the graph to at most $n - 1$, and reduces the parameter to $k - 1$.

Reduction Rule 10 (**General Fold** *Chen et al. (2010)) Given an independent set I, and its neighborhood $N(I)$ where $|N(I)| = |I| + 1$, with the property that for every $\emptyset \subset S \subseteq I$ $|N(S)| > |S|$, either*

1. *$N(I)$ induces an independent set, we can remove I and $N(I)$, add I to the vertex cover, add a new vertex u and add the edge uv whenever v was a neighbor of some vertex $w \in N(I)$, and reduce k by $|I|$, or*
2. *$N(I)$ does not induce an independent set, and we may remove I and $N(I)$ from the graph, and add $N(I)$ to the vertex cover and reduce k by $N(I)$.*

B Current methods and practices of automated heuristics design

In relation to the toolkit of techniques to design metaheuristics, Crainic and Toulouse (2003) in Crainic and Toulouse (2003) presented a survey on parallel metaheuristic developments. It mainly focused on the parallel design and implementation principles of metaheuristics for larger problems to solve in reasonable computing times. It presented the parallel design principles for genetic algorithm, simulated annealing and tabu search. Birattari et al. (2006) reviewed about the analysis of problem of evaluating metaheuristics proposed in theory and used in practice Birattari et al. (2006). The experimental practice in machine learning for evaluating the performances of metaheuristics were mainly reported in this article. Zhang et al. (2017) edited a special issue of "Journal of Optimization" Zhang et al. (2017), focused on various applications of *algorithmic design* for metaheuristic optimization algorithms. We found following complex metaheuristics (mostly dependent on population-based search techniques) being applied on some classical and real-life problems:

- **Genetic Algorithm:** The *Genetic Algorithm* (GA) was used for solving the Minimum Dominating Set of Queens Problem and image processing problem to detect spots or disease on the plant.
- **Particle Swarm Optimizers:** Hybrid Particle Swarm Optimizers (PSO) was applied for software engineering optimization in MapReduce programming, identifying genetic signature for cancer classification and for solving (with help of memetic algorithm) a complex military problem.
- **Memetic Algorithm:** Memetic algorithm (MA) is popular in combinatorial optimization problems Berretta et al. (2003); França et al. (1999); Moscato et al. (2010); Naeni et al. (2014); Moscato (2012); Berretta et al. (2012) and Multi-objective variation of memetic algorithms were employed to solve an examination timetabling problems and a complex real-world military problem (in conjunction with hybrid PSO to solve the problem).

– **A* Search:** Sparse A* Search (SAS) was used for unmanned combat aerial vehicle (UCAV) path planning problem.

Sörensen et al. (2008) in Sörensen et al. (2018) presented a brief history of metaheuristics for five development periods. The authors highlighted the paradigm shift in development of heuristic methods from *method-centric* to *framework-centric* period and to further explore the succession into the *scientific period*. They expected to see more work in the research field of this scientific period to generate structure knowledge to benefit both the researchers and practitioners. Zufferey (2012) in Zufferey (2012) proposed a set of 17 rules to use for metaheuristics design. Among them, eight rules being proposed for designing the metaheuristic. They also proposed rules for designing local search and evolutionary methods for metaheuristics. The author illustrated each of the rules for three types of well-known optimization problems: graph coloring, vehicle routing and job-shop scheduling problems. Their analysis of the literature showed that the complex metaheuristic methods which combined the population search and efficient local search procedures were seemed to be the most promising. Nakib et al. (2017) in Nakib et al. (2017) proposed a complex framework to design metaheuristic using *machine learning* method evolved on the mutual information metric until the stopping criteria were meet. The maximum likelihood principle was used to implement the framework. The method was tested only on a set of functions in the literature of large scale continuous optimization.

More recent reviews by references Stützle and López-Ibáñez (2018, 2019); Hussain et al. (2019) summarised the latest advances in metaheuristic algorithms. Among them, Stützle and López-Ibáñez (2019) focused on the automatic design and configuration of metaheuristics algorithms and Hussain et al. (2019) presented a comprehensive survey of metaheuristics for 33 years (starting from 1983 to 2006) in Ref. Hussain et al. (2019). In their both publications Stützle and López-Ibáñez (2018, 2019), Stützle and López–Ibáñez mainly focused on the drawbacks of the manual, labor-intensive and intuition-based approach of algorithm design, and the recent advancement on automatic design and configuration of metaheuristic algorithms. However, in Hussain et al. (2019) the authors discussed the trends of metaheuristics by application area, types of metaheuristics and the theoretical and mathematical foundations of metaheuristic design.

All of these works of literature presented the methods and practices of automated heuristics design. We have found mainly two basic approaches of heuristic design and configuration. The first category is self-tuning and self-adapting heuristics driven by search techniques (local search and/or population-based method, *e.g.*, Ref. Zufferey (2012)). The other group learns from a set of training instances and then generalises to unseen instances (often denoted as the Machine Learning-based approach, *e.g.*, in Nakib et al. (2017)). There are some methods which combined both approaches (it is worth noting that Zufferey (2012) found the combined methods to be the most promising) to create complex heuristic.

In contrast, the more practically oriented field of Heuristics, and particularly the modern practice of developing metaheuristics employs a much more experimentally driven approach, including toolkits based around adversarially developed heuristics, machine learning and so forth. Unfortunately, the loop back to Algorithmics (i.e.

mathematical procedures that have theorem-proven guarantees) is not closed, and while many well tested and effective heuristics and metaheuristics that work well in practice have been produced Birattari et al. (2006); Crainic and Toulouse (2003); Nakib et al. (2017); Stützle and López-Ibáñez (2019); Zhang et al. (2017); Zufferey (2012), the advancement of the algorithmic understanding of the problem is at best slow.

C Current trends in metaheuristics for this problem

We offer here a brief survey of metaheuristics for the vertex cover problem where the results are presented for BHOSLIB and DIMACS benchmarking dataset. However, we note that most of the metaheuristics reported results only for a subset of the instances of the BHOSLIB and DIMACS benchmark datasets.

Guturu and Dantu in Guturu and Dantu (2008) presented an impatient EA with probabilistic tabu search (**IEA-PTS**) for the Min Vertex Cover (VC) problem. The proposed method works in two stages. First, the problem is mapped onto the maximum clique-finding problem (MCP), which is then solved using an evolutionary strategy. The EA learns not only form previously successful search directions but also from previous failures. The probabilistic tabu-search (PTS approach is used to discourage the search of earlier unfruitful directions. They have used 37 instances form the DIMACS benchmark set for VC problems. Along with the minimum cover size obtained by the IEA-PTS, they also reported the results for two other metaheuristics: a stochastic local search algorithm named *Cover Edges Randomly* (**COVER**) Richter et al. (2007) and *CycleKernelized Ant Colony System* (**CKACS**) Gilmour and Dras (2006). The COVER Richter et al. (2007) is a Stochastic Local Search (SLS) method for that uses an *edge weighting*-based heuristic. It starts from a uniformly random initial candidate solution which is iteratively improved by small step thus creating neighboring candidate solutions. The method increases the weights of yet uncovered edges at each step of the iteration. The CKACS is an Ant Colony System (ACS) metaheuristic which continually reinforces the kernelization information as the global pheromone update rule Gilmour and Dras (2006).

A multi-start metaheuristic called Greedy Randomized Adaptive Search Procedure for the variant of the core problem called Connected Vertex Cover (**GRASP-CVC**) has been proposed in Zhang et al. (2018). We review it here since any feasible solution of this method is also one of the unrestricted version and it can give an idea of the power of this method. Also, some of the heuristics proposed in this paper (in Sect. 4.6) can be useful as part of a GRASP strategy. **GRASP-CVC** used a greedy function and a restricted candidate list to construct high-quality initial solutions. The initial solution is then iteratively improved and a neighborhood of solutions is explored via a local search mechanism. This method was only tested on a small subset of DIMACS data instances. O. Ugurlu proposed an Isolation Algorithm (**IA**) Ugurlu (2012) based on isolating the vertex with a minimum degree and followed by the addition of the neighboring vertices of the isolated vertex into in the covering set. They applied the proposed algorithm on the BHOSLIB and DIMACS instances. In Cai et al. (2013), a metaheuristic for VC with two new strategies, called **NuMVC**, is proposed. First, a strategy for selecting two vertices (one vertex from the current candidate solution for *removal*, then uniformly at random selects another vertex from uncovered edges for *addition*) to exchange separately. The exchange of those vertices is performed in two stages: one at the *remove stage* and another in the *add stage*. Next, the strategy is to periodically increase and decrease the edge weighting. This algorithm has been applied to both the BHOSLIB and DIMACS instances.

C.1 More complex heuristics on BHOSLIB Dataset

We found several local search based more complex heuristics for the Min Vertex Cover problem which reported results on the BHOSLIB benchmark datasets. The vertex cover size and runtime reported by COVER Richter et al. (2007), CKACS Gilmour and Dras (2006), IEA-PTS Guturu and Dantu (2008), GRASP-CVC Zhang et al. (2018), and the times they took are surveyed here to give some context in comparison with constructive heuristics like IA Ugurlu (2012) and our $h3$. The results on the BHOSLIB instances are shown in Table 5. Note that, generally speaking, iterative improvement based schemes require orders of magnitude more CPU time.

C.2 Complex metaheuristics on a subset of DIMACS instances

The vertex cover results reported by three metaheuristics (GRASP-CVC, IA, NuMVC) on a subset of DIMACS instances are shown in Table 6.

C.3 Summary of results

From the result on BHOSLIB instances in Table 5, we can see that the COVER approach exhibited comparatively better performances in finding the optimal solution reported by Guturu and Dantu (2008). It has found the exact solution for 18 instances. However, in Richter et al. (2007) the COVER method was being applied on DIMACS instances and it did not perform well on the `brock` family of graphs where the algorithm failed to escape local minima Richter et al. (2007). In this regard, our heuristic $h3$ seemed to be an effective constructive method in finding high quality solutions form different types of graphs. They may give researchers new insights about how to improve their metaheuristics (e.g. the heuristic can be used as an initialization approach and to help recombination in evolutionary algorithms).

The compiled results for Min Vertex Cover by different metaheuristics on the subset of DIMACS instances are shown in Table 6. Here we can see that NuMVC (C_{nu}) exhibits the best performance. However, the runtimes have not been reported. Only the GRASP-CVC and IA methods reported their runtimes. The runtimes are presented here as a guideline, but they are not comparable, because these algorithms being executed in completely different computers.

Table 5 The cover size (C) and runtime (t in seconds) reported for BHOSLIB instaces by the five complex metaheuristics [we used subscripted short name of er for COVER (Richter et al. 2007), ck for CKACS (Gilmour and Dras 2006), ea for IEA-PTS (Guturu and Dantu 2008), as for GRASP-CVC (Zhang et al. 2018) and ia for IA (Ugurlu 2012)]

Instance	**C**	C_{er}	C_{ck}	C_{ea}	C_{as}	C_{ia}	t_{er}	t_{ck}	t_{ea}	t_{as}	t_{ia}
frb30-15-1	420	**420**	424	**420**	424	424	0.08	NR	0.409	11.3735	0.016
frb30-15-2	420	**420**	424.5	**420**	425	423	0.1	NR	0.4175	7.873	0.031
frb30-15-3	420	**420**	424.6	420.71	424	424	0.08	NR	1.4649	13.4266	0.016
frb30-15-4	420	**420**	424	**420**	424	422	0.05	NR	0.1464	15.2355	0.031
frb30-15-5	420	**420**	423.6	420.44	423	424	0.17	NR	2.218	25.6118	0.015
1b35-17-1	560	**560**	565.5	560.88	565	565	0.9	NR	3.4726	29.4486	0.016
frb35-17-2	560	**560**	566.5	560.81	565	565	0.84	NR	1.8495	8.5406	0.047
frb35-17-3	560	**560**	564.4	**560**	565	564	0.27	NR	0.7229	2.7005	0.063
frb35-17-4	560	**560**	565.5	561.05	565	564	1.12	NR	2.1753	172.9833	0.046
frb35-17-5	560	**560**	564.1	560.44	565	565	0.49	NR	3.9634	34.4835	0.031
frb40-19-1	720	**720**	725.6	720.18	728	725	0.62	NR	5.0149	187.7404	0.063
frb40-19-2	720	**720**	726.8	721.01	726	725	10.21	NR	2.3475	213.6616	0.078
frb40-19-3	720	**720**	727.6	720.92	725	724	3.17	NR	4.0084	101.0795	0.031
frb40-19-4	720	**720**	726.1	720.88	726	725	8.81	NR	3.5101	8.3877	0.078
frb40-19-5	720	**720.04**	725.3	721	725	723	63.47	NR	2.3618	33.3235	0.046

Table 5 continued

Instance	**C**	C_{er}	C_{ck}	C_{ea}	C_{as}	C_{ia}	t_{er}	t_{ck}	t_{ea}	t_{as}	t_{ia}
frb45-21-1	900	**900**	908.2	901.21	908	907	8.48	NR	10.1251	665.5343	0.062
frb45-21-2	900	**900**	908.5	901.13	908	904	28.46	NR	8.9796	83.904	0.063
frb45-21-3	900	**900.01**	908.3	901.17	908	906	70.13	NR	8.682	479.1447	0.124
frb45-21-4	900	**900**	908.4	901.01	907	905	12.28	NR	4.5054	135.2191	0.109
frb45-21-5	900	**900.01**	909.1	901.18	909	906	66.53	NR	8.0862	235.1495	0.062
frb50-23-1	1100	**1100.11**	1110.4	1101.63	1109	1108	171.92	NR	18.769	409.0972	0.14
frb50-23-2	1100	**1100.7**	1109.7	1101.3	1110	1108	543.56	NR	14.3682	87.4731	0.094
frb50-23-3	1100	**1100.76**	1108.3	1101.6	1110	1105	2.61	NR	16.9703	414.1894	0.109
frb50-23-4	1100	**1100**	1109.6	1101.24	1110	1108	16.94	NR	19.9206	431.3235	0.094
frb50-23-5	1100	**1100.02**	1110.3	1101.3	1109	1106	88.94	NR	14.3112	837.1014	0.094
frb53-24-1	1219	**1219.91**	1229.9	1221.44	1230	1226	11.31	NR	22.8648	695.5386	0.125
frb53-24-2	1219	**1219.66**	1229.3	1220.95	1230	1224	403.98	NR	24.6499	390.4401	0.141
frb53-24-3	1219	**1219.09**	1231.6	1221.13	1229	1226	157.8	NR	17.3548	552.5761	0.125
frb53-24-4	1219	**1219.76**	1230.5	1221.45	1229	1227	10.74	NR	23.9462	588.6573	0.109
frb53-24-5	1219	**1219.16**	1231.8	1221.42	1230	1227	253.05	NR	26.234	95.6667	0.109

Table 5 continued

Instance	**C**	C_{er}	C_{ck}	C_{ea}	C_{as}	C_{ia}	t_{er}	t_{ck}	t_{ea}	t_{as}	t_{ia}
frb56-25-1	1344	**1344.85**	1356.8	1346.76	1357	1352	20.73	NR	30.0986	90.5451	0.156
frb56-25-2	1344	**1344.88**	1355.7	1346.74	1353	1353	30.33	NR	36.0432	412.38	0.156
frb56-25-3	1344	**1344.24**	1355.6	1346.46	1356	1353	435.3	NR	35.8522	190.6417	0.156
frb56-25-4	1344	**1344.16**	1354.8	1346.78	1356	1353	291.11	NR	29.6232	47.1687	0.218
frb56-25-5	1344	**1344.02**	1354.6	1346.48	1355	1352	89.58	NR	29.9072	616.9286	0.265
frb59-26-1	1475	**1475.89**	1486.8	1478.06	1487	1481	30.76	NR	35.8522	656.1494	0.234
frb59-26-2	1475	**1475.94**	1486.4	1478.25	1488	1483	40.86	NR	32.1628	100.06	0.374
frb59-26-3	1475	**1475.88**	1487.8	1478.35	1489	1483	65.04	NR	32.2282	495.5281	0.359
frb59-26-4	1475	**1475.99**	1487.3	1478.26	1487	1484	75.92	NR	36.5014	319.4073	0.172
frb59-26-5	1475	**1475.11**	1486.3	1477.44	1487	1482	292.6	NR	30.79	662.7202	0.203

A value of ‘NR’ in a cell denote that *the method used that instance but did not report the value*

Table 6 The cover size (C) and runtime (t in seconds) reported in the papers for a subset of DIMACS instaces by the three complex metaheuristics

Instance	**C**	C_{as}	C_{nu}	C_{ia}	t_{as}	t_{nu}	t_{ia}
brock200_2	188	190	**188**	191	0.04	NR	0
brock200_4	183	184	**183**	–	1.4	NR	–
brock400_2	371	376	**371.16**	378	6.2	NR	0
brock400_4	367	376	**367**	–	10.29	NR	–
brock800_2	776	780	**779**	781	35.04	NR	0.438
brock800_4	774	780	**779**	–	174	NR	–
C125.9	91	**91**	**91**	**91**	<0.01	NR	0
C250.9	206	207	**206**	207	2.38	NR	0.094
C500.9	443	448	**443**	447	12.05	NR	0.016
C1000.9	932	939	**932**	941	66.33	NR	0.047
C2000.9	1920	1933	**1920**	–	96.85	NR	–
C2000.5	1984	1986	**1984**	1987	942.59	NR	0.219
C4000.5	3982	3986	**3982**	3985	400.23	NR	0.969
DSJC500.5	487	**487**	**487**	**487**	2.42	NR	0.282
DSJC1000.5	985	986	**985**	987	887.52	NR	0.062
gen200_p0.9_44	156	164	**156**	**156**	0.06	NR	0.109
gen200_p0.9_55	145	156	**145**	**145**	0.06	NR	0.015
gen400_p0.9_55	345	358	**345**	–	72.48	NR	–
gen400_p0.9_65	335	354	**335**	350	18.81	NR	0.015
gen400_p0.9_75	325	357	**325**	**325**	2.8	NR	0.64
hamming8-4	240	**240**	**240**	**240**	0.02	NR	0
hamming10-4	984	990	**984**	989	273.01	NR	0.078
keller4	160	**160**	**160**	**160**	3.93	NR	0
keller5	749	756	**749**	752	4.64	NR	0.453
keller6	3302	3324	**3302**	3316	82.16	NR	0.813
MANN_a27	252	260	**252**	375	0.05	NR	0
MANN_a45	690	704	**690**	1032	0.85	NR	0.047
MANN_a81	2221	2241	**2221.94**	3318	5.1	NR	0.453
p_hat300-1	292	**292**	**292**	**292**	0.11	NR	0.062
p_hat300-2	275	**275**	**275**	276	0.64	NR	0
p_hat300-3	264	**264**	**264**	266	0.34	NR	0.203
p_hat700-1	689	**689**	**689**	692	7.78	NR	0.031
p_hat700-2	656	**656**	**656**	657	1.79	NR	0.062
p_hat700-3	638	**638**	**638**	**638**	176.31	NR	0.063
p_hat1500-1	1488	1489	**1488**	1491	19.73	NR	0.125
p_hat1500-2	1435	1438	**1435**	**1435**	27.61	NR	0.438
p_hat1500-3	1406	1409	**1406**	1412	210.48	NR	0.36

The metaheuristics are GRASP-CVC (Zhang et al. 2018) (denoted as as), NuMVC (Cai et al. 2013) (denoted as nu) and IA (Ugurlu 2012) (ia) in the table. A dash mark '–' in a cell denotes that *the particular method did not use that instance* and 'NR' is used to denote that *the method used that instance, but had not reported the value*

Table 7 Number of times the Proposed Heuristic 3 wins, loses and had a tie with Heuristic IA (Ugurlu 2012)

Instance	H3.Wins	H3.Loses	H3.Ties
DIMACS	23	9	46
BHOSLIB	13	13	14
Asgeirsson and Stein (2005)	2	2	11
Network data	2	0	5

C.4 Performance comparison with Isolation Algorithm (IA) proposed in Ugurlu (2012)

O. Ugurlu proposed a simple heuristic, Isolation Algorithm (IA), in Ugurlu (2012), based on adding all the neighbors of the minimum degree vertex into the cover. They presented both the size of vertex cover and the runtime of the heuristic for BHOSLIB and a subset of DIMACS instances. However, we noticed that some of the exact covers reported in the paper do not match known values of the optimal cover. For instance, the reported optimum vertex cover size of 345 for `gen400_p0.9_65`, is problematic since it is known to be smaller (335). The optimum cover reported for MANN instances are also not matching with NuMVC in Cai et al. (2013). NuMVC reported that 252 and 690 are optimum values for `MANN_a27` and `MANN_a45`, respectively; IA reported them as 375 and 1032. They also reported the best-known cover size of `MANN_a81` as 2221; however, IA reported it as 3318. This cover size is more than 1000 vertex larger than the known size. These large differences in the known cover sizes used in the paper presents some problems to directly compare with the published values.

To address this problem, we have implemented the IA algorithm following Ugurlu (2012). However, from their description, it is also unclear how ties are meant to be broken when more than one vertex have the minimum degree. In our implementation of the heuristic IA, we took a vertex uniformly at random from those with the minimum degree. We then executed both our Heuristic 3 and the IA method on BHOSLIB, DIMACS, the small subset of DIMACS Datasets used by Asgeirsson and Stein (2005) and also on seven instances form Network Data.[10] Here we report the summary of the comparison in terms of the number of times H3 wins, loses and tie with IA in Table 7.

In the work of Cai et al. (2013), the `brock` graphs are referred as one of the hardest types of instances for vertex cover in DIMACS database. Among the 12 `brock` instances, IA ties with $h3$ in 7 cases and $h3$ wins in the remaining 5 cases. The most difficult instances from DIMACS are `C2000.5`, `MANN_a81`, `keller6` and `MANN_a45` Cai et al. (2013); Grosso et al. (2008); Richter et al. (2007). While we compare the vertex cover sizes found by IA and $h3$ for these instances, the IA has a tie with $h3$ for both of the `MANN` instances, wins on `C2000.5`, and loses on `keller6`. These clearly indicate that $h3$ is competitive with IA.

[10] Seven instances (adjnoun, as-22july06, celegansneural, dolphins, football, karate and lesmis) are taken from Network Data at http://www-personal.umich.edu/~mejn/netdata/.

Finally, to find if $h3$ is statistically better than IA, we conducted a one-tailed Wilcoxon signed ranked test.[11] Among the combined 140 instances from four sources (in Table 7). We found 76 cases of the tie, which yields the number of samples to $N = 64$. For those samples with a confidence level $\alpha = 0.05$, the one-tailed Wilcoxon Signed-Rank test revealed the W-value=788.5. As the distribution is approximately normal, the z-value is used. The value of z is -1.6819 with the p-value = 0.04648. Hence, the result is significant at $p < 0.05$ and "*$h3$ is significantly better or equal than IA*".

References

Abu-Khzam, F.N., Collins, R.L., Fellows, M.R., Langston, M.A., Suters, W.H., Symons, C.T.: Kernelization algorithms for the vertex cover problem: Theory and experiments. In: Arge, L., Italiano, G.F., Sedgewick, R. (eds.) Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, USA, 10 January, 2004, pp. 62–69. SIAM (2004)

Abu-Khzam, F.N., Fellows, M.R., Langston, M.A., Suters, W.H.: Crown structures for vertex cover kernelization. Theory Comput. Syst. **41**(3), 411–430 (2007)

Ahammed, F., Moscato, P.: Evolving L-systems as an intelligent design approach to find classes of difficult-to-solve traveling salesman problem instances. In: Applications of Evolutionary Computation—EvoApplications 2011: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Torino, Italy, 27–29 April 2011, Proceedings, Part I, pp. 1–11 (2011)

Akiba, T., Iwata, Y.: Branch-and-reduce exponential/FPT algorithms in practice: a case study of vertex cover. Theor. Comput. Sci. **609**, 211–225 (2016)

Asgeirsson, E., Stein, C.: Vertex cover approximations: experiments and observations. In: Nikoletseas, S.E. (ed.) Experimental and Efficient Algorithms, pp. 545–557. Springer, Berlin (2005)

Balasubramanian, R., Fellows, M.R., Raman, V.: An improved fixed-parameter algorithm for vertex cover. Inf. Process. Lett. **65**(3), 163–168 (1998)

Berretta, R., Cotta, C., Moscato, P.: Enhancing the performance of memetic algorithms by using a matching-based recombination algorithm. In: Resende, M.G.C., de Sousa, J.P. (eds.) Metaheuristics: Computer Decision-Making, pp. 65–90. Springer, Boston (2003)

Berretta, R., Cotta, C., Moscato, P.: Memetic algorithms in bioinformatics. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, Studies in Computational Intelligence, vol. 379, pp. 261–271. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-23247-3_16

Binkele-Raible, D., Fernau, H.: A parameterized measure-and-conqueranalysis for finding a k-leaf spanning treein an undirected graph. Discrete Math. Theor. Comput. Sci. **16**(1) (2014)

Birattari, M., Zlochin, M., Dorigo, M.: Towards a theory of practice in metaheuristics design: a machine learning perspective. RAIRO Theor. Inform. Appl. **40**(2), 353–369 (2006). https://doi.org/10.1051/ita:2006009

Buss, J.F., Goldsmith, J.: Nondeterminism within P^*. SIAM J. Comput. **22**(3), 560–572 (1993). https://doi.org/10.1137/0222038

Cai, S., Su, K., Luo, C., Sattar, A.: NuMVC: an efficient local search algorithm for minimum vertex cover. J. Artif. Int. Res. **46**(1), 687–716 (2013)

Camby, E., Cardinal, J., Fiorini, S., Schaudt, O.: The price of connectivity for vertex cover. Discrete Math. Theor. Comput. Sci. **16**(1), 207–224 (2014)

Chen, J., Kanj, I.A., Jia, W.: Vertex cover: further observations and further improvements. J. Algorithms **41**(2), 280–301 (2001)

Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theor. Comput. Sci. **411**(40), 3736–3756 (2010). https://doi.org/10.1016/j.tcs.2010.06.026

Chitturi, B.: Layered graphs: a class that admits polynomial time solutions for some hard problems. CoRR arXiv:abs/1705.06425 (2017)

[11] https://www.socscistatistics.com/tests/signedranks.

Chitturi, B., Balachander, S., Satheesh, S., Puthiyoppil, K.: Layered graphs: applications and algorithms. Algorithms **11**, 7 (2018). https://doi.org/10.3390/a11070093

Cotta, C., Moscato, P.: A mixed evolutionary-statistical analysis of an algorithm's complexity. Appl. Math. Lett. **16**(1), 41–47 (2003). https://doi.org/10.1016/S0893-9659(02)00142-8

Crainic, T.G., Toulouse, M.: Parallel Strategies for Meta-Heuristics, pp. 475–513. Springer, Boston (2003). https://doi.org/10.1007/0-306-48056-5_17

Demsar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)

Dinur, I., Safra, S.: The importance of being biased. In: Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing, STOC '02, pp. 33–42. ACM, New York (2002). https://doi.org/10.1145/509907.509915

Escoffier, B., Gourvès, L., Monnot, J.: Complexity and approximation results for the connected vertex cover problem in graphs and hypergraphs. J. Discrete Algorithms **8**(1), 36–49 (2010). https://doi.org/10.1016/j.jda.2009.01.005

Feige, U.: Vertex cover is hardest to approximate on regular graphs. Technical Report MCS03-15, Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel (2003)

Fernau, H., Fluschnik, T., Hermelin, D., Krebs, A., Molter, H., Niedermeier, R.: Diminishable parameterized problems and strict polynomial kernelization. In: Manea, F., Miller, R.G., Nowotka, D. (eds.) Sailing Routes in the World of Computation, pp. 161–171. Springer International Publishing, Cham (2018)

França, P., Mendes, A., Moscato, P.: Memetic algorithms to minimize tardiness on a single machine with sequence-dependent setup times. In: Proceedings of the 5th International Conference of the Decision Sciences Institute, Athens, Greece, pp. 1708–1710 (1999)

Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)

Gell-Mann, M.: The Quark and the Jaguar: Adventures in the Simple and the Complex. W. H. Freeman & Co., New York (1995)

Gilmour, S., Dras, M.: Kernelization as heuristic structure for the vertex cover problem. In: Dorigo, M., Gambardella, L.M., Birattari, M., Martinoli, A., Poli, R., Stützle, T. (eds.) Ant Colony Optimization and Swarm Intelligence, pp. 452–459. Springer, Berlin (2006)

Grosso, A., Locatelli, M., Pullan, W.: Simple ingredients leading to very efficient heuristics for the maximum clique problem. J. Heuristics **14**(6), 587–612 (2008)

Guturu, P., Dantu, R.: An impatient evolutionary algorithm with probabilistic Tabu search for unified solution of some NP-hard problems in graph and set theory via clique finding. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **38**(3), 645–666 (2008). https://doi.org/10.1109/TSMCB.2008.915645

Hochbaum, D.S.: Efficient bounds for the stable set, vertex cover and set packing problems. Discrete Appl. Math. **6**(3), 243–254 (1983). https://doi.org/10.1016/0166-218X(83)90080-X

Hooker, J.N.: Needed: an empirical science of algorithms. Oper. Res. **42**(2), 201–212 (1994). https://doi.org/10.1287/opre.42.2.201

Hooker, J.N.: Testing heuristics: we have it all wrong. J. Heuristics **1**(1), 33–42 (1995). https://doi.org/10.1007/BF02430364

Hougardy, S., Zhong, X.: Hard to solve instances of the euclidean traveling salesman problem. Math. Program. Comput. 1–24 (2020)

Hussain, K., Mohd Salleh, M.N., Cheng, S., Shi, Y.: Metaheuristic research: a comprehensive survey. Artif. Intell. Rev. **52**(4), 2191–2233 (2019). https://doi.org/10.1007/s10462-017-9605-z

Iwata, Y., Oka, K., Yoshida, Y.: Linear-time FPT algorithms via network flow. In: Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'14, pp. 1749–1761. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2014). URL http://dl.acm.org/citation.cfm?id=2634074.2634201

Karakostas, G.: A better approximation ratio for the vertex cover problem. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) Automata, Languages and Programming, pp. 1043–1050. Springer, Berlin (2005)

Mariano, A., Moscato, P., Norman, M.G.: Using L-systems to generate arbitrarily large instances of the euclidean traveling salesman problem with known optimal tours. In: In Anales del XXVII Simposio Brasileiro de Pesquisa Operacional, pp. 6–8 (1995)

Mathieson, L., de Vries, N.J., Moscato, P.: Using network alignment to identify conserved consumer behaviour modelling constructs, pp. 513–541. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-06222-4_12

Moscato, P.: Memetic algorithms: the untold story. In: Neri, F., Cotta, C., Moscato, P. (eds.) Handbook of Memetic Algorithms, Studies in Computational Intelligence, vol. 379, pp. 275–309. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-23247-3_17

Moscato, P.: Business Network Analytics: From Graphs to Supernetworks, pp. 307–400. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-06222-4_7

Moscato, P., Berretta, R., Cotta, C.: Memetic Algorithms. Wiley Encyclopedia of Operations Research and Management Science (2010)

Moscato, P., Mendes, A., Berretta, R.: Benchmarking a memetic algorithm for ordering microarray data. Biosystems **88**(1), 56–75 (2007). https://doi.org/10.1016/j.biosystems.2006.04.005

Moscato, P., Norman, M.G.: On the performance of heuristics on finite and infinite fractal instances of the euclidean traveling salesman problem. INFORMS J. Comput. **10**(2), 121–132 (1998). https://doi.org/10.1287/ijoc.10.2.121

Naeni, L.M., de Vries, N.J., Reis, R., Arefin, A.S., Berretta, R., Moscato, P.: Identifying communities of trust and confidence in the charity and not-for-profit sector: A memetic algorithm approach. In: 2014 IEEE Fourth International Conference on Big Data and Cloud Computing, BDCloud 2014, Sydney, Australia, 3–5 December 2014, pp. 500–507. IEEE Computer Society (2014). https://doi.org/10.1109/BDCloud.2014.83

Nakib, A., Hilia, M., Heliodore, F., Talbi, E.G.: Design of metaheuristic based on machine learning: a unified approach. In: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 510–518. IEEE (2017)

Norman, M.G., Moscato, P.: The euclidean traveling salesman problem and a space-filling curve. Chaos Solitons Fractals **6**, 389–397 (1995). https://doi.org/10.1016/0960-0779(95)80046-J

Richter, S., Helmert, M., Gretton, C.: A stochastic local search approach to vertex cover. In: Hertzberg, J., Beetz, M., Englert, R. (eds.) KI 2007: Advances in Artificial Intelligence, pp. 412–426. Springer, Berlin (2007)

Salzberg, S.L.: On comparing classifiers: pitfalls to avoid and a recommended approach. Data Min. Knowl. Discov. **1**(3), 317–328 (1997)

Sheskin, D.J.: Parametric and Nonparametric Statistical Procedures. Chapman & Hall/CRC, Boca Raton (2000)

Skiena, S.S.: The Algorithm Design Manual. Springer, London (2008)

Sörensen, K., Sevaux, M., Glover, F.: A History of Metaheuristics, pp. 1–18. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-07153-4_4-1

Stege, U.: Resolving Conflicts in Problems from Computational Biology. Ph.D. thesis, ETH Zurich (2000)

Stützle, T., López-Ibáñez, M.: Automated design of metaheuristic algorithms. Technical Report TR/IRIDIA/2018-008, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2018)

Stützle, T., López-Ibáñez, M.: Automated Design of Metaheuristic Algorithms, pp. 541–579. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-319-91086-4_17

Thorup, M.: All structured programs have small tree-width and good register allocation. Inf. Comput. **142**(2), 159–181 (1998). https://doi.org/10.1006/inco.1997.2697

Ugurlu, O.: New heuristic algorithm for unweighted minimum vertex cover. In: 2012 IV International Conference "Problems of Cybernetics and Informatics" (PCI), pp. 1–4 (2012). https://doi.org/10.1109/ICPCI.2012.6486444

Wang, J., Cheng, J.: Truss decomposition in massive networks. PVLDB **5**(9), 812–823 (2012). https://doi.org/10.14778/2311906.2311909

Wilcoxon, F.: Individual Comparisons by Ranking Methods, pp. 196–202. Springer, New York, NY (1992). https://doi.org/10.1007/978-1-4612-4380-9_16

Zhang, G., Pan, L., Neri, F., Gong, M., Leporati, A.: Metaheuristic optimization: Algorithmic design and applications. J. Optim. **2017**, (2017)

Zhang, Y., Wu, J., Zhang, L., Zhao, P., Zhou, J., Yin, M.: An efficient heuristic algorithm for solving connected vertex cover problem. Math. Probl. Eng. **2018**, (2018)

Zufferey, N.: Metaheuristics: some principles for an efficient design. Comput. Technol. Appl. **3**(6), 446-462 (2012)

Affiliations

Pablo Moscato[1] · **Luke Mathieson**[2] · **Mohammad Nazmul Haque**[1]

✉ Pablo Moscato
Pablo.Moscato@newcastle.edu.au

Luke Mathieson
luke.mathieson@uts.edu.au

Mohammad Nazmul Haque
Mohammad.Haque@newcastle.edu.au

1 College of Engineering, Science and Environment, The University of Newcastle, Callaghan, NSW 2308, Australia

2 School of Software, University of Technology Sydney, Ultimo, NSW 2007, Australia