On the number of 2-packings in a connected graph

Konstanty Junosza-Szaniawski, Paweł Rzążewski*
{k.szaniawski, p.rzazewski}@mini.pw.edu.pl

Warsaw University of Technology
Faculty of Mathematics and Information Science
Pl. Politechniki 1 , 00-661 Warsaw, Poland

Abstract

By a 2-packing in a graph we mean a subset of its vertex set, in which all the vertices are in distance at least 3 from each other.

The question about the maximum number of 2-packings in a graph is strongly related to the problem of $L(2,1)$-labeling of graphs (see [2]).

In this paper we find new asymptotic upper and lower bounds on the maximum number of 2-packings in a connected graph on n vertices. The bounds are $O(1.5399..^n)$ and $\Omega(1.4970..^n)$, respectively.

Moreover, we present a lower bound on the number of k-element 2-packings in a connected graph, which is $\max(\binom{n-2k+2}{k}, (k+1)\binom{\lfloor\frac{n-1}{2}\rfloor}{k})$.

1 Introduction

The direction of research that we follow in this paper is related to several famous results in extremal graph theory like, for example, the upper bound for the number of maximal cliques in a graph by Moon and Moser [6] or the upper bound for the number of maximal independent sets in a tree by Sagan [7]. These bounds are widely used in complexity analysis of algorithms.

*The authors have been working on this paper when visiting KAM at Charles University in Prague in October 2010.

Besides the bounds, also graphs achieving them are subject of interest. In both cases mentioned above such graphs are well characterized.

In this paper we study another problem of extremal graph theory, which is the problem of finding the maximum number of 2-packings in a graph. A subset of vertices in a graph is called a 2-packing if no two vertices of this subset have a common neighbor. We can think of 2-packings in a graph G as of independent sets in the graph G^2. Some authors call 2-packings 2-stable sets (see [1]).

Notice that no 2-packing in a connected graph has more than $\frac{n}{2}$ vertices, since each of them must have at least one neighbor and no two of them may share the same neighbor.

The question how large the number of 2-packings in a graph can be arises naturally in the analysis of an exact algorithm for finding an $L(2,1)$-labeling in a graph (for the definition of this concept see [2]) by Havet *et al.* [3]. The bound on the number of 2-packings is crucial in discussion of the computational complexity of this algorithm. The authors showed in [3] that the number of k-element 2-packings in a connected graph on n vertices (denoted by $u_k(n)$) does not exceed $\binom{n/2}{k}2^n$. From this inequality we can derive a bound on the number of all 2-packings in a connected graph on n vertices (denoted by $u(n)$), which is $\sum_{k=0}^{n}\binom{n/2}{k}2^n = O(1.7320..^n)$. In [5] we improved these bounds by showing that $u_k \leq \binom{n-k+1}{k}$ and $u(n) \leq \sum_{k=0}^{n}\binom{n-k+1}{k} = O(1.6180..^n)$.

If we drop the connectivity restriction, the question about the maximum number of 2-packings becomes trivial – in a graph with no edges the number of 2-packings is equal to 2^n, where n denotes the number of vertices in this graph.

In Section 3 of this paper, we present an algorithm for generating all 2-packings in a connected graph. From the analysis of this algorithm it follows a better bound on $u(n)$, which is $O(1.5399..^n)$.

In Section 4 a lower bound on $u(n)$ is established, by constructing a graph with $\Theta(1.4970..^n)$ 2-packings.

Finally, in Section 5, we present a lower bound on $u_k(n)$, which is $\max(\binom{n-2k+2}{k}, (k+1)\binom{\lfloor\frac{n-1}{2}\rfloor}{k})$.

2 Preliminaries

For graphs H and G we write $H \subseteq G$ if and only if H is a subgraph of G. By $G[X]$ (G is a graph, $X \subseteq V(G)$) we denote the subgraph of G induced by the vertices in X.

Let $N_G(v) = \{u : uv \in E(G)\}$ denote the *neighborhood* of a vertex v in a graph G. The set $N_G[v] = N_G(v) \cup \{v\}$ denotes the *closed neighborhood* of v. By the *neighborhood* of a set X of vertices in G we mean the set $N_G(X) = \bigcup_{v \in X} N_G(v)$ and by the *closed neighborhood* of X – the set $N_G[X] = N_G(X) \cup X$.

Let $dist_G(x, y)$ denote the *distance* between vertices x and y in a graph G, which is the length of the shortest x-y-path. A set $X \subseteq V(G)$ is a 2-*packing* in G if and only if $\forall x, y \in X\ dist_G(x, y) > 2$.

For a tree T let $L(T)$ denote the set of leaves in T (i.e. vertices with degree equal to 1).

We write $f(n) = O^*(g(n))$ if there exists a polynomial $p(n)$ such that $f(n) \leq p(n) \cdot g(n)$ for all n greater than some n_0.

3 Algorithm for generating all 2-packings in a connected graph

In this section we present a recursive algorithm for generating all 2-packings in a connected graph G. The analysis of this algorithm gives an improved upper bound on the number of 2-packings.

Notice that removing an edge from a graph does not reduce the number of 2-packings. Hence a connected graph having the maximum number of 2-packings is a tree.

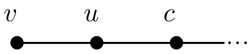
If the graph has at most 2 vertices, all 2-packings can be easily enumerated. For graphs with more vertices, the algorithms proceeds to its main part.

In the beginning, the algorithm finds T – a spanning tree of G and then P, which is the longest path in T (the longest path can be found by the algorithm described in [8]). Let v be an end-vertex of the path P, u its neighbor on P, and c a neighbor of u on P other that v (the third vertex on P). Depending on the structure of T near the end-vertex of P, the algorithm chooses one of the branching rules described below to construct a 2-packing. Let us denote this constructed 2-packing by S.

3.1 Branching Rules

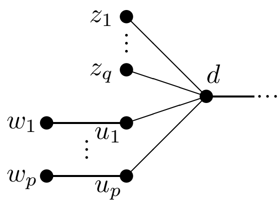
Branching rule A

This rule can be applied if $\deg_T u = 2$ and $\deg_T c \leq 2$.



The algorithm recursively generates 2-packings not containing v and 2-packings containing v. If v is not included in generated 2-packing S, we can delete v from the graph and proceed with the graph $G - v$. On the other hand, if v is included in S, none of the vertices u and c can belong to S. Thus we can proceed with the graph $G - \{v, u, c\}$.

Branching rule $B^{p,q}$ (for any $p, q \in \mathbb{N}$ such that $p+q \geq 2$) can be applied if $\deg_T u > 2$ or $\deg_T c > 2$. Let d denote a neighbor of c with at least two neighbors in $L(T)$, if there exists such a vertex. Otherwise let $d = c$.



Let $Z = \{z_1, .., z_q\}$ and $U = \{u_1, .., u_p\}$ be the sets of neighbors of the vertex d, having degree 1 and 2 in the tree T, respectively. Let $W = \{w_1, .., w_p\} = L(T) \cap N(U)$. Notice that p or q might be equal to zero, but not at the same time. In case $p = 1$, $q = 0$ we can apply branching rule A, so we can assume that $p + q \geq 2$.

In this case we branch on all the vertices in $Z \cup W \cup U$ simultaneously. If none of these vertices is included in S, we can delete them all from the graph and proceed with the graph $G - (Z \cup W \cup U)$. In other case we can choose any nonempty 2-packing $\widehat{S}$ in $G[Z \cup W \cup U \cup \{d\}]$ not containing d, and include its vertices in S. Notice that the vertex d is in distance at most 2 from any of the vertices in $Z \cup W \cup U$, so it cannot belong to S. Thus we can proceed with the graph $G - (Z \cup W \cup U \cup \{d\})$.

There are $p2^{p-1} + (q+1)2^p - 1$ nonempty 2-packings in $G[Z \cup W \cup U \cup \{d\}]$ not containing d.

To decide which branching rule should be applied, the algorithm first checks if $\deg_T u = 2$ and $\deg_T c \leq 2$. In this case branching rule A is chosen. In the other case the algorithm checks if there is a neighbor x of the vertex c, adjacent to more than one leaf in T. In such case branching rule $B^{0, \deg_T x - 1}$ is applied for $d = x$. If there is no such neighbor then branching rule $B^{p,q}$ is applied for $d = c$, and proper p, q.

Notice that application of these branching rules may generate some sets that are not 2-packings in T. For example in branching rule $B^{p,q}$ when any neighbor of c is added to S, then the neighbor of c in $V(P) \setminus (Z \cup U)$ can not belong to generated 2-packing, but the algorithm does not make use of this information. Moreover, not all 2-packings in T are also 2-packings in G. Thus after generating each set we should check if it is a 2-packing in G and delete it if it is not. Checking if a given set is a 2-packing can be performed in polynomial time.

Here is the pseudocode of the algorithm **Generate-2-packings**. Notice that we have to keep the initial graph G all the time to check if a generated set is a 2-packing in G. Hence in recursive calls we modify the copy of G, which is $\widehat{G}$.

Algorithm 1: Generate-2-packings

Call : Generate-2-packings(G, $\widehat{G}$, S)

Arguments: Connected graph G, Connected graph $\widehat{G} \subseteq G$, Set $S \subseteq V(G) \setminus V(\widehat{G})$

if $|V(\widehat{G})| \leq 2$ **then**
    foreach $v \in V(\widehat{G})$ **do**
        if $S \cup \{v\}$ *is a* 2*-packing in* G **then**
            Print $S \cup \{v\}$
    if S *is a* 2*-packing in* G **then**
        Print S
    return
$T \leftarrow$ spanning tree of $\widehat{G}$
$P \leftarrow$ longest path in T
$v \leftarrow$ end-vertex of P
$u \leftarrow$ neighbor of v on P
$c \leftarrow$ neighbor of u on P other than v (next vertex on P)
$U \leftarrow \{x \in N_T(c) \setminus V(P) : \deg_T x \geq 2\} \cup \{u\}$
$Z \leftarrow \{x \in N_T(c) : \deg_T x = 1\}$
$W \leftarrow \{x \in N_T(U) : \deg_T x = 1\}$
if $\deg_T u = 2$ *and* $\deg_T c \leq 2$ **then**
    Generate-2-packings$(G, \widehat{G} - v, S)$
    Generate-2-packings$(G, \widehat{G} - \{v, u, c\}, S \cup \{v\})$
else if *there exists* $x \in U$ *such that* $\deg_T x > 2$ **then**
    foreach $y \in N_T(x) \setminus \{c\}$ **do**
        Generate-2-packings$(G, \widehat{G} - (N_T[x] \setminus \{c\}), S \cup \{y\})$
    Generate-2-packings$(G, \widehat{G} - (N_T(x) \setminus \{c\}), S)$
else
    foreach $\widehat{S}$ *– nonempty* 2*-packing in* $G[U \cup W \cup Z \cup \{c\}]$ *not containing* c **do**
        Generate-2-packings$(G, \widehat{G} - (U \cup W \cup Z \cup \{c\}), S \cup \widehat{S})$
    Generate-2-packings$(G, \widehat{G} - (U \cup W \cup Z), S)$

In order to generate all 2-packings in a graph G we call the algorithm: **Generate-2-packings**$(G, G, \emptyset)$.

Lemma 1. *The algorithm call **Generate-2-packings**$(G, G, \emptyset)$ generates each 2-packing in G exactly once.*

Proof. Let G be a connected graph, $\widehat{G}$ a connected subgraph of G and $S \subseteq V(G) \setminus V(\widehat{G})$. We will prove by induction on the number of vertices in $\widehat{G}$ that the algorithm call **Generate-2-packings**$(G, \widehat{G}, S)$ generates exactly once each set A, such that A is a 2-packing in G and $S \subseteq A \subseteq S \cup V(\widehat{G})$

If $|V(\widehat{G})| \leq 2$ the algorithm obviously gives the correct result (lines 1-7).

Assume that for any connected graph G, connected graph $\widehat{G} \subseteq G$ with less than k vertices and any $S \subseteq V(G) \setminus V(\widehat{G})$ the algorithm call **Generate-2-packings**$(G, \widehat{G}, S)$ generates every 2-packing A in G such that $S \subseteq A \subseteq S \cup V(\widehat{G})$ exactly once.

Notice that if S is not a 2-packing in G, there is no 2-packing in G containing S. In such case the algorithm **Generate-2-packings** returns no set, since before returning any set it checks if it is a 2-packing in G (lines 3, 5). Hence from now we can assume that S is a 2-packing.

Consider a connected graph G, connected graph $\widehat{G} \subseteq G$ with k vertices and $S \subseteq V(G) \setminus V(\widehat{G})$. Let v, u, c, U, Z, W be defined as in the algorithm.

If $deg_T u = 2$ and $deg_T c \leq 2$, we can divide the set of all 2-packings A in G, such that $S \subseteq A \subseteq S \cup V(\widehat{G})$, to those not containing v and those containing v. Graphs $G - v$ and $\widehat{G} - \{v, u, c\}$ are connected so by inductive assumption all the 2-packings not containing v are generated by recursive call in line 17, while ones containing v are generated in recursive call in line 18.

If there exists $x \in U$ such that $deg_T x > 2$ then the set of all 2-packings A such that $S \subseteq A \subseteq S \cup V(\widehat{G})$, can be divided into two sets: those containing any of vertices in $N_T(x) \setminus \{c\}$ and those not containing any of them. Notice that at most one vertex from $N_T(x) \setminus \{c\}$ can belong to each 2-packing. Graphs $\widehat{G} - (N_T[x] \setminus \{c\})$ and $\widehat{G} - (N_T(x) \setminus \{c\})$ are connected so by inductive assumption all 2-packings containing a vertex from $N_T(x) \setminus \{c\}$ are generated in the recursive calls in lines 20-21 and all the 2-packings not containing vertices from $N_T(x) \setminus \{c\}$ are generated in the recursive call in line 22.

In all remaining cases we can divide the set of all the 2-packings A such that $S \subseteq A \subseteq S \cup V(\widehat{G})$ into sets: $\mathcal{A}_{\widehat{S}} = \{A : A$ is 2-packing such that $S \subseteq A \subseteq S \cup V(\widehat{G})$ and $A \cap (U \cup W \cup Z) = \widehat{S}\}$ where $\widehat{S}$ is a 2-packing in $\widehat{G}[U \cup W \cup Z \cup \{c\}]$ not containing c (notice that a subset of a 2-packing is a 2-packing). Graphs $\widehat{G} - (U \cup W \cup Z)$ and $\widehat{G} - (U \cup W \cup Z \cup \{c\})$ are connected so by inductive assumption for non-empty 2-packing $\widehat{S} \subseteq U \cup W \cup Z$ 2-packings from $\mathcal{A}_{\widehat{S}}$ are generated in the recursive calls in lines 25 and 2-packings from $\mathcal{A}_{\emptyset}$ are generated in the recursive call in line 26.

Since those are all possible cases, the proof is finished. $\square$

3.2 Time complexity analysis

In time complexity analysis we will use a widely known technique first presented by Oliver Kullmann [4]. We start this section with a short introduction to this method.

Let T be a rooted tree, $D\colon V(T) \to \mathcal{P}(V(T))$ its *descendant function*, i.e.
$D(v) = \{u\colon u \text{ is a child of } v \text{ in } T\}$. A function $\mu\colon V(T) \to \mathbb{R}^+ \cup \{0\}$ is a *measure* if and only if $\mu(v) > \mu(u)$ for any $v \in V(T) \setminus L(T)$ and $u \in D(v)$. For a fixed rooted tree T and a measure μ let τ_v (where $v \in V(T) \setminus L(T)$) denote the positive root of the equation

$$1 = \sum_{u \in D(v)} \tau^{-(\mu(v)-\mu(u))} \tag{1}$$

Notice that this equation has a unique positive solution. Moreover, it is easy to observe that $\tau_v > 1$.

Lemma 2 (Kullmann [4]). *For a rooted tree T and a measure μ, if there exists a real number τ_0 such that the inequality $\tau_v \leq \tau_0$ holds for every $v \in V(T) \setminus L(T)$ then $|L(T)| = O\left(\tau_0^{\mu(r)}\right)$ where r is the root of the tree T.*

We will apply this method to prove the main theorem of our paper:

Theorem 1. *The maximum number of 2-packings in a connected graph on n vertices does not exceed $O(1.5399..^n)$. The complexity of the algorithm **Generate-2-packings** is bounded by $O^*(1.5399..^n)$.*

Proof. Let $\mathcal{T}$ be the recursive calls tree of the algorithm **Generate-2-packings** for a graph G. Notice that each leaf in $\mathcal{T}$ corresponds to at most one 2-packing (some recursive calls of **Generate-2-packings** may finish with no 2-packing generated, see lines 3, 5), while inner nodes of $\mathcal{T}$ correspond to pairs (H, S), where H is a subgraph of G and $S \subseteq V(G)$. We will use Lemma 2 for $\mu(H) = |V(H)|$ to estimate the value of $|L(\mathcal{T})|$, which is the bound on the number of 2-packings in G. Now let us show that

$\tau_{(H,S)} \leq \tau_0$ for every pair (H, S) corresponding to some inner node of $\mathcal{T}$, where $\tau_0 = 1.5399..$ is the positive root of the equation

$$\tau^7 = \tau + 19 \tag{2}$$

This will show that $|L(\mathcal{T})| = O(\tau_0^n)$.

Notice that $\tau_{(H,S)}$ does not depend on S, but only on the branching rule that is applied to subgraph H. Hence we will denote $\tau_{(H,S)}$ by τ_A if branching rule A is applied to H by de algorithm, and by $\tau_{p,q}$ if branching rule $B^{p,q}$ is applied to H. The number τ_A is the positive root of the equation:

$$1 = \tau^{-1} + \tau^{-3} \tag{3}$$

$>$From this equation we obtain the value of $\tau_A = 1.4650.. < 1.5399.. = \tau_0$.

The number $\tau_{p,q}$ is the positive root of the equation:

$$1 = \tau^{-(2p+q)} + (p2^{p-1} + (q+1)2^p - 1)\tau^{-(2p+q+1)} \tag{4}$$

Notice that $\tau_0 = \tau_{3,0}$.

We can transform the above equation to a form:

$$\tau^{2p+q+1} = \tau + (p2^{p-1} + (q+1)2^p - 1) \tag{5}$$

Let $L^{p,q}(\tau) = \tau^{2p+q+1}$ and $R^{p,q}(\tau) = \tau + (p2^{p-1} + (q+1)2^p - 1)$.

We observe that $\tau_{p,q} \leq \tau_{p+1,q-2}$ for all n, p and $q \geq 2$.

Let us compare the equations defining $\tau_{p,q}$ and $\tau_{p+1,q-2}$:

$\tau_{p,q}$: $\tau^{2p+q+1} = \tau + (p2^{p-1} + (q+1)2^p - 1)$

$\tau_{p+1,q-2}$: $\tau^{2p+q+1} = \tau + ((p+1)2^p + (q-1)2^{p+1} - 1)$

Notice that $L^{p,q}(\tau) = L^{p+1,q-2}(\tau)$ and $R^{p,q}(\tau) < R^{p+1,q-2}(\tau)$, so $\tau_{p,q} < \tau_{p+1,q-2}$.

Thanks to this observation we can consider only the cases where $q = 0$ or $q = 1$, (all the others values of $\tau_{p,q}$ can be bounded by the ones with reduced value of q).

The table below presents values of $\tau_{p,q}$ for small p, q.

$\tau_{p,q}$		q	
		0	1
p	1	X	1.5337..
	2	1.5354..	1.5238..
	3	**1.5399..**	1.5201..

Now let us show that $\tau_{p,0} < \tau_{3,0} = \tau_0$ for all $p > 3$. To prove it, it is enough to show that:

$$L^{p,0}(\tau_0) > R^{p,0}(\tau_0) \tag{6}$$

and

$$\frac{d}{d\tau}L^{p,0}(\tau) > \frac{d}{d\tau}R^{p,0}(\tau) \text{ for } \tau \geq \tau_0 \tag{7}$$

Conditions (6) and (7) imply that $L^{p,0}(\tau) > R^{p,0}(\tau)$ for $\tau \geq \tau_0$. Hence the positive root of the equation (5) must be smaller than τ_0.

Notice that the inequality (6) is equivalent to:

$$\tau_0 \left(\frac{\tau_0^2}{2}\right)^p > \frac{p+2}{2} + \frac{\tau_0 - 1}{2^p}$$

Let $l(p) = \tau_0 \left(\frac{\tau_0^2}{2}\right)^p$ and $r(p) = \frac{p+2}{2} + \frac{\tau_0-1}{2^p}$.

To prove that $l(p) > r(p)$, it is enough to show that $l(4) = 3.0430.. > 3.0337.. = r(4)$ and $l'(p) > r'(p)$ for $p > 3$.
$l'(p) = (0.2622..) \cdot (1.1856..)^p > 0.5$
$r'(p) = 0.5 - (0.3742..) \cdot 2^{-p} < 0.5$

Hence $l(r) > r(l)$ and therefore $L^{p,0}(\tau_0) > R^{p,0}(\tau_0)$, which ends the proof of the inequality (6).

To prove the inequality (7), it is enough to show that $\frac{d}{d\tau}L^{p,0}(\tau) = (2p+1)\tau^{2p} > 1 = \frac{d}{d\tau}R^{p,0}(\tau)$.

Hence $\tau_{p,0} < \tau_0$ for all $p > 3$.

In analogous way we can show that $\tau_{p,1} < \tau_{1,1}$ for all $p > 2$.

Since $\tau_{1,1} = 1.5337.. < 1.5399.. = \tau_0$, the assumptions of Lemma 2 are fulfilled. Hence the number of leaves in $\mathcal{T}$ and therefore the number of 2-packings in G does not exceed $O(\tau_0^n) = O(1.5399..^n)$.

All the local operations (i.e. finding a spanning tree, finding the longest path in a tree, deleting vertices, checking if a set is a 2-packing etc.) are performed in polynomial time, hence the total computational complexity of the algorithm is $O^*(1.5399..^n)$. □

The idea presented in this section can be developed to obtain better bound on the value of $u(n)$. To do so, one have to consider four or more consecutive vertices on the end of the longest path in a spanning tree (instead of three). However, this leads to many technical calculations.

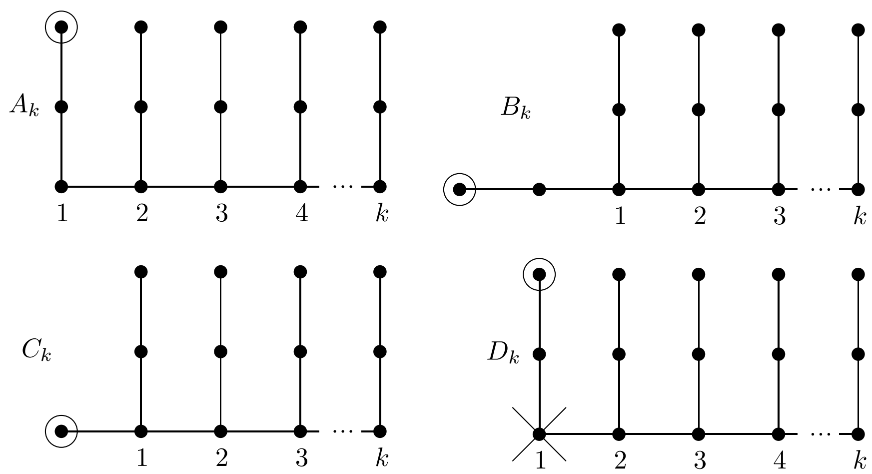
4 Lower bound for the maximum number of 2-packings

In this section we present a lower bound on the maximum number of 2-packings in a connected graph.

Theorem 2. *The maximum number of* 2*-packings in a connected graph is bounded from below by* $\Omega(1.4970..^n)$.

Proof. We will prove the theorem by showing a graph with $\Theta(1.4970..^n)$ 2-packings.

Let us consider the following graphs:



Let a_k, b_k and c_k denote the number of 2-packings in graphs A_k, B_k and C_k, respectively. Let d_k denote the number of such 2-packings in graph D_k, which do not contain the crossed out vertex.

Considering the number of 2-packings not containing and 2-packings containing marked vertices, we obtain the following system of recursions:

$$\begin{cases} a_k = b_{k-1} + a_{k-1} \\ b_k = c_k + d_k \\ c_k = a_k + 2d_{k-1} \\ d_k = 2a_{k-1} + d_{k-1} \end{cases}$$

Solving this system we obtain the result $a_k = \Theta(3.3553..^k)$. Since $k = n/3$, the graph A_k contains $a_k = \Theta(3.3553..^{n/3}) = \Theta(1.4970..^n)$ 2-packings. This is also a lower bound on the maximum number of 2-packings in a connected graph. $\square$

5 Lower bound for the maximum number of k-element 2-packings

In this section we present a lower bound for the value of $u_k(n)$. To show it, let us prove the following lemmas.

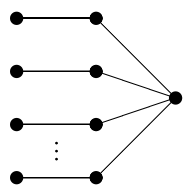
Lemma 3. *The maximum number of k-element 2-packings in a connected graph on n vertices is at least $\binom{n-2k+2}{k}$.*

Proof. Let us consider a graph P_n, which is a path on n vertices. Let $p_k(n)$ be the number of k-element 2-packings in P_n. Notice that $p_k(n)$ is equal to the number of binary sequences of length n with exactly k ones, such that there are at least two zeros between every pair of ones. Observe that there are exactly $\binom{n-2k+2}{k}$ such sequences.

Hence the maximum number of k-element 2-packings in a connected graph on n vertices is at least $\binom{n-2k+2}{k}$. $\square$

Lemma 4. *The maximum number of k-element 2-packings in a connected graph on n vertices is at least $(k+1)\binom{\lfloor \frac{n-1}{2} \rfloor}{k}$.*

Proof. For an odd number n by S_n we denote a graph obtained from a matching of size $(n-1)/2$ by adding one vertex and joining it with exactly one vertex from every edge of the matching.



Let $s_k(n)$ denote the number of k-element 2-packings in S_n. It is easy to observe that

$$s_k(n) = \begin{cases} n \text{ for } k = 1 \\ (k+1)\binom{\frac{n-1}{2}}{k} \text{ for } k \neq 1 \end{cases}$$

Hence the maximum number of k-element 2-packings in a connected graph on n vertices is at least $(k+1)\binom{\lfloor \frac{n-1}{2} \rfloor}{k}$. □

We observe that $p_2(n) > s_2(n)$ (for $n > 5$), while $s_{\frac{n-1}{2}}(n) > p_{\frac{n-1}{2}}(n) = 0$ (for $n > 7$).

Corollary 1. *The number of k-element all 2-packings in a connected graph on n vertices can be bounded from below by* $\max(\binom{n-2k+2}{k}, (k+1)\binom{\lfloor \frac{n-1}{2} \rfloor}{k})$.

6 Open problems

In this paper we showed that the value of $u(n)$ (the maximum number of all 2-packings in a connected graph) is between $\Omega(1.4970..^n)$ and $O(1.5399..^n)$. The question what is the value of $u(n)$ in terms of Θ notation remains open.

The other problem that remains open is what is the exact value of $u_k(n)$, the maximum number of k-element 2-packings in a connected graph.

It is also interesting to characterize graphs with $u(n)$ 2-packings and graphs with $u_k(n)$ k-element 2-packings.

Acknowledgement: The authors are grateful to Professor Zbigniew Lonc for his comments and suggestions that helped improving this paper.

References

[1] Chang G.J., Kuo, D. The $L(2,1)$-labeling problem on graphs, SIAM Journal on Discrete Mathematics 9 (1996) 309–316.

[2] Griggs J.R., Yeh R.K. Labeling graphs with a condition on distance 2, SIAM Journal on Discrete Mathematics 5 (1992) 586-595.

[3] Havet, F., Klazar, M., Kratochvíl, J., Kratsch, D., Liedloff, M. Exact Algorithms for $L(2,1)$-Labeling of Graphs, Algorithmica DOI 10.1007/s00453-009-9302-7

[4] Kullmann O., New methods for 3-SAT decision and worst-case analysis, Theoretical Computer Science 223 (1-2) (1999) 1-72.

[5] Junosza-Szaniawski K., Rzążewski P., On improved algorithms for $L(2,1)$-labeling of graphs, IWOCA 2010, Lecture Notes in Computer Science 6460 (2011) 34–37.

[6] Moon, J. W., Moser, L. On cliques in graphs, Israel Journal of Mathematics 3 (1965) 23–28.

[7] Sagan B.E. A note on independent sets in trees, SIAM Journal on Discrete Mathematics 1 (1988) 105-108.

[8] Wu, B.Y., Chao, K.-M. Spanning Trees and Optimization Problems, Chapman & Hall/CRC Press, USA (ISBN 1-58488-436-3) (2004).