A fresh look at a randomized massively parallel graph coloring algorithm

Boštjan Gabrovšek[1,2] and **Janez Žerovnik**[1,3,*]

[1] *University of Ljubljana, Faculty of Mechanical Engineering, Aškerčeva 6, 1000 Ljubljana, Slovenia*

[2] *Institute of Mathematics, Physics and Mechanics, Jadranska ulica 19, 1000 Ljubljana, Slovenia*
E-mail: ⟨*bostjan.gabrovsek@fs.uni-lj.si*⟩

[3] *Rudolfovo – Science and Technology Centre Novo mesto, Podbreznik 15, 8000 Novo mesto, Slovenia*
E-mail: ⟨*janez.zerovnik@fs.uni-lj.si*⟩

Abstract. Petford and Welsh introduced a sequential heuristic algorithm to provide an approximate solution to the NP-hard graph coloring problem. The algorithm is based on the antivoter model and mimics the behavior of a physical process based on a multi-particle system of statistical mechanics. It was later shown that the algorithm can be implemented in a massively parallel model of computation. The increase in computational processing power in recent years allows us to perform an extensive analysis of the algorithms on a larger scale, leading to the possibility of a more comprehensive understanding of the behavior of the algorithm, including the phase transition phenomena.

Keywords: combinatorial optimization, graph coloring, randomized local search procedure, temperature

Received: January 25, 2024; accepted: June 4, 2024; available online: October 7, 2024

DOI: 10.17535/crorr.2024.0009

1. Introduction

Graph coloring is one of the most studied NP-hard problems in combinatorial optimization [17]. In addition to its purely combinatorial appeal, the graph coloring problem is widely used in many engineering projects, including various practical problems such as planning and scheduling, timetabling, and frequency assignment [1, 18]. It is generally believed that NP-hard problems cannot be solved optimally in times that are polynomially bounded functions of the input size. However, the conjecture P≠NP? is still an open problem. This conjecture is one of the most important problems in contemporary mathematics because it is on the list of seven millennium problems for which a prize of 1 million dollars is offered by the Clay Institute [9]. NP-complete problems are decision problems with the following property: if any of them enjoys a polynomial time algorithm giving the exact solution, then P=NP, thus solving the P versus NP problem [15, 12]. On the other hand, it is widely believed that P≠NP, in other words, that there is no polynomial time algorithm for any NP-complete problem. It also holds true for NP-hard problems, i.e., problems that are at least as difficult as an NP-complete problem. Therefore, there is great interest in heuristic algorithms that can find near-optimal solutions to the graph coloring problem (or any other NP-hard problem) within reasonable running times for a large number of instances [11].

Threshold phenomena have attracted a lot of attention in the context of random combinatorial problems [8] and in theoretical physics [21]. In statistical physics, phase transitions have

[*]Corresponding author.

been studied for more than a century. Let us only mention here that spin glasses, a purely theoretical concept, have triggered a new branch of theoretical physics that resulted in a Nobel Prize for Giorgio Parisi in 2022 [19]. The Ising model is a statistical model that can be used to describe the energy of a system of atoms arranged in a lattice, where the interaction between these atoms is governed by the interaction between their spins and a possible external magnetic field. This model and some more complicated versions of it can then be used to describe the behavior of spin glasses. It is well-known that the Ising model is in general NP hard [3]. On the other hand, the Ising model is closely related to graph theory, in particular to the graph coloring problem [3]. Graph coloring with $k = 3$ colors has been considered in several papers, see [5, 2] and references therein. In a study of random graphs it was found that the phase transition is closely related to the mean degree of the graphs. In the same paper, the analysis implies that the hardest instances are among random graphs with an average degree between 4.42 and $\alpha_{crit} \approx 4.7$.

Based on the antivoter model of Donnely and Welsh [6], a randomized algorithm for graph coloring that performs very well on some random graphs [20, 27] was proposed by Petford and Welsh. The algorithm was later successfully tested on some other types of graphs [23]. With some straightforward modifications, the algorithm was also very competitive in frequency-assignment problems [7, 24, 28]. Petford and Welsh experimented with their algorithm on a model of random 3-colorable graphs. They observed that there are some combinations of parameters of random graphs that are extremely hard for their algorithm, while their algorithm otherwise runs in linear time, on average. Having no theoretical explanation, Petford and Welsh write that the curious behavior "is not unlike the phenomenon of phase transition that occurs in the Ising model, Potts model and other models of statistical mechanics".

We have recently designed an algorithm for clustering that is motivated by this coloring algorithm. The results were very promising [14]. This motivated us to better understand the basic algorithm, which we did in an experimental study that is reported here. As high inherent parallelism is an interesting feature of the approach, we implemented a parallel version of the algorithm, keeping in mind its potential use in alternative models of computation. The algorithm was tested on a large quantity of data so as to have a better understanding of its performance. As the parallel execution is simulated on a sequential machine, the computation time is naturally measured in the number of parallel steps. In particular, we run and analyse the following experiments:

- In the experiments reported in [25, 26], the algorithms perform poorly in some instances. We perform several experiments and test the hypothesis that the performance of the algorithm depends only on the average degree for random graphs and for regular graphs.
- To test whether the performance of the algorithm depends on the average degree, we test the performance by varying the average degree of the graph and the temperature (parameter T) of the system. We test the performance on random graphs and on r-regular graphs (both are known to be colorable).
- We test whether the algorithm behaves in a similar way for higher-degree colorings ($k = 4, 5, \ldots, 10$) and confirm the existence of critical regions that are related to the phase transition phenomena.

The rest of the paper is organized as follows. In Section 2 we recall the k-coloring decision problem and the algorithms described in [25] and [26]. In Section 3 we present the new experiments and analyse the results. For our experiments we use two classes of randomly generated k-colorable graphs.

2. Graph coloring problem and the algorithm of Petford and Welsh

Graph coloring problem. The k-coloring decision problem (for $k \geq 3$) is a well-known NP-complete problem. It is formally stated as follows:

> *Input: graph G, integer k*
> *Question: is there a proper k-coloring of G?*

A *coloring* is any mapping $c : V(G) \to \mathbb{N}$ and is a feasible solution of the k-coloring problem. A mapping $c : V(G) \to \mathbb{N}$ is a *proper coloring* of G if it assigns different colors to adjacent vertices. The cost function $E(c)$ is the number of *bad* edges. By definition, the bad edges for coloring c are edges with both ends colored by the same color in coloring c. Such vertices are called *bad*. Proper colorings are exactly the colorings with $E(c) = 0$ and finding a coloring c with $E(c) = 0$ is equivalent to answering the above decision problem. The coloring constructed is a *witness* c proving the correctness of the answer. A graph for which a proper coloring with k colors exists is said to be *k-colorable*.

The algorithm of Petford and Welsh. The basic algorithm [20] starts with an initial 3-coloring of the input graph that is chosen at random. Then an iterative process is started. During each iteration, a bad vertex is chosen at random. The chosen bad vertex is recolored randomly, according to some probability distribution. The color distribution favors colors that are less represented in the neighborhood of the chosen vertex, see the expression (1) below. The algorithm has a straightforward generalization to k-coloring (taking $k = 3$ gives the original algorithm) [27].

In pseudo code, the algorithm of Petford and Welsh is written as follows

Algorithm 1 Petford-Welsh algorithm

```
1: color vertices randomly with colors $1, 2, \dots, k$
2: while not stopping condition do
3:     select a bad vertex $v$ (randomly)
4:     assign a new color $c$ to $v$
5: end while
```

A bad vertex is selected uniformly at random among vertices that are endpoints of some bad (e.g., monochromatic) edge. A new color is assigned at random. The new color is taken from the set $\{1, 2, \dots, k\}$. Sampling is conducted according to the probability distribution defined as follows:

The probability p_i of color i to be chosen as a new color of vertex v is proportional to

$$p_i \approx \exp(-S_i/T), \tag{1}$$

where S_i is the number of edges with one endpoint at v and with color i assigned to the other endpoint. Petford and Welsh used 4^{-S_i} which is equivalent to using $T \approx 0.72$ in (1). (Because $\exp(-x/T) = 4^{-x}$ implies $T \approx 0.72$.)

The stopping criterion includes two conditions. The algorithm stops either when the time limit (in terms of the number of calls to the function that computes a new color) is reached, or when a proper coloring is found. If a proper coloring is found, the answer to the decision problem is positive, and the proper coloring is reported as a witness. In the case when no proper coloring is found, the feasible solution with minimal cost $E(c)$ is reported as an approximate solution to the problem. The answer to the decision problem is in this case negative; however, it might not be correct. Note that there is no guarantee of the quality of the solution.

Connection to simulated annealing and the generalized Boltzmann machine. Here we briefly discuss the parameter T of the algorithm. Due to an analogy between the temperature of the simulated annealing algorithm and the temperature of the generalized Boltzmann machine neural network [22], parameter T can naturally be called the temperature. With the term *simulated annealing* (SA) we refer to the optimization heuristics as proposed, for example, in [16]. The *Generalized Botzmann machine* is a generalization of the Boltzmann machine, a neural network that is based on a stochastic spin-glass model and has been widely used in artificial intelligence [13]. The main difference between the two is that the generalized Boltzmann machine as defined in [22] uses multistate neurons, in contrast to the bipolar neurons of the usual Boltzmann machine.

These analogies are based on the following simple observation. Pick a vertex and denote the old color of the chosen vertex by j and the new color by i. The number of bad edges E' after the move is

$$E' = E - S_j + S_i \tag{2}$$

where E is the number of bad edges before the change. We define $\Delta E = E - E' = S_j - S_i$. During each step, j is fixed and hence S_j and E are fixed. Consequently, it is equivalent if we define the probability of choosing the new color i to be proportional to either $\exp(-S_i/T)$, $\exp(\Delta E/T)$ or $\exp(E'/T)$.

To see the relation to the Boltzmann machine, recall that the number of bad edges is a usual definition of the energy function, in both the simulated annealing and in the generalized Boltzmann machine with multistate neurons. This indicates that the algorithm of Petford and Welsh is closely related to the generalized Boltzmann machine operating at a constant temperature (for details, see [22] and the references therein). The major difference between the two is in the firing rule. While in the Boltzmann machine all the neurons are fired with equal probability, in the algorithm of Petford and Welsh, only the bad vertices are activated.

As already explained, the original algorithm of Petford and Welsh uses probabilities proportional to 4^{-S_i}, which corresponds to $T \approx 0.72$. Other choices of T are possible. On one hand, low values of T make the algorithm behave very much like an iterative improvement, and it will be quickly converging to a local minimum. On the other hand, a large T means a higher probability of accepting a move that increases the number of bad edges. Consequently, a very high T results in chaotic behavior that is similar to a pure random walk among the colorings, regardless of their energy.

Both Petford and Welsh and simulated annealing are local-search-type heuristics. Besides the different uses of T (fixed temperature versus cooling schedule), there is another slight difference. Namely, in the usual implementation of SA, a change that improves the cost is always accepted, while in the other case, the acceptance probability is used, which depends on both the difference in costs and the temperature. In the algorithm of Petford and Welsh, all the changes are made according to the probabilities using (1). The cost-improving changes thus might not be accepted, although this happens very rarely in the majority of cases.

Thus, our algorithm is similar to both the simulated annealing heuristics and to the sequential operation of the Boltzmann machine. Its acceptance probability is (for a given T) practically equivalent to the Boltzmann machine. As the Boltzmann machine is a highly parallel asynchronous device, a comparison with parallel implementations is even more interesting. Here, we recall the parallel version of the algorithm of Petford and Welsh that differs from the generalized Boltzmann machine only in the firing rules in both phases of its operation.

Parallel algorithm. In [25], a massively parallel version of Algorithm 1 was proposed. The naive algorithm (Algorithm 2) was later improved in [26] by a version that runs in two phases, thus avoiding the looping that can appear for some configurations within the instance. The improved algorithm (Algorithm 3) first aims to find a $2k$ coloring and does not recolor all the bad vertices simultaneously because each change is only done with some probability (i.e., 0.6). In the second phase, the result of the first phase provides independent sets of vertices that can be recolored in parallel without any conflict. The resulting algorithm still shows the maximum speedup in comparison to the original version, e.g., the instances solved in linear time sequentially are expected to be solved in constant time in parallel [26]. The algorithm formally reads as Algorithm 3.

Note that in the first phase, the firing rule is nearly equivalent to that of the Boltzmann machine in which each neuron (vertex) wakes up at some random time and performs the recoloring. Note that there is no synchronization among the neurons. While synchronization is not of particular importance in the first phase of our algorithm, in the second phase, it is essential that synchronization based on the result of the first phase is used.

Algorithm 2 Massively parallel variant of the Petford–Welsh algorithm (naive version)

```
1: color vertices randomly with colors $1, 2, \dots, k$
2: while not stopping condition do
3:     $bad\_vertices \leftarrow \{v \mid v is bad\}$
4:     for all $v \in bad\_vertices$ do          ⎫
5:         assign a new color $c$ to $v$          ⎬ in parallel
6:     end for                                   ⎭
7: end while
```

Algorithm 3 Massively parallel variant of the Petford–Welsh algorithm

```
1: procedure MPPW_PHASE1(G)
2:     color vertices randomly with colors $1, 2, \dots, k, k+1, \dots, 2k$
3:     while not stopping condition do
4:         $bad\_vertices \leftarrow \{v \mid v is bad\}$
5:         for all $v \in bad\_vertices$ do                              ⎫
6:             assign a new color $c$ to $v$ with a probability of 60%    ⎬ in parallel
7:         end for                                                       ⎭
8:     end while
9: return coloring $c$
10: end procedure

11: procedure MPPW_PHASE2($G$)
12:     $bc \leftarrow$ MPPW_PHASE1(G)
13:     color vertices randomly with colors $1, 2, \dots, k$
14:     while not stopping condition do
15:         $bad\_vertices \leftarrow \{v \mid v is bad\}$
16:         for all $v \in bad\_vertices$ do                ⎫
17:             if $step \bmod (2k) = bc(v)$ then           ⎬ in parallel
18:                 assign a new color $c$ to $v$           ⎭
19:             end if
20:         end for
21:     end while
22: return coloring $c$
23: end procedure
```

3. Experiments

In our experiments we use two classes of graphs. The first class is graphs of the form

$$G(n, k, p), \tag{3}$$

where n is the number of vertices, k is the number of partitions and p is the probability that two vertices from distinct partitions are adjacent (see [20, 25, 10]).

The second class is d-regular graphs of the form

$$R(n, k, d), \tag{4}$$

where n is the number of vertices, k is the number of partitions and d is the degree of vertices. The Python code for generating such graphs can be found in [10]. In short, the algorithm splits the vertices into k partitions and connects, in a random order, each vertex to d randomly chosen vertices in distinct partitions. If there are vertices left that are not of degree d and cannot be connected, we delete an edge and add two other edges. More precisely, if u and v are vertices with $\deg(u) < d$ and $\deg(v) < d$, then we find an edge $u'v'$, such that u and u' do not belong to the same partition and v and v' do not belong to the same partition. We delete the edge $u'v'$ and add edges uu' and vv' to the graph.

In both cases the partitions are of equal size if k divides n, otherwise their sizes differ by at most one vertex.

Preliminary experiment. With the code in [10] we reproduced the results from [25, 26], with a much larger sample size, $n = 10000$ (instead of $n = 100$). The results are presented in Figure 1, confirming that our implementation runs exactly the same algorithm as the original.

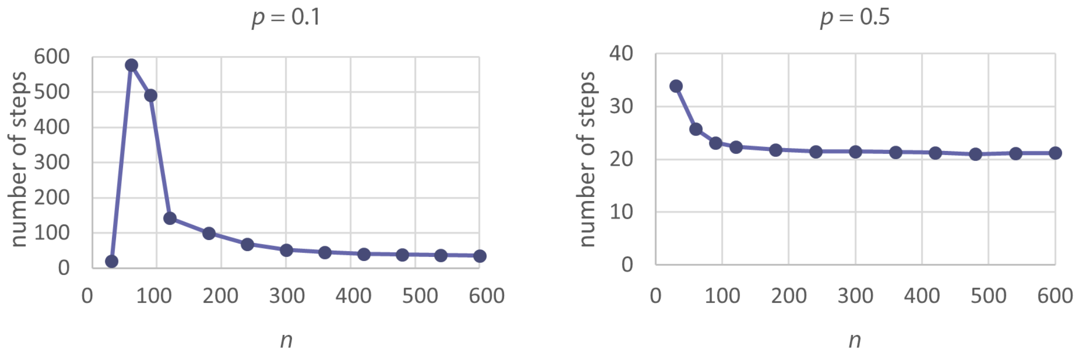

Figure 1: *Performance of IMPPW (parallel steps as a function of n), see Table 1 that resembles Table 1 in [25].*

$p = 0.1$		
n	$T_2(n)$	succ. runs
30	19.	100
60	261.	89
90	345.	93
120	142.	100
180	99.	100
240	71.	100
300	52.	100
360	45.	100
420	40.	100
480	39.	100
540	36.	100
600	35.	100
660	34.	100
720	34.	100
780	32.	100
840	33.	100
900	32.	100

$p = 0.5$		
n	$T_2(n)$	succ. runs
30	33.	100
60	25.	100
90	24.	100
120	23.	100
180	22.	100
240	22.	100
300	21.	100
360	21.	100
420	21.	100
480	21.	100
540	21.	100
600	22.	100
660	21.	100
720	22.	100
780	21.	100
840	21.	100
900	21.	100

Table 1: *Performance of IMPPW (number of parallel steps as a function of n). Number of successful runs (out of 100) is given in the third column.*

Observe that the algorithm does not perform well on a narrow interval only, which we call the *critical region*. Observing some experimental results, limited by the computing resources available at the time, the following conjecture was proposed [27].

Conjecture 1. *The critical regions are characterized by the equation*

$$\frac{2pn}{k} \approx \frac{16}{3}. \tag{5}$$

This conjecture generalizes the conjecture of Petford and Welsh, who observed that the equation $\frac{2pn}{3} \approx \frac{16}{3}$ is valid within the critical region [20]. More precisely, they observed that given p, the graphs $G(n, 3, p)$ with $n \approx \frac{8}{p}$ are likely hard instances for the algorithm.

After we confirm the basic observations in the main references, we continue with experimental results that can shed some more light on the behaviour of the algorithm and possibly on some more general phenomena. In particular, we wanted to understand better, if and how the hard instances can be related to the average degree of the graphs. In relation to this, we wish to check whether the conjecture above captures the main information that determines the critical regions. Furthermore, we are interested in the question "what is the effect of the temperature" (or, equivalently, the basis of the exponent expression (1)) on the performance of the algorithm? Below we provide the results of the experiments with some comments that answer some questions and, at the same time, highlight some new questions.

First experiment. In the first experiment we show that, with a fixed number of components, the critical region depends on the average degree $\overline{d}(G)$ of the graph, where

$$\overline{d}(G) = \frac{np(k-1)}{k}. \tag{6}$$

We choose four datasets: graphs of classes $G(90, 3, p)$, $G(120, 3, p)$, $G(300, 3, p)$, and $G(3000, 3, p)$. The sample size is 10000 for $n \in \{90, 120, 300\}$ and a bit smaller, 2000, for $n = 3000$. We choose the parameter p in such a way that the average degree of each class varies from 2 to 9. The results are presented in diagrams in Figure 2.

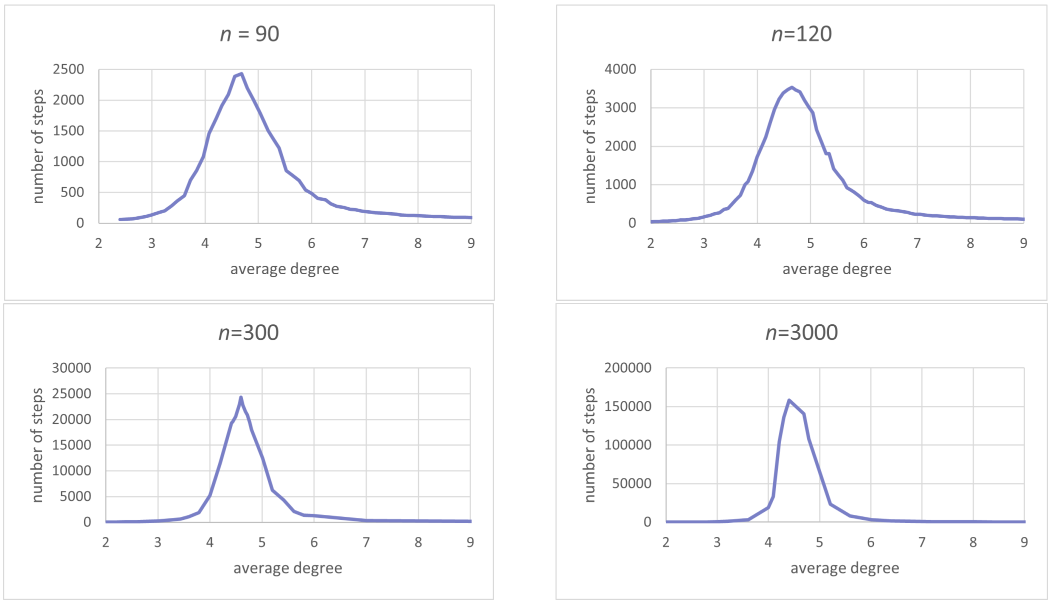

Figure 2: *Performance of IMPPW (steps as a function of average degree).*

Note that in all four experiments, the extreme value does not depend on the number of vertices. It is obvious that the hard instances are in the interval where the average degree is between 4 and 5; even more, the peaks are within the range 4.5–4.7 in all four diagrams.

The phenomena can be explained as follows. Observe that before the critical region ($\overline{d}(G) < 4.5$), the partitions (e.g., sets of vertices of the same color in a proper coloring) are loosely defined and there are multiple optimal solutions (see Supplementary Video 1). On the other hand, after the critical region ($\overline{d}(G) > 4.7$), the algorithm converges fast, since the partitions are densely connected and thus very well defined (see Supplementary Video 3). For a graph in the critical region see Supplementary Video 2. All three videos can also be accessed at [10].

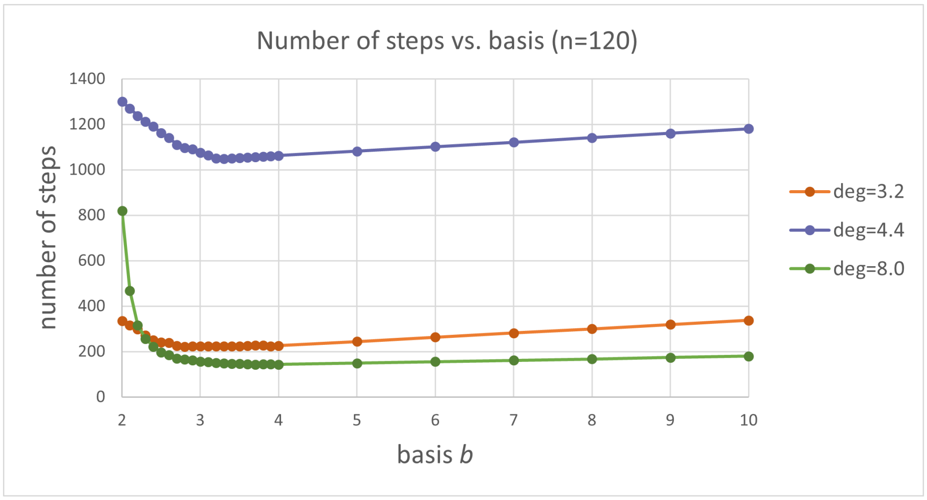

Figure 3: *Performance of IMPPW (basis vs. number of steps).*

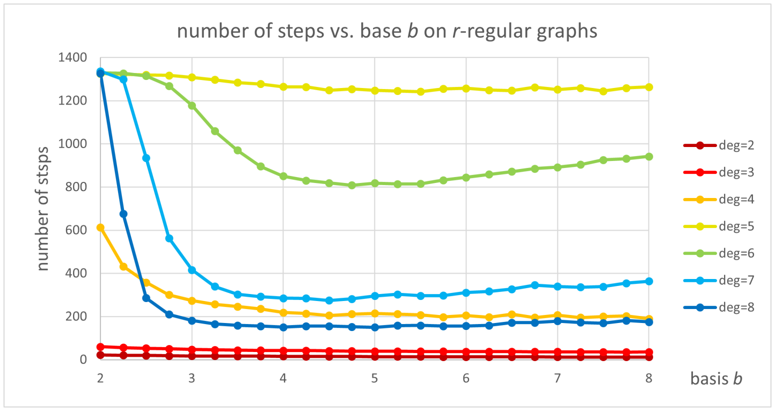

Figure 4: *Performance of IMPPW (basis vs. number of steps for d-regular graphs).*

Second experiment. In this experiment, we vary the parameter b over the values from 2 to 10 and measure the performance of the algorithm in the classes $G(120, 3, p)$, where p is chosen in such a way that the average degrees are 3.2, 4.4, and 8.0. The results are presented in the diagram in Figure 3.

We conclude that the basis b does not dramatically influence the performance. Seemingly, the values between 3.0 and 4.0 are well behaved and indeed, taking any value $b \in [3, 4]$, perform similarly. We repeat the experiment on the graphs $R(120, 3, d)$, where $d = 2, 3, \ldots, 8$. The results are presented in the diagram in Figure 4 and clearly confirm the earlier observations.

The question "what is the optimal value b?" should have an answer between 3 and 4. Note that this is a question equivalent to the question as to which is the optimal temperature of the simulated annealing (i.e. "annealing" with constant temperature)? However, this seems to be a rather complex problem [4, 29]. (Recall that $\exp(-x/T) = b^{-x}$ so $b = \exp(1/T)$.) As our insight is limited by the special class of instances used, we do not wish to dig deeper into the question of optimal b. On the other hand, we can confirm that, probably, the algorithm is robust regarding the choice of b and, equivalently, to the choice of parameter T).

Third experiment. In this experiment we compute the run times for the graphs in $G(n, k, p)$ where we fix $k = 3, 4, \ldots, 8$ partitions and $n = 60, 120, 240$ vertices, and in each case observe how the number of steps needed depends on the probability p. In all the experiments the sample size is 5000. The results are presented in Figure 5.

According to Conjecture 1, the hard instances are characterized by some constant value of $\frac{2pn}{k}$. However, in Figure 6 we observe how the performance depends on $\frac{2pn}{k}$. We conclude that the critical region is not characterized exactly by $\frac{2pn}{k}$, thus the Conjecture 1 should be replaced by a better one.

In Figure 7 we plot how the number of steps depends on the expression $\frac{p(k-1)}{k}$, which is proportional, if n is fixed, to the average degree. We observe from the figure that the critical region cannot be explained only by the average degree.

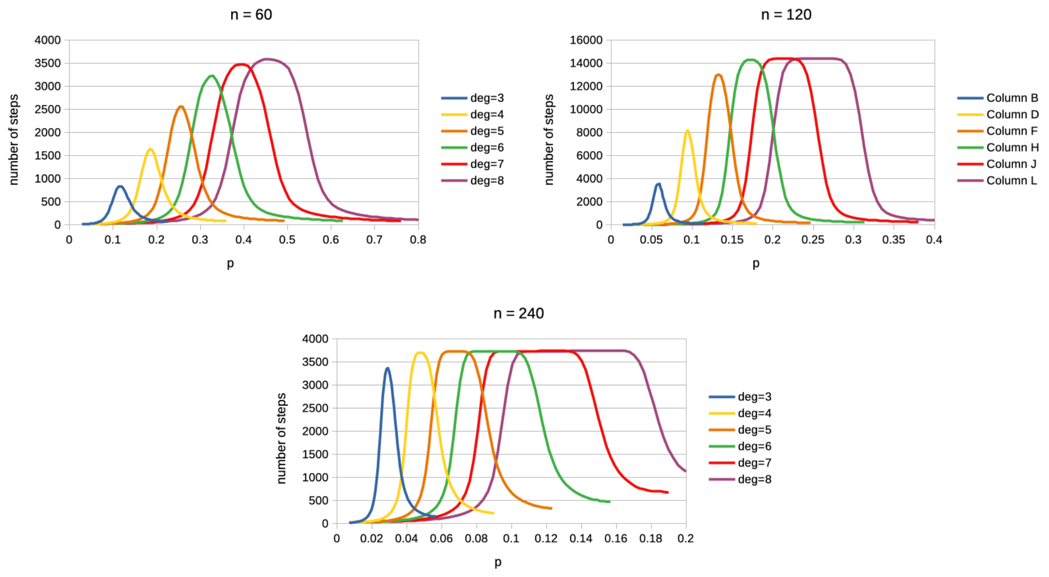

Figure 5: *Performance of IMPPW for k-colorings. The graphs show how the number of steps depends on the parameter p.*

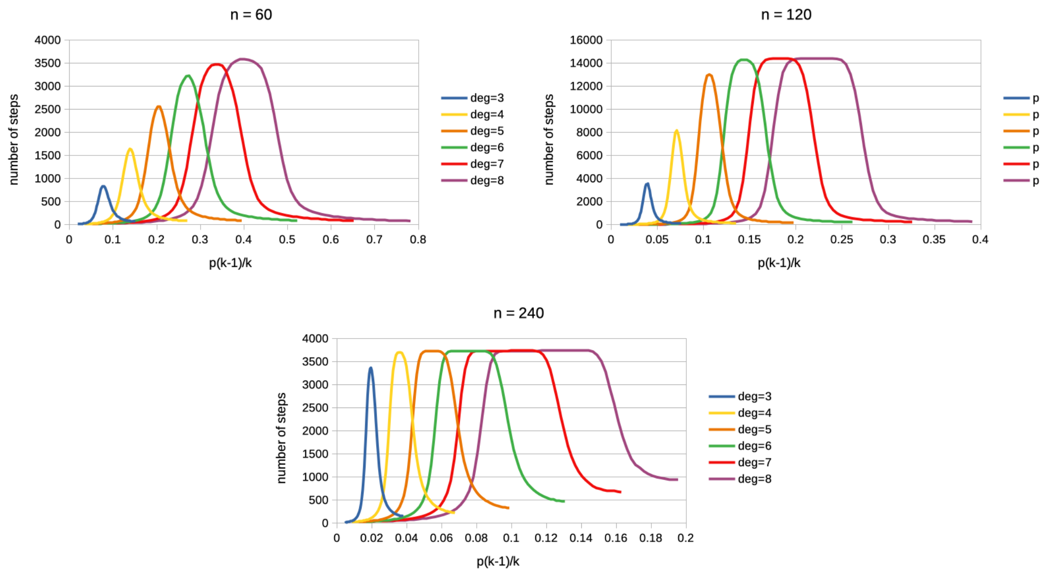

Figure 7: *Performance of IMPPW for k-colorings. The graphs show how the number of steps depends on the average degree, which is proportional to $\frac{p(k-1)}{k}$.*

At present we do not have a good idea of how to improve the conjecture to better characterize the critical region with an expression that would have some natural meaning. We have tested some slight modifications and found that the expression $\frac{p}{k-1.5}$ remains fairly constant for varied k. See Figure 8, where we plot how the number of steps depends on the expression $\frac{p}{k-1.5}$.

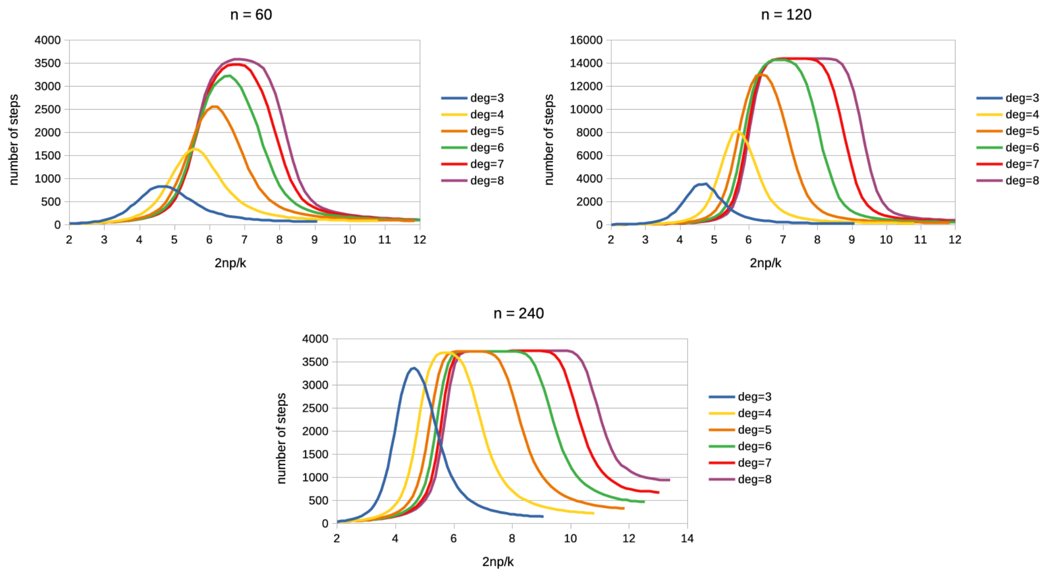

Figure 6: Performance of IMPPW for k-colorings. The graphs show how the number of steps depends on the parameter p.

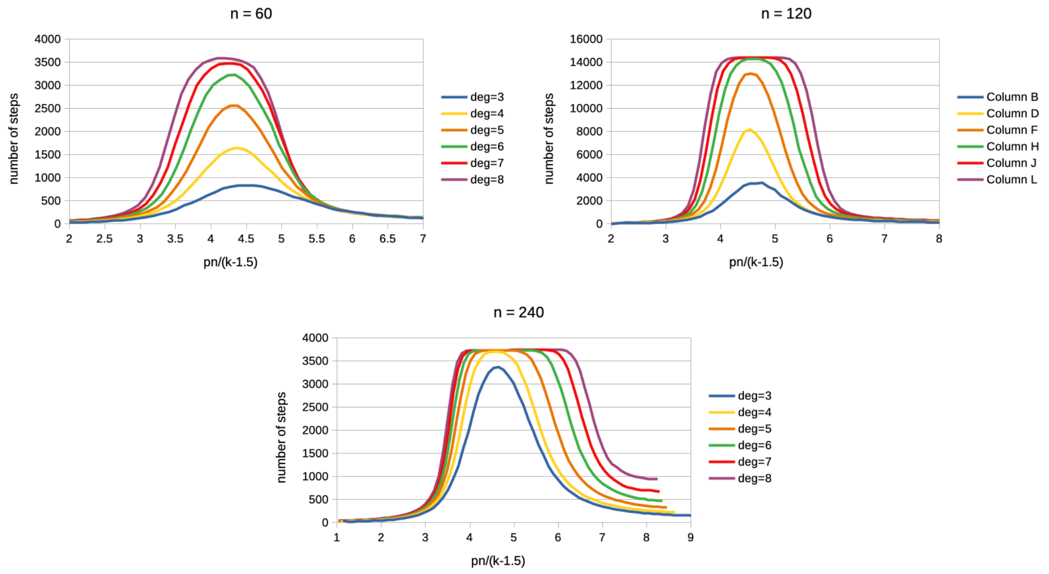

Figure 8: *Performance of IMPPW for k-colorings. The graphs show how the number of steps depends on the expression $\frac{pn}{k-1.5}$. The cut-off is due to the time constraint of the algorithm.*

Therefore, it seems that the critical region is approximately characterized by the equation

$$\frac{np}{k-1.5} \approx 4.3. \tag{7}$$

4. Conclusions

A parallel version of Petford and Welsh's k-coloring algorithm was extensively tested on two classes of random graphs. The conjecture about the existence of a critical region where the algorithm has nearly prohibitively long run times was confirmed for the case $k = 3$, while a generalized conjecture [27] was shown to need an adjustment. More precisely, we have observed that the critical region appears where $\frac{p}{k-1.5}$ holds. This result opens at least two interesting questions.

- can the property $\frac{np}{k-1.5} \approx 4.3$ be naturally explained as some feature of the instances?
- is there another expression that fits the data, and has some meaning that can explain the structure of hard instances?

In contrast to the graph classes of k colorable graphs used in this paper, the usual random graph model considered are graphs $G(n, p)$ where each of the possible $\frac{n(n-1)}{2}$ edges appears independently with a probability p. Not surprisingly, for 3-coloring, it is found that the critical mean degree where the phase transition occurs is around $\alpha_{crit} \approx 4.7$, for example, the estimate 4.703 was put forward in [2]. In the same paper, the analysis implies that the hardest instances are among the graphs with an average degree between 4.42 and α_{crit}. It seems that the k-coloring was not considered, hence we cannot compare our findings about the hardest instances with previous work. We conclude that further study of the critical regions is a promising avenue of research that might have some implications that go beyond understanding of the behavior of the algorithm of Petford and Welsh.

Acknowledgements

The authors were supported in part by the Slovenian Research Agency, grants J2-2512, J1-4031, N1-0278 (B. Gabrovšek), and J2-2512 and P2-0248 (J. Žerovnik).

References

[1] Babaei, H., Karimpour, J. and Hadidi, A. (2015). A survey of approaches for university course timetabling problem. Computers & Industrial Engineering, 86, 43-59. doi: 10.1016/j.cie.2014.11.010

[2] Boettcher, S. and Percus, A. G. (2004) . Extremal optimization at the phase transition of the three-coloring problem. Phys. Rev. E, 69, 066703. doi: 10.1103/PhysRevE.69.066703

[3] Cipra, B. A. (2000). The Ising model is NP-complete. SIAM News, 33(6), 1-3. Retrieved from: semanticscholar.org

[4] Cohn, H. and Fielding, M. (1999) . Simulated annealing: searching for an optimal temperature schedule. SIAM Journal on Optimization, 9, 779-802. doi: 10.1137/S1052623497329683

[5] Culberson, J. and Gent, I. (2001). Frozen development in graph coloring. Theoretical Computer Science, 265, 227-264. doi: 10.1016/S0304-3975(01)00164-5

[6] Donnelly, P. and Welsh, D. (1984). The antivoter problem: random 2-colourings of graphs. In Graph Theory and Combinatorics (Cambridge, 1983), (pp. 133-144), Academic Press, London.

[7] Chamaret, B., Ubeda, S. and Žerovnik, J. (1996). A Randomized Algorithm for Graph Colouring Applied to Channel Allocation in Mobile Telephone Networks. Proceedings of the 6th International Conference on Operational Research KOI'96. 25-30, Croatian Operational Research Society, Zagreb.

[8] Dubois, O., Monasson, R., Selman, B. and Zecchina, R. (2001). Editorial. Theoretical Computer Science, 265(1–2), 1. doi: 10.1016/S0304-3975(01)00133-5

[9] Fortnow, L. (2009). The status of the P versus NP problem. Commun. ACM 52, 9 (September 2009), 78–86. doi: 10.1145/1562164.1562186

[10] Gabrovšek, B. (2023). Massively parallel algorithm for graph coloring based on the Petford-Welsh algorithm, source code, Retrieved from: github.com/bgabrovsek/petford-welsh-coloring

[11] Galinier, P. and Hertz, A. (2006). A survey of local search methods for graph coloring. Computers Operations Research, 33(9), 2547–2562. doi: 10.1016/J.COR.2005.07.028

[12] Garey, M. R. and Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman.

[13] Hinton, G. (2011). Boltzmann Machines. In Sammut, C. and Webb, G.I. (Eds.) Encyclopedia of Machine Learning. Springer, Boston, MA. doi: 10.1007/978-0-387-30164-$8_8$3

[14] Ikica, B., Gabrovšek, B., Povh, J. and Žerovnik, J. (2022). Clustering as a dual problem to colouring. Computational and Applied Mathematics, 41, 147. doi: 10.1007/s40314-022-01835-0

[15] Karp, R. (1972), Reducibility among combinatorial problems. In R. Miller and J. Thatcher (Eds.) Complexity of Computer Computations. Plenum Press, 85-103.

[16] Kirckpatrick, S., Gellat, C. D. and Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598), 671-680. doi: 10.1126/science.220.4598.671

[17] Lewis, R. M. R. (2021). Guide to Graph Colouring, Texts in Computer Science (second edition), Spirnger Nature Switzerland. doi: 10.1007/978-3-030-81054-2

[18] Martín, H. J. A. (2013). Solving Hard Computational Problems Efficiently: Asymptotic Parametric Complexity 3-Coloring Algorithm. PLoS ONE, 8(1), e53437. doi: 10.1371/journal.pone.0053437

[19] Mézard, M. (2022). Spin glasses and optimization in complex systems. Europhysics News, 53(1), 15-17. doi: 10.1051/epn/2022105

[20] Petford, A. and Welsh, D. (1989). A Randomised 3-coloring Algorithm. Discrete Mathematics, 74, 253-261. doi: 10.1016/0012-365X(89)90214-8

[21] Philathong, H., Akshay V., Samburskaya, K. and Biamonte, J. (2021). Computational phase transitions: benchmarking Ising machines and quantum optimisers. Journal of Physics: Complexity, 2(1), 011002. doi: 10.1088/2632-072X/abdadc

[22] Shawe-Taylor, J., and Žerovnik, J. (1992). Boltzmann Machines with Finite Alphabet. In Proceedings International Conference on Artificial Neural Networks, ICANN'92. Elsevier Science. 391-394 . Retrieved from: eprints.soton.ac.uk

[23] Shawe-Taylor, J., and Žerovnik, J. (1995). Analysis of the Mean Field Annealing Algorithm for Graph Colouring. Journal of Artificial Neural Networks, 2, 329-340.

[24] Ubeda, S. and Žerovnik, J. (1997). A randomized algorithm for a channel assignment problem. Speedup, 11, 14-19.

[25] Žerovnik, J. (1990). A parallel variant of a heuristical algorithm for graph coloring. Parallel Computing, 13, 95-100.

[26] Žerovnik, J. and Kaufman, M. (1992). A parallel variant of a heuristical algorithm for graph coloring - corrigendum. Parallel Computing, 18, 897-900.

[27] Žerovnik, J. (1994). A Randomized Algorithm for k-colorability. Discrete Mathematics, 131, 379-393. doi: 10.1016/0012-365X(94)90402-2

[28] Žerovnik, J. (1998) . On the convergence of a randomized algorithm frequency assignment problem. Central European Journal for Operations Research and Economics, 6, 135-151.

[29] Žerovnik, J. (2000). On temperature schedules for generalized Boltzmann machine. Neural Network World, 3, 495-503.