A note on packing chromatic number of the square lattice

Roman Soukal [1]

Přemysl Holub [2]

Abstract.

The concept of a packing colouring is related to a frequency assignment problem. The packing chromatic number $\chi_p(G)$ of a graph G is the smallest integer k such that the vertex set $V(G)$ can be partitioned into disjoint classes $X_1, \ldots, X_k$, where vertices in X_i have pairwise distance greater than i. In this note we improve the upper bound on the packing chromatic number of the square lattice.

Keywords: Packing chromatic number, Packing colouring, Square lattice

AMS Subject Classification (2000): 05C15, 05C12

1 Introduction

In this paper we consider only simple undirected graphs. We use [1] for terminology and notation not defined here. Let $\mathrm{dist}_G(x, y)$ denote the distance between vertices x and y in G. The *Cartesian product* $G\Box H$ of graphs G and H is the graph with vertex set $V(G) \times V(H)$, where vertices (x, y) and (x', y')

[1]Department of computer science and engineering, Univerzitni 22, 306 14 Pilsen, Czech Republic, e-mail: soukal@kiv.zcu.cz. Research supported by the Grant Agency of the Czech Republic - the project GA 201/09/0097.

[2]Department of Mathematics, University of West Bohemia, and Institute for Theoretical Computer Science (ITI), Charles University, Univerzitni 22, 306 14 Pilsen, Czech Republic, e-mail: holubpre@kma.zcu.cz. Research Supported by Grant No. 1M0545 of the Czech Ministry of Education.

are adjacent whenever $xx' \in E(G)$ and $y = y'$, or $x = x'$ and $yy' \in E(H)$. For two infinite paths P_1, P_2, the Cartesian product $P_1 \Box P_2$ is usually called the *(infinite) square lattice.*

The concept of the packing colouring comes from the area of frequency planning in wireless networks, and was introduced by Goddard et al. in [3] under the name *broadcast colouring.* The *packing chromatic number* of a graph G, denoted by $\chi_p(G)$, is the smallest integer k such that $V(G)$ can be partitioned into k disjoint sets $X_1, \ldots, X_k$, where for each pair of vertices $x, y \in X_i$ $\mathrm{dist}_G(x, y) > i$, for every $i = 1, \ldots, k$. In other words, vertices with the same colour i are pairwise at distance greater than i.

Goddard et al. showed in [3] that the packing chromatic number of the infinite square lattice is finite, more precisely between 9 and 23. Fiala and Lidický in [2] improved the lower bound to 10, and Schwenk in [8] has shown that the packing chromatic number of the infinite square lattice is at most 22.

Using a computer we found the following result improving the upper bound of $\chi_p(G)$ for the infinite square lattice.

Theorem 1. *Let G be the infinite square lattice. Then $\chi_p(G) \leq 17$.*

The square pattern on 24×24 vertices (see Figure 1) can be copied for the whole infinite square lattice.

2 Method and algorithm

In [3] it was shown that to decide if, for a general graph G, $\chi_p(G) \leq 4$ is NP-complete. For this problem we used heuristics based on simulated annealing (SA).

SA is one of methods to determine a suboptimal solution of combinatorial problems, mostly NP-complete. This means that there is a large space of acceptable solutions and it is not possible to find a globally optimal solution in polynomial time. As its name implies, SA exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline struc-

ture (the annealing process) and the search for a minimum in a more general system. The algorithm is based upon that of Metropolis et al. [6], which was originally proposed as a means of finding the equilibrium configuration of a collection of atoms at a given temperature. The connection between this algorithm and mathematical minimization was first noted by Pincus [7], but it was Kirkpatrick et al. [4] who proposed that it form the basis of an optimization technique for combinatorial (and other) problems. SA's major advantage over other methods is an ability to avoid becoming trapped at local minima. The algorithm employs a random search which not only accepts changes that decrease objective function f, but also some changes that increase it. For more information about SA we refer to [5]. The main idea of SA algorithm is that we iteratively search for a $x_{min} \in D(f)$ (where $D(f)$ is a definition scope of an objective function f) such that $f(x_{min})$ is as close as possible to the minimum of $f(x)$. The searching process is based on a degression of a temperature t. For a fixed t there are several iterations and in each of these iterations we choose an arbitrary x_p from a neighbourhood of current x_{min} (the radius of the neighbourhood depends on a current iteration, higher iteration means smaller neighbourhood). A new x_p is accepted with a probability $p = e^{\frac{f(x_{min})-f(x_p)}{t}}$ which means, that a better solution (x_p with smaller value of $f(x)$) is accepted automatically ($p \geq 1$) and a worse solution is accepted with a specific probability, which decreases with lower temperature t.

3 Algorithms

In this section we present a pseudocode of used algorithms. This variation of SA algorithm gives us a colouring of a square pattern that we can repeat for the whole infinite square lattice, and the obtained colouring is a packing colouring. Hence we can fill an infinite square grid (each square on 24×24 vertices), where each square is represented by our square pattern. The main idea is to colour as many vertices of a (partially filled) square pattern with a current colour c as possible. We have a list of the best found patterns and we

memorize each pattern (up to symmetric patterns) with maximum number of vertices coloured with c in this list. Thus we start with putting a colour one into an empty square pattern, then we continue with putting a colour 2 into a pattern (patterns) filled with colour one, etc., until at least one pattern from the list of the best found patterns is whole coloured.

In the pseudocode of Algorithm 1 we use two lists of patterns. The list Γ is a list of the best found patterns (with the most number of coloured vertices) after colouring with colour $(c-1)$. The list Δ is a list of currently best found patterns after colouring with c. We also define the following functions:

(i) *NonColoured (pattern K)* - gives the number of non-coloured vertices in a pattern K.

(ii) *FreeForColour (colour c, pattern K)* - gives the number of vertices of K which can be coloured with a colour c. Note that if none of the vertices of K is coloured with c, then *FreeForColour (colour c, pattern K) = NonColoured (pattern K)*, and, on the other hand, if there is a vertex in K coloured with c, then non-coloured vertices at distance not greater than c cannot be coloured with c, hence *FreeForColour (colour c, pattern K) < NonColoured (pattern K)*.

(iii) *PlantColour (pattern K, colour c, temperature t)* - puts current colour c in K under a temperature t and returns a modified pattern L. Note that there is no vertex in L which can be coloured with c.

For a current colour c and for each $K \in \Gamma$ we use the function *PlantColour (pattern K, colour c, temperature t)* repeatedly, each time with a different temperature t (the value of t is decreasing), and we memorize a new list of patterns Δ. The list Δ contains patterns in which we placed the current colour c the most times. If we find a new pattern in which we used the colour c the same number of times as in patterns in Δ, we add this pattern into Δ. If we find a new pattern in which we used the colour c more times than in patterns in Δ, we clear the list Δ and add this new pattern in it. This means that each patterns K from the list Δ has the same (and the smallest found) value of the function *NonColoured (pattern K)*.

The procedure for colouring of a pattern K with a current colour c is based on the following method, where we set a pattern L as a copy of K. In k iterations, we are looking for a vertex v which we colour with c. In each iteration we choose an arbitrary vertex w of L, we get $f(w) = \textit{FreeForColour (colour c, pattern L after colouring of } w \textit{ with } c)$, and we accept this vertex w (and set $v = w$) with a probability $p = e^{\frac{f(v)-f(w)}{t}}$ (see SA algorithm), where $f(v) = \textit{FreeForColour}(c, L \text{ after colouring of } v \text{ with } c)$. Note that in the first iteration we accept w automatically. After k iterations we colour the vertex v with c and we repeat this method until *FreeForColour (colour c, pattern L)* $= 0$. When *FreeForColour (colour c, pattern L)* $= 0$, we return L to Algorithm 1.

Remark

Let us note that we used this algorithm also for other shapes of pattern, e.g., rectangular patterns on 16×24, 24×32 vertices and also a square pattern on 32×32 vertices. However, the square pattern on 24×24 vertices gave the best solutions.

```
1  2  1  3  1  2  1 10  1  4  1  9  1  2  1  3  1  2  1  5  1  4  1 14
7  1  5  1  6  1  3  1  2  1  3  1  8  1  5  1  4  1  3  1  2  1  3  1
1  3  1  2  1  4  1  7  1  5  1  2  1  3  1  2  1 11  1  6  1 10  1  2
4  1  9  1  3  1  2  1  3  1  6  1  4  1  7  1  3  1  2  1  3  1  5  1
1  2  1 15  1  5  1 11  1  2  1  3  1  2  1 17  1  5  1  4  1  2  1  3
6  1  3  1  2  1  3  1  4  1 14  1  5  1  3  1  2  1  3  1  7  1  8  1
1  5  1  4  1 16  1  2  1  3  1  2  1 10  1  4  1 13  1  2  1  3  1  2
3  1  2  1  3  1  6  1  5  1  7  1  3  1  2  1  3  1  9  1  5  1  4  1
1  7  1 10  1  2  1  3  1  2  1  4  1  6  1  5  1  2  1  3  1  2  1 11
2  1  3  1  5  1  4  1  8  1  3  1  2  1  3  1  7  1  4  1  6  1  3  1
1  4  1  2  1  3  1  2  1  9  1  5  1 11  1  2  1  3  1  2  1 12  1  5
3  1  6  1 13  1  7  1  3  1  2  1  3  1  4  1  8  1  5  1  3  1  2  1
1  2  1  3  1  2  1  5  1  4  1 15  1  2  1  3  1  2  1 10  1  4  1  9
8  1  5  1  4  1  3  1  2  1  3  1  7  1  5  1  6  1  3  1  2  1  3  1
1  3  1  2  1 11  1  6  1 10  1  2  1  3  1  2  1  4  1  7  1  5  1  2
4  1  7  1  3  1  2  1  3  1  5  1  4  1  9  1  3  1  2  1  3  1  6  1
1  2  1 17  1  5  1  4  1  2  1  3  1  2  1 14  1  5  1 11  1  2  1  3
5  1  3  1  2  1  3  1  7  1  8  1  6  1  3  1  2  1  3  1  4  1 15  1
1 10  1  4  1  9  1  2  1  3  1  2  1  5  1  4  1 16  1  2  1  3  1  2
3  1  2  1  3  1 12  1  5  1  4  1  3  1  2  1  3  1  6  1  5  1  7  1
1  6  1  5  1  2  1  3  1  2  1 11  1  7  1 10  1  2  1  3  1  2  1  4
2  1  3  1  7  1  4  1  6  1  3  1  2  1  3  1  5  1  4  1  8  1  3  1
1 11  1  2  1  3  1  2  1 13  1  5  1  4  1  2  1  3  1  2  1  9  1  5
3  1  4  1  8  1  5  1  3  1  2  1  3  1  6  1 12  1  7  1  3  1  2  1
```

Figure 1: A pattern on 24×24 vertices.

Input: size of a pattern n
simulated annealing algorithm constants t_{min}, t_{max}, q ($t_{max} > t_{min} > 0$, $q \in (0, 1)$

Output: Γ - list of the best found patterns

```
// initialization step
Γ := empty list of the best found patterns;
pattern K := new pattern; // a null squared matrix of order n
add K to Γ;
colour c := 1; // c is a current colour
// end of the initialization step

// value of function nonColoured(K) is the same for all patterns K in Γ
// L is the first pattern in Γ
while nonColoured(L) > 0 do
    Δ := empty list of patterns;
    int p := nonColoured(L);
    int m := 0; // m is a maximum number of coloured vertices with current colour c
    foreach K in Γ do
        t := t_max;
        while t > t_min do
            T := plantColour(K, c, t); // colour vertices of K with c - see Algorithm 2
            int a := p - nonColoured(T); // a is number of vertices coloured with c in T
            if a >= m then
                // better possibility for colour c than patterns in Δ
                if a > m then
                    Δ := empty list of patterns; // Δ is cleared
                    m := a; // a is a new maximum number of vertices coloured with c
                end
                if T ∉ Δ then
                    add T to Δ;
                end
            end
            t := t * q; // decrease temperature t
        end
    end
    Γ := Δ; // put all the best found patterns from Δ to Γ
    c := c + 1; // set a new colour c
end

return Γ; // return a list of the best found patterns
```

Algorithm 1: Main algorithm for determining a pattern of packing colouring of the square lattice

Input: the original pattern K
the current colour c
the simulated annealing parameter t (temperature)
Auxiliary: the simulated annealing algorithm constant k (number of iterations)
Output: pattern L - the original pattern K coloured with c

L := K; // L is a copy of K
// while there is at least one vertex in L which can be coloured with c
while $freeForColour(c, L) > 0$ **do**
    v := null; // v is a vertex colourable with c, at the beginning we have no such a vertex
    // $f(v)$ is a number of vertices we lose for colouring with c while we colour v with c
    // at the beginning $f(v)$ is set as a maximum int value, because it will be minimized
    int $f(v)$:= max int value;
    for $i = 1$ *to* k **do**
        w := randomly chosen vertex from L which can be coloured with c;
        // $f(w)$ is a number of vertices we lose for colouring with c while we colour w with c
        int $f(w)$:= $freeForColour(c, L) - freeForColour(c, L\ with\ coloured\ w)$;
        // SA accepts possibility with probability $p = e^{\frac{f(v)-f(w)}{t}}$
        // if $f(v) > f(w)$ then $p > 1$ and better possibility is accepted automatically
        if $random(0, 1) < e^{\frac{f(v)-f(w)}{t}}$ **then**
            $f(v)$:= $f(w)$;
            v := w;
        end
    end
    colour vertex v with c;
end
return L;

Algorithm 2: Function $plantColour(K, c, t)$

References

[1] J.A. Bondy, U.S.R. Murty, Graph Theory with Applications.
Macmillan, London and Elsevier (1976).

[2] J. Fiala, B. Lidický, Packing chromatic number for lattices.
Abstract in: Workshop Cycles and Colourings 2007 (I. Fabrici, M. Horňák, S. Jendrol', eds.), IM Preprint, series A, No. 8/2007.

[3] W. Goddard, S.M.Hedetniemi, S.T.Hedetniemi, J.M.Harris, D.F.Rall, Broadcast chromatic numbers of Graphs.
Ars Combin. 43 (1996), 149-157.

[4] S. Kirkpatrick, C.D. Gerlatt Jr., M.P. Vecchi, Optimization by Simulated Annealing.
Science 220 (1983), 671-680.

[5] P.J.M. van Laarhoven, E.H.L. Aarts, Simulated Annealing: Theory and Applications.
Reidel, Dordrecht, Holland (1987).

[6] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equations of State Calculations by Fast Computing Machines.
J. Chem. Phys. 21 (1953), 1087-1092.

[7] M. Pincus, A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems.
Oper. Res. 18 (1970), 1225-1228.

[8] A. Schwenk, personal communication, 2002.