

Circular Chromatic Index of Small Snarks

Dušan Bernát¹, Ján Mazák^{1,*}

¹FMFI UK, Bratislava, Slovakia

Abstract

This paper verifies and extends previous computational results on circular chromatic index of small snarks, gives further support for certain conjectures, and discusses various practical aspects of using SAT solvers to solve graph colouring problems (e.g. the impact of solver choice and Boolean formula construction on running time).

Keywords

cubic graph, snark, circular chromatic index, SAT solver

1. Introduction

The main aim of this article is to popularize the usage of SAT solvers for computations in combinatorics. For more than a decade, SAT solvers have been among the most efficient tools for problems in discrete mathematics, and there are plenty of them freely available. However, many researchers in the area, both young and more experienced, are still not sufficiently aware of their advantages and the simplicity of their use. We will illustrate the usage of SAT solvers through computations concerned with edge-colourings of graphs.

Cubic graphs that do not admit a 3-edge-colouring (so-called *uncolourable*) have been studied for more than a century, initially in connection with the Four Colour Problem, and more recently in the context of flows and cycle covers. In this article, a *snark* is a bridgeless simple cubic graph with chromatic index 4. Since the problem of determining the chromatic index of a cubic graph is NP-complete, we cannot expect a simple characterization of snarks, and volumes have been written on various attempts to gain a better understanding of their structure [1].

One possible approach is to acknowledge that “not all snarks are of the same difficulty” and introduce a measure that somehow splits snarks into classes with “increasing levels of uncolourability” [2]. For instance, if we consider the minimum number of odd cycles in a 2-factor of a cubic graph (its *oddness*), there are cubic graphs with oddness $2k$ for every non-negative integer k . For $k = 0$, we get colourable graphs, and presumably as oddness increases, the difficulty of proving theorems for such snarks increases: e.g. the 5-flow conjecture was gradually proved for snarks with oddness 0, 2, and 4 [3].

ITAT'24: Information technologies – Applications and Theory, September 20–24, 2024, Drienica, Slovakia


*Corresponding author.

✉ dusan.bernat@fmph.uniba.sk (D. Bernát);

jan.mazak@fmph.uniba.sk (J. Mazák)

ORCID [0000-0002-6498-9984](https://orcid.org/0000-0002-6498-9984) (J. Mazák)

© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

This paper focuses on circular chromatic index χ'_c , which is a refinement of chromatic index that allows real values as colours instead of just integers. For $r \geq 1$, a *circular r -edge-colouring* of a graph G is a mapping $c : E(G) \rightarrow [0, r)$ such that $1 \leq |c(e) - c(f)| \leq r - 1$ for any two incident edges e and f of G . If a graph G has a circular r -edge-colouring, we say that it is *circularly r -edge-colourable*. The *circular chromatic index* χ'_c of G is the infimum of all r such that G has a circular r -edge-colouring. This infimum is in fact a minimum; for a finite graph it is always attained. Moreover, it is rational, and the only possible candidates for $\chi'_c(G)$ are fractions p/q such that $p \leq |E(G)|$. Circular colourings can be used for optimization in certain types of scheduling problems, but we are not exploring this in this paper. For a detailed introduction to circular colourings, we refer the reader to the survey [4].

Colourable cubic graphs have $\chi'_c = 3$, while snarks have $\chi'_c > 3$, so χ'_c is also a “measure of uncolourability”, perhaps the most natural one. Snarks with χ'_c close to 3 are plentiful, whereas snarks with χ'_c above $10/3$ seem rare. This is partly known, partly hypothesized. An intuition behind this statement is as follows: In a 3-edge-colouring, when you fix the colours of two edges at a vertex, the colour of the third edge is uniquely determined—there is no wiggle room. But in a circular $(3 + \varepsilon)$ -edge-colouring, there is typically a small interval of length up to ε for allowed values of the third colour. With a bit of flexibility at each vertex, over a large enough subgraph, we can gradually shift the colours into pretty much anything we want on the edges leaving the subgraph, making it easy to get a circular $(3 + \varepsilon)$ -edge-colouring of the whole large graph even for a small value of ε . Of course, there are situations where this does not apply—for instance, if we have a small subgraph that cannot be coloured with $3 + \varepsilon$ colours for a small ε , it will ensure a lower bound on χ'_c for the whole graph.

The following theorems and conjectures summarize the current state of knowledge about χ'_c of cubic graphs.

Theorem 1 ([5]). *There is no graph with $\chi'_c \in (11/3, 4)$.*

Theorem 2 ([6, 7]). *The Petersen graph has $\chi'_c = 11/3$. There exists an infinite class of snarks with connectivity 3 and $\chi'_c = 7/2$.*

Theorem 3 ([8]). *For every $\varepsilon > 0$, there exists an integer g such that every snark with girth at least g has $\chi'_c \leq 3 + \varepsilon$.*

Theorem 4 ([6]). *The circular chromatic index of every snark with girth at least 6 is at most $7/2$.*

Theorem 5 ([9]). *For every rational $r \in (3, 10/3) \cup \{3 + p/(3p - 1) \mid p \in \mathbb{Z}^+\}$, there exists a snark with $\chi'_c = r$.*

Conjecture 1 ([6, 7]). *The circular chromatic index of every snark G except the Petersen graph is at most $7/2$. If G is cyclically 4-connected, $\chi'_c(G) < 7/2$.*

It is still open if the Petersen graph is the only graph with $\chi'_c = 11/3$. Based on our computational results, we propose two additional conjectures and one problem.

Conjecture 2. *If G is a snark with girth at least 6, then $\chi'_c(G) \leq 10/3$.*

The evidence for Conjecture 2 is somewhat scarce: it holds for Isaacs snarks with girth 6 (though not for those of smaller girth), and we verified it for all 42 snarks with girth 6 on at most 38 vertices and an incomplete list of snarks on 40 to 44 vertices (all of them being the result of a dot product applied to smaller snarks). Weak indirect support is also provided by Theorems 3 and 4.

Conjecture 3. *All cyclically 4-connected snarks G have $\chi'_c(G) \leq 17/5$.*

Problem 4. *Characterise cyclically 4-connected snarks G with $\chi'_c(G) > 10/3$. Is the set $M_{>10/3}$ of such snarks finite? Is the set of the values of χ'_c of such snarks finite?*

We have found 17 members of $M_{>10/3}$, described in more detail in Section 3.

2. Computing χ'_c

In light of the conjectures in Section 1, our computations are primarily focused on proving that there are no snarks with χ'_c bigger than some value r . This is the easier thing to do—one only needs to find a suitable circular r -edge-colouring. On the other hand, determining χ'_c requires proving a lower bound, which takes much more time. Anyway, where computationally feasible, we fully determine χ'_c , not just verify the conjectures.

For a given input graph and a fraction p/q , we construct a boolean formula that is satisfiable if and only if G is circularly p/q -edge-colourable. This formula uses a variable $x_{e,c}$ for each element $(e, c) \in E(G) \times \{0, 1, \dots, p - 1\}$; it states that each edge gets exactly one

colour and that no two incident edges can be assigned colours with difference less than q (using clauses like $\neg x_{e,0} \vee \neg x_{e,1}$). Then a SAT solver is used to decide the satisfiability of the problem instance. As explained in Section 1, there are finitely many candidates for fractions p/q and some bounds can be set based on the size of the input graph, so it is possible to determine χ'_c by subsequently solving several SAT instances. This type of problem is often solved by a binary search (we order the possible fractions from the lowest to the highest, look into the middle etc.), but it would be suboptimal in this case, as we explain in the next paragraph.

The time required to solve a particular instance for a given graph highly depends on p : a large value of p results in a large number of variables, which is the biggest determinant of time complexity (Section 4.2 gives more details on this). Our experiments clearly indicate that we need to reduce the number of instances with large values of p as much as possible during a search for χ'_c . There are two other factors at play, but both play a minor role. The first is that the time increases when the tested fraction is near χ'_c —when the value is far below χ'_c , the solver can quickly recognise unsatisfiability; when the fraction is too far above χ'_c , it is easy to find a colouring. The second is specific to circular colourings: fractions of the form $3 + 1/k$ are easier to exclude than $3 + 2/k$, $3 + 3/k$ etc. because there is less flexibility in colouring edges incident to a vertex. Based on these observations, we use a heuristic that always starts testing for the fraction with the smallest numerator.

3. Results of computations

The exhaustive lists of snarks with given order and cyclic connectivity we used come from [10]. The counts of cyclically 4-connected snarks with girth 5 up to 30 vertices with particular χ'_c were presented in [11]. The results of this diploma thesis were not published properly and perhaps are not widely known. We have fully confirmed the results of [11] and extended them for snarks of girth 4, see Table 1 (rows are ordered according to the increasing value of χ'_c).

All the results presented in this section were obtained using the *CaDiCaL_vivinst*, one of the winning solvers of the 2023 SAT Competition [12]. We used a machine with 20 cores (Intel Xeon Gold 5220R CPU @ 2.20GHz). Because the computations for different graphs are completely independent, the jobs within a batch are well suited for parallelisation, yielding the speedup approaching the number of CPU cores. The most demanding batch comprised all snarks of order 30 and girth greater or equal to 4; it took two weeks to complete. We verified the computed indices by using SBVA combined with Kissat [12], but so far only with the same formula-generation algo-

Table 1
Counts of snarks with girth at least 4 and specific χ'_c .

Order	10	18	20	22	24	26	28	30
χ'_c								
16/5	-	-	-	-	-	-	-	3
29/9	-	-	-	-	-	1	-	13
13/4	-	-	-	-	-	17	849	24070
36/11	-	-	-	-	-	-	-	1
23/7	-	-	-	-	19	396	5438	61838
33/10	-	-	-	-	-	-	-	2
10/3	-	1	5	29	136	883	6229	53925
17/5	-	-	1	2	-	-	1	2
7/2	-	1						
11/3	1							

rithm; we are working on additional verification and extension of the results to snarks on 32 or more vertices. On the other hand, our code is completely independent of what Kunertová used, and the results also agree with theoretically determined indices for graphs that we checked (e.g. Isaacs and generalized Blanuša snarks), which gives us confidence in the results.

Conjecture 3 suggests that non-trivial snarks with $\chi'_c > 10/3$ are quite exceptional. This is supported also by Table 1 which is sorted by index value. The members of the set $M_{>10/3}$ from Section 1 are below a line: the Petersen graph, the type 2 Blanuša snark, the flower snark on 20 vertices and five other snarks on 22, 28 and 30 vertices. Additional computations for all cyclically 4-connected snarks on up to 36 vertices (girth at least 4) identified one more snark of order 34 and two snarks of order 36 with $\chi'_c = 17/5$. In the incomplete set of generated snarks of order 38, there were six additional snarks (three of them have $\chi'_c = 27/8$, three of them have $\chi'_c = 37/11$). All these snarks are available at <https://github.com/janmazak/research-data> (see `circular_chromatic_index`).

4. SAT solvers for graph theory

4.1. Construction of Boolean formulas

We experimented with different ways of constructing Boolean formulas capturing 3-edge-colouring of cubic graphs. For instance, one can say that for every colour c and every two incident edges, at least one of the edges does not have colour c . Or that every colour is used on the three edges incident to a vertex at least once; or something based on combinatorial nullstellensatz etc. The running times were very similar, somewhat depended on the input graphs, and our experiences show that arcane ways of formula construction are likely not worth it for smaller-scale computations.

We suggest starting with a formula minimizing the chance of a mistake (either conceptual or in implementation—we made plenty of both kinds, so do not underestimate it, and if you can, make sure you verify your implementation against some independently obtained data). Of course, it pays off to employ known tricks allowing translation of disjunctive normal forms into small equisatisfiable conjunctive normal forms¹, thus avoiding the exponential blowup in the number of clauses.

4.2. Number of variables and clauses

Internally, almost all current leading SAT solvers use the CDCL algorithm. They thus learn new clauses along the way and operate with many clauses that were not part of the original input. Consequently, including some seemingly redundant clauses in the input might even help (though it usually does not). Also, solvers use preprocessing which significantly rewrites the given formula: while colouring problems typically yield clauses of size 2 and 3, it seems solvers prefer bigger clauses of size about 10 (at least in the couple of situations we were able to check). Thanks to technical tricks like watched literals, the clauses are not manipulated over and over, but are accessed only when relevant. The takeaway here is that the number of clauses is of very little practical concern (unless it is really extreme).

On the other hand, the number of variables is critical because it affects the depth (and thus the number of nodes) of the CDCL search tree. For SAT instances derived from graphs, the number of variables is typically some polynomial dependent on the size m of the graph, say, m^k . The resulting theoretical worst-case complexity of roughly 1.3^{m^k} seems terrifying, but in practice, SAT solvers do much better. We will illustrate this with our experience with several problems on cubic graphs.

- For 3-edge-colouring with integer colours, $k = 1$. Solvers easily beat other approaches² and can deal with graphs with hundreds or even thousands of vertices.
- For circular edge-colouring, the number of colours linearly depends on the size of the graph, and we need a variable for each pair [edge, colour], so $k = 2$ at worst. The number of clauses is cubic. Solvers work decently for small graphs, but somewhere around 60 edges the computation time significantly increases (from minutes to days per single graph). However, solvers are

¹For instance, the Tseytin transformation, or the approach mentioned in https://en.wikipedia.org/wiki/Conjunctive_normal_form#Other_approaches.

²Except tiny graphs and possibly except a recent algorithm based on path-width decomposition [13].

Table 2

Comparison of SAT solvers. Consumed CPU time in seconds for χ'_c of all snarks of order 24 and girth greater or equal to 4.

Solver	lingeling (2017)	MergeSat (two threads*)	Kissat 3.1.0	Kissat hywalk-exp	Kissat MAB_DeepWalk+	CaDiCaL vivinst	SeqFROST
T[s]	3024	7431	2575	1906	1880	2096	958
T/T'[%]	144	355*	123	91	90	100	46
S ₂₀	16	18	14	17	16	15	15

much faster than our backtracking algorithm³ at that threshold.

- For Hamiltonian cycles, the best formula construction we are aware of uses variables for pairs [vertex, order of vertex along the cycle], i.e. $k = 2$. The number of clauses is also cubic. Again, there is a threshold around 60–75 vertices after which computation takes days for a single graph. In contrast to circular colouring, backtracking competes with solvers fairly well.

In summary, SAT solvers are likely to be all you would ever need if you can keep the number of variables linear. If quadratic, they are helpful, but large graphs (somewhere above 50–80 edges) are out of their reach, and you are likely to run into trouble if you want to do computations for sizeable complete lists of graphs of a given order.

4.3. Which solver should I use?

We conducted several simple experiments in order to evaluate different SAT solvers. We considered state-of-the-art solvers available from the website of SAT Competition [12]. We included also *lingeling*, used in the 2017 work [11]. The criterion we used was the overall CPU user time consumed by all the spawned processes. The test case for performance evaluation was the computation of χ'_c for all cyclically 4-connected snarks on 24 vertices. There are 155 such graphs and the computation for each of them involved solving several SAT instances as explained in Section 2.

The results are shown in Table 2 which shows total run time T and comparison to the *CaDiCaL_vivinst*, denoted T' , which was one of the winners of the competition (the third place in main track). The table also shows the speedup achieved on the 20 CPU machine denoted S_{20} . At first sight, the *MergeSat* solver looks like quite a bad choice but as it was always using two computational threads the real time of result delivery was ac-

³The algorithm first constructs a linear ordering of edges such that the next edge in the ordering has the most incident edges among the edges already included in the ordering, and then tries all possible colours for each edge in the ordering subsequently, backtracking when a colouring conflict arises. Our other attempts at backtracking did not yield anything meaningfully faster.

tually shorter, in theory, the real time might be half of the CPU time. The *Kissat* variants do about 10% better than the *CaDiCaL*. The solver *lingeling* used in previous work is about 7 years old but was only 1.5 times slower. The *SeqFROST* solver appeared to do much better on our benchmark task than the reference SAT solver. Repeated runs for each solver gave almost the same results, the differences in time consumption were all less than 0.5%.

In another experiment we let the solvers search for χ'_c of one single graph on 36 vertices with girth 5 (the result is 10/3). The results presented in Table 3 show a different perspective than previous benchmark. For this kind of task, all solvers except plain *Kissat* beats the *CaDiCaL*. Even several years old *lingeling* was almost twice as fast. For *MergeSat*, the total time consumed by two threads is shown, while the running time was only 1304 seconds, which is 44% of the time needed by *CaDiCaL*. Interestingly, *SeqFROST*⁴ repeated the best performance as it took only one quarter of the time of the reference solver.

For computations taking hours, solver selection is not worth the effort. But for longer tasks taking weeks, it is likely to pay off. A reasonable way is to randomly pick a small sample of the instances in question and test them on solvers that are available. A word of warning: solvers might contain rarely-encountered bugs; it has happened in the past for the solvers submitted to the competition. So in situations where you are proving something for all instances, it would be better to stay with a more mainstream and seasoned solver; if you are trying to just find some suitable object that can be verified later, possible bugs in a freshly implemented solver are not a critical problem.

4.4. Should I strive for the best solver configuration?

We experimented with tweaking configurable solver parameters a bit (for instance, *lingeling* has hundreds of them). The best speedup we were able to get is 2-4x for a single graph and less than 2x for a large set of graphs. However, it is unclear if the gains are transferable to other input instances. For instance, the distribution of

⁴A winner of SAT Competition 2022, https://gears.win.tue.nl/papers/sc2022_seqfrost.pdf, <https://github.com/muhos/SeqFROST>

Table 3Comparison of SAT solvers. Consumed CPU time in seconds for χ'_c of one snark on 36 vertices with girth 5.

Solver	lingeling (2017)	MergeSat (two threads*)	Kissat 3.1.0	Kissat hywalk-exp	Kissat MAB_DeepWalk+	CaDiCaL vivinst'	SeqFROST
T[s]	1599	2387	3758	1917	2220	2943	775
T/T'[%]	54	81*	128	65	75	100	26

graph features allowing the speedup for smaller graphs might be different for larger graphs. Additionally, fine-tuning the parameters takes quite some time because of the necessary repeated solver runs. Overall, if your computations are going to run for weeks, it might be worthwhile to work on this, but for quick verification of conjectures and the like, not really. Also, note that in Section 4.3, *lingeling* turned out to be slow for small instances, but fast for large instances, thus solver configuration should be focused on bigger instances primarily or even exclusively.

Your time might be better spent on trying to break the symmetry of your Boolean formulas [14]. For instance, for circular edge-colouring, one can pre-colour an edge with 0 without a loss of generality, and immediately get rid of a handful of variables, potentially gaining a better speedup than anything that could be gained by tweaking solver parameters.

Our experience with parallelism is that it does not pay off to use multiple threads per single SAT instance. It seems preferable to solve many instances in parallel, each in its own thread.

4.5. Alternatives to SAT

Kunertová [11] described several alternatives for computing χ'_c , some of them with measurements of computation time. It seems that they all fall far short of SAT solvers (unless we are talking about very small instances, where backtracking is king). These alternatives typically rely on some specific properties of circular colourings, so are not generalizable and we will not discuss them. One of the approaches, though, deserves attention.

For a graph G with $\chi'_c(G) = r$, one way to prove the lower bound is to refute the existence of an r' -edge-colouring for r' being the largest relevant fraction smaller than r . Another is finding all r -edge-colourings of G and checking that each of them contains a so-called tight cycle. While it is not particularly fruitful for circular colourings [11], it might sometimes be the most efficient approach for graphs of limited size. The problem of finding all satisfying assignments to a given Boolean formula is called AllSAT and there are specialized solvers for it [15], although not many, and they are not heavily employed, so there are concerns about reliability (but any ordinary SAT solver can be easily turned into an AllSAT

solver by repeatedly adding clauses excluding already found solutions to the input formula). There is a fair chance this might work if the number of solutions is in thousands or perhaps millions. With billions of solutions (or larger input instances in general), solvers tend to run out of memory or experience a significant slowdown.

As an example of this method, it is much easier to describe a 2-factor than a Hamiltonian cycle (because the conditions encoded into the Boolean formula are local, around a single vertex)—the resulting formula has a linear number of variables. For each 2-factor F computed by an AllSAT solver, we then check if F is connected. This approach is comparable to or even somewhat faster than direct SAT solving for cubic graphs with around 30 vertices, but it fails for graphs above 40 vertices because the number of 2-factors grows exponentially [16].

One possible reason for the success of generic SAT solvers on graph problems (versus other approaches tailored to solve specific problems) is that a solver can see more than is obvious. We have experienced this when considering an alternative approach to colourings: instead of running an exponential algorithm on the whole graph at once, what if we split the graph along a small cut, ran the slow algorithm on the much smaller parts separately, and then combined the results? It turned out it is no big win (unless the graph in question contains many isomorphic subgraphs), possibly because all the small cuts are reflected in the Boolean formula (as subsets of clauses barely sharing variables), and thus the solver can work with all the cuts as it sees fit anyway. Also, leading solvers include tricks like random restarts, so e.g. if the colouring conflict is in a specific subgraph, the solver does not get stuck outside of this subgraph for too long.

SAT solvers also have a disadvantage: there is no effective way of reporting computation progress (unlike for plain backtracking having at least a somewhat balanced search tree). In case one wants to solve a single particularly hard instance, when does one give up?...

A note on integer/mixed linear programming (ILP): every SAT instance can be expressed as a problem of ILP, and there are powerful ILP solvers, so ILP might be worth a try for your problem. Our limited experience shows that ILP is significantly slower, and it comes with numerical and rounding problems, so perhaps the best use of it would be for refuting working hypotheses or for

guessing optimal values which are then proved otherwise (by hand or using a SAT solver).

Our feeling is that a SAT solver should perhaps be the first tool to try (unless there exists a solver specifically designed or easily adaptable for your purpose, e.g. for the travelling salesman problem).

Acknowledgments

We thank J. Karabáš, R. Lukořka, and R. Nedela for interesting discussions related to the topics covered in this article.

This work was partially supported from the research grants APVV-19-0308, VEGA 1/0743/21, VEGA 1/0727/22 and Operational Program Integrated Infrastructure for the project Advancing University Capacity and Competence in Research, Development and Innovation (ACCORD, ITMS2014+:313021X329), co-financed by the European Regional Development Fund.

Part of the research results was obtained using the computational resources procured in the national project National competence centre for high performance computing (project code: 311070AKF2) funded by European Regional Development Fund, EU Structural Funds Informatization of society, Operational Program Integrated Infrastructure.

References

- [1] J. Mazák, J. Rajník, M. Škoviera, Morphology of small snarks, *Electronic Journal of Combinatorics* 29(4) (2022) P4.30.
- [2] J. Karabáš, E. Máčajová, R. Nedela, M. Škoviera, Girth, oddness, and colouring defect of snarks, *Discrete Mathematics* 345 (2022) 113040. doi:10.1016/j.disc.2022.113040.
- [3] G. Mazzuoccolo, E. Steffen, Nowhere-zero 5-flows on cubic graphs with oddness 4, *Journal of Graph Theory* 85 (2017) 363–371. doi:10.1002/jgt.22065.
- [4] X. Zhu, Circular chromatic number: a survey, *Discrete Mathematics* 229 (2001) 371–410. doi:10.1016/S0012-365X(00)00217-X.
- [5] P. Afshani, M. Ghandehari, M. Ghandehari, H. Hatami, R. Tusserkani, X. Zhu, Circular chromatic index of graphs of maximum degree 3, *Journal of Graph Theory* 49 (2005) 325–335.
- [6] D. Král, E. Máčajová, J. Mazák, J.-S. Sereni, Circular edge-colorings of cubic graphs with girth six, *Journal of Combinatorial Theory, Series B* 100 (2010) 351–358. doi:10.1016/j.jctb.2009.10.003.
- [7] J. Mazák, Circular edge-colourings of cubic graphs, PhD thesis, 2011.
- [8] T. Kaiser, D. Král, R. Škrekovski, X. Zhu, The circular chromatic index of graphs of high girth, *Journal of Combinatorial Theory, Series B* 97 (2007) 1–13. doi:10.1016/j.jctb.2006.03.002.
- [9] R. Lukořka, J. Mazák, Cubic graphs with given circular chromatic index, *SIAM Journal on Discrete Mathematics* 24 (2010) 1091–1103. doi:10.1137/090752316.
- [10] K. Coolsaet, S. D’hondt, J. Goedgebeur, House of graphs 2.0: A database of interesting graphs and more, *Discrete Applied Mathematics* 325 (2023) 97–107. doi:10.1016/j.dam.2022.10.013.
- [11] O. Kunertová, Circular chromatic index of small snarks, Diploma thesis supervised by J. Mazák, 2017. URL: <http://www.dcs.fmph.uniba.sk/diplomovky/obhajene/getfile.php/kunertova.pdf?id=426&fid=756&type=application%2Fpdf>.
- [12] T. Balyo, M. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), *Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions*, Department of Computer Science Series of Publications B, Department of Computer Science, University of Helsinki, Finland, 2023.
- [13] R. Lukořka, J. Tětek, A 3-edge-coloring algorithm, *Bordeaux Graph Workshop* (2019). URL: <https://bgw.labri.fr/2019/booklet.pdf>.
- [14] H. Metin, S. Baarir, M. Colange, F. Kordon, Cdclsym: Introducing effective symmetry breaking in sat solving, in: D. Beyer, M. Huisman (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, Springer International Publishing, Cham, 2018, pp. 99–114.
- [15] T. Toda, T. Soh, Implementing efficient all solutions sat solvers, *ACM J. Exp. Algorithmics* 21 (2016). doi:10.1145/2975585.
- [16] L. Esperet, F. Kardoš, A. D. King, D. Král, S. Norine, Exponentially many perfect matchings in cubic graphs, *Advances in Mathematics* 227 (2011) 1646–1664. doi:10.1016/j.aim.2011.03.015.