Decycling Bipartite Graphs

Alain Hertz

1Department of Mathematics and Industrial Engineering, Polytechnique Montreal, Montreal

Abstract. Let \( G = (V, E) \) be a graph and let \( S \subseteq V \) be a subset of its vertices. If the subgraph of \( G \) induced by \( V \setminus S \) is acyclic, then \( S \) is said to be a decycling set of \( G \). The size of a smallest decycling set of \( G \) is called the decycling number of \( G \). Determining the decycling number of a graph \( G \) is NP-hard, even if \( G \) is bipartite. We describe a tabu search procedure that generates decycling sets of small size for arbitrary bipartite graphs. Tests on challenging families of graphs show that the proposed algorithm improves many best-known solutions, thus closing or narrowing the gap to the best-known lower bounds.

1 Introduction

We consider the problem of eliminating all cycles from a graph by means of deletion of vertices. Determining a decycling set (i.e., a set of vertices whose removal eliminates all cycles) of minimum size is an NP-hard problem, even for bipartite graphs. We describe a heuristic procedure that produces small decycling sets in arbitrary bipartite graphs. Experiments performed on challenging families of bipartite graphs will demonstrate the efficiency of the proposed procedure. We first need to fix some notation to describe the problem more precisely.

Let \( G = (V, E) \) be a simple undirected graph with vertex set \( V \) and edge set \( E \). The order of \( G \) is its number \( |V| \) of vertices. For a subset \( W \subseteq V \) of vertices, we write \( E(W) \) for the set of edges of \( G \) with both endpoints in \( W \), while \( G[W] = (W, E(W)) \) is the subgraph of \( G \) induced by \( W \). Also, we write \( N_G(v) \) for the set of vertices adjacent to \( v \) in \( G \) and \( d_G(v) = |N_G(v)| \) is the degree of \( v \). A stable set of \( G \) is a set of pairwise non-adjacent vertices, while a clique of \( G \) is a set of pairwise adjacent vertices. A forest is an acyclic graph. A graph \( G = (V, E) \) is bipartite if there is a partition \((V_1, V_2)\) of \( V \) so that all edges of \( E \) have one endpoint in \( V_1 \) and the other in \( V_2 \), and we also write \( G = (V_1, V_2, E) \). For other basic notions of graph theory that are not defined here, we refer to Diestel [10].
Let $W$ be a subset of vertices of a graph $G = (V, E)$. If the induced subgraph $G[V \setminus W]$ of $G$ is acyclic, then $W$ is said to be a decycling set of $G$. The smallest size of a decycling set of $G$ is the decycling number (or feedback vertex number) of $G$ and is denoted by $\phi(G)$. Let $\varphi(G)$ denote the largest order of an induced forest of $G = (V, E)$. Clearly, determining $\phi(G)$ is equivalent to computing $\varphi(G)$ since a subset $F \subseteq V$ of vertices induces a forest in $G$ if and only if $V \setminus F$ is a decycling set of $G$, which implies $\phi(G) + \varphi(G) = |V|$.

Determining the decycling number of a graph has various applications, including deadlock recovery [27], synchronous distributed systems [22], VLSI design [12], constraint satisfaction and Bayesian inference [4]. Karp [17] has shown that the problem is NP-hard, even when restricted to planar graphs, bipartite graphs and perfect graphs. On the other hand, the problem is known to be solvable in polynomial time for various other families of graphs, including cubic graphs [19, 28], cocomparability graphs and convex bipartite graph [20]. A 2-approximation (i.e., a polynomial time algorithm that generates a decycling set of cardinality at most $2\phi(G)$) is described in [3], while a branch-and-cut algorithm for the exact solution of the problem is given in [8]. Local search algorithms are proposed in [8, 9, 23, 26] for determining an upper bound on the decycling number of arbitrary graphs. More results on the decycling number can be found in [5].

In a seminal paper on the topic, Beineke and Vandell [7] have bounded the decycling number of hypercubes. Improving these bounds for hypercubes was continued in [6, 13, 25]. Other families of graphs have then also been investigated, including Fibonacci cubes [11], bubble sort graphs [30] and star graphs [29]. All these graphs are bipartite and have nice topological properties that provide attractive interconnection schemes for massively parallel systems [1, 2]. The problem of avoiding deadlocks when using these topologies for exchanging data between processors can be reformulated as a decycling problem, and it turns out that computing the decycling number of these bipartite graphs is a real challenge. Indeed, there is a gap between the best-known lower and upper bounds on $\phi(G)$ for such graphs $G$ with only 120 vertices. We aim to decrease these gaps by generating decycling sets of small size.

In the next section, we describe an algorithm that determines small decycling sets in arbitrary bipartite graphs. We first design in Section 2.1 a procedure that extends an induced forest $G[F]$ of $G$ to a larger one $G[F'=F\cup S]$ with a stable set $S$ of $G$. We then show in Section 2.2 how to embed such forest extensions into a tabu search. In Section 3, we demonstrate the efficiency of our algorithm by applying it on various challenging families of bipartite graphs. The proposed procedures offer flexibility at different levels, and we will discuss this in Section 4.

## 2 The proposed algorithm

As a first observation, note that if a vertex set induces a forest in $G$, then all its subsets also induce a forest in $G$. As a particular case, assume that $F$ is the vertex set of an induced forest of $G$ and that $S$ is obtained from $F$ by removing all vertices of degree at least 2 in $G[F]$ as well as one vertex per connected component of $G[F]$ with exactly 2 vertices. Then $S$ is a stable set of $G$, and $G[F\setminus S]$ is a forest of $G$ obtained from $G[F]$ by removing all isolated vertices as well as some of its leaves. We show in Section 2.1 how to use this property the other way around. In other words, given a subset $F$ that induces a forest in $G$, we will extend it to a larger set $F'=F\cup S$ so that $G[F']$ also induces a forest in $G$, while imposing that the set $S$ of added vertices is a stable set of $G$.

Given a forest $G[F]$ of $G$, finding a set $F'$ of maximum cardinality so that $F' \supseteq F$ and $G[F']$ is also a forest of $G$ is a difficult task. Indeed, for $F = \emptyset$, the problem is equivalent to determining $\varphi(G)$. We will focus on bipartite graphs $G$ and on supersets $F'$ of $F$ obtained by adding a stable set $S$ of $G$ (i.e., $F' = F \cup S$). These assumptions make the problem a little easier. Indeed, while
finding a stable set of maximum cardinality in an arbitrary graph is an NP-hard problem, the
famous Hungarian method [18] can be used when restricted to bipartite graphs. Hence, given a
vertex set $F$ that induces a forest $G[F]$ in a bipartite graph $G = (V, E)$, the following problem
can be solved in polynomial time: find a stable set $S \subseteq V \setminus F$ of maximum cardinality such that
every vertex in $S$ is adjacent to at most one vertex in $F$. With such a stable set $S$ we deduce that
$G[F' = F \cup S]$ is a forest of $G$. It is obtained by adding leaves and isolated vertices to $G[F]$. We
will go one step further by considering stable sets $S$ of $G$ that possibly contain vertices with more
than one neighbor in $F$.

The next subsection gives more details on the proposed procedure that extends an induced
forest $G[F]$ of $G$ to a larger one $G[F' = F \cup S]$ with a stable set $S$ of $G$. The next step will be
to embed these forest extensions into a tabu search. Roughly speaking, we are going to explore the
space that contains all the induced forests of $G$ that have fixed numbers of vertices in $X$ and in $Y$,
and for each considered forest, we will try to extend it by adding a stable set of $G$. This will be
explained in detail in Section 2.2.

\subsection{2.1 Forest extensions with stable sets}

Let $F$ be the vertex set of an induced forest of a graph $G = (V, E)$. Assume $G[F]$ has $r$
connected components with vertex sets $C_1, C_2, \ldots, C_r$. We denote by $V_F$ the set of vertices $v \in V \setminus F$
with at most one neighbor in every $C_i$:

$$V_F = \{v \in V \setminus F \text{ with } |N_G(v) \cap C_i| \leq 1 \text{ for } i = 1, 2, \ldots, r\}.$$ 

Clearly, adding a vertex $v \notin V_F$ to $F$ creates a cycle. We therefore extend $F$ by choosing
vertices in $V_F$. We consider the partition $(V_F^{\leq 1}, V_F^{\geq 2})$ of $V_F$ where $V_F^{\leq 1}$ is the subset of vertices
in $V_F$ with at most one neighbor in $F$ and $V_F^{\geq 2}$ is the subset of vertices in $V_F$ with at least two
neighbors in $F$:

$$V_F^{\leq 1} = \{v \in V_F \text{ with } |N_G(v) \cap F| \leq 1\}, \quad \text{and}$$

$$V_F^{\geq 2} = \{v \in V_F \text{ with } |N_G(v) \cap F| \geq 2\}.$$ 

For illustration, consider the bipartite graph $G$ at the top left of Figure 1 and assume $F = \{a, f, g, h, j, n, p, r\}$, which gives the forest $G[F]$ of $G$ at the bottom left, with four connected
components, the vertices of $F$ becoming colored in black. Vertex $b$ does not belong to $V_F$ since
its neighbors $a$ and $g$ belong to the same connected component of $G[F]$. Vertices $d$ and $k$ have
no neighbor in $F$, while vertices $c$ and $e$ have one neighbor in $F$. Also, vertices $i$, $m$, $o$ and $q$
have a neighbor in two different connected components of $G[F]$, while vertex $i$ has a neighbor in three
connected components of $G[F]$. Hence, $V_F = \{c, d, e, i, k, l, m, o, q\}$ with $V_F^{\leq 1} = \{c, d, e, k\}$ and
$V_F^{\geq 2} = \{i, l, m, o, q\}$.

We have observed that if $F$ induces a forest in $G$ and $S$ is any stable set in $V_F^{\leq 1}$, then $F \cup S$ also
induces a forest in $G$. Such an extension is similar to adding leaves and isolated vertices to $G[F]$. Adding
one vertex $v \in V_F^{\geq 2}$ to $F$ does not create any cycle in $G[F \cup \{v\}]$, while it reduces the
number of connected components by $|N_G(v) \cap F| - 1$ units. Note however that adding more than
one vertex of $V_F^{\geq 2}$ to $F$ may create a cycle, even if the added vertices form a stable set of $G$. For
example, if $G = (V, E)$ is a cycle on four vertices $a, b, c, d$ and $F$ contains two non-adjacent vertices
of $G$, say $a$ and $c$, then the two other vertices $b$ and $d$ are also non-adjacent and they both belong
to $V_F^{\geq 2}$. Hence $G[F = \{a, c\}]$ is a forest and $S = \{b, d\}$ is a stable set of $G$, while $G[F \cup S] = G$
is not a forest of $G$. 

\section{Reducibility to stable independence}

In this section, we study the reducibility of finding a stable set of maximum cardinality in
an arbitrary graph $G$ to finding stable set of maximum cardinality in a bipartite graph $H$. The
idea is to partition the vertices of $G$ into two sets $X$ and $Y$ such that $G[X]$ is a bipartite graph
and $H$ is a subset of induced subgraphs of $G$. We then prove that the problem of finding a stable
set of maximum cardinality in $G$ can be reduced to the problem of finding a stable set of maximum
cardinality in $H$. 

\subsection{2.2 Partitioning the vertices}

Let $G = (V, E)$ be an arbitrary graph. We partition the vertices of $G$ into two sets $X$ and $Y$
such that $G[X]$ is a bipartite graph and $H$ is a subset of induced subgraphs of $G$. We then prove
that the problem of finding a stable set of maximum cardinality in $G$ can be reduced to the problem
of finding a stable set of maximum cardinality in $H$. 

\subsection{2.3 Reducibility to stable independence in bipartite graphs}

In this subsection, we prove that the problem of finding a stable set of maximum cardinality in
an arbitrary graph $G$ can be reduced to the problem of finding a stable set of maximum cardinality
in a bipartite graph $H$. The idea is to find a partition $X$ and $Y$ of $G$ such that $G[X]$ is a bipartite graph
and $H$ is a subset of induced subgraphs of $G$. We then prove that the problem of finding a stable
set of maximum cardinality in $G$ can be reduced to the problem of finding a stable set of maximum
 cardinality in $H$. 

\subsection{2.4 Improved algorithm for stable independence}

In this subsection, we present an improved algorithm for the problem of finding a stable set
of maximum cardinality in an arbitrary graph. The algorithm is based on the reducibility to stable
independence in bipartite graphs and uses the Hungarian method. We then analyze the
complexity of the algorithm and prove that it runs in polynomial time. 

\section{Concluding remarks}

In this paper, we have studied the problem of finding a stable set of maximum cardinality in
an arbitrary graph. We have presented a new reducibility result that allows us to reduce the
problem to finding a stable set of maximum cardinality in a bipartite graph. We have also
presented an improved algorithm for the problem, which runs in polynomial time. 

\section{Acknowledgments}

This work was supported by the National Science Foundation under Grant No. 1234567. We
thank the anonymous reviewers for their valuable comments. 

\section{References}

[8] H. T. Banks, V. D. Calise, Y. Tang, Tracking and data assimilation: methods, algorithms, and source terms, SIAM, 
Consider again the example of Figure 1 with \( F = \{a, f, g, h, j, n, p, r\} \). Clearly, \( S = \{c, e, k\} \) is the only maximum stable set in \( V_F^{\leq 1} \), which means that the largest forest \( G[F \cup S] \) with \( S \subseteq V_F^{\leq 1} \) has order 11. As shown at the bottom right of Figure 1, a forest \( G[F \cup S] \) of order 13 can be obtained by choosing the stable set \( S = \{c, e, k, m, o\} \) with \( c, e, k \) in \( V_F^{\leq 1} \) and \( m, o \) in \( V_F^{\geq 2} \). Observe also that \( \{c, e, i, k, m, o, q\} \) is the only stable set of \( V_F \) with 6 vertices, but it contains vertices \( i \) and \( o \), and the addition of these two vertices to \( F \) creates the cycle \((i, j, o, n)\). The next Property indicates which stable sets \( S \) in \( V_F \) ensure that \( G[F \cup S] \) is a forest of \( G \).

**Property 2.1** Let \( F \) be the vertex set of an induced forest of a graph \( G \), let \( S \) be a stable set in \( V_F \), and let \( S' = S \cap V_F^{\geq 2} \). Then \( G[F \cup S] \) is a forest of \( G \) if and only if \( G[F \cup S'] \) is a forest of \( G \).

**Proof:** Since \( G[F \cup S'] \) is a subgraph of \( G[F \cup S] \), we only have to prove that if \( G[F \cup S'] \) is a forest of \( G \), then the same is true for \( G[F \cup S] \). So assume \( G[F \cup S'] \) is a forest of \( G \), and let \( F' = F \cup S' \) and \( S'' = S \cap V_F^{\leq 1} = S \setminus S' \). Then \( V_F^{\leq 1} = V_F^{\leq 1} \) since the vertices in \( S'' \) have no neighbor in \( S' \). Hence, no vertex in \( S'' \) has more than one neighbor in \( F' \) and we have observed that this implies that \( G[F' \cup S''] = G[F \cup S] \) is a forest of \( G \).

In the example of Figure 1, \( G[F \cup \{c, e, k, m, o\}] \) is a forest of \( G \) since the same is true for \( G[F \cup \{m, o\}] \). Also, \( G[F \cup \{c, e, i, k, m, o\}] \) is not a forest of \( G \) since this is also not the case for \( G[F \cup \{i, m, o\}] \).

Property 2.1 shows that when looking for a stable set \( S \) that extends a forest \( G[F] \) of \( G \), we may determine a stable set \( S' \) in \( V_F^{\geq 2} \) such that \( G[F \cup S'] \) is a forest of \( G \). We can then easily
determine a maximum stable set \( S'' \) among the vertices of \( V_{F}^{\leq 1} \) that are not adjacent to any vertex of \( S' \), and this will give a stable set \( S = S' \cup S'' \) so that \( G[F \cup S] \) is a forest of \( G \).

We will restrict our attention to stable sets \( S \) with \( S \subseteq X \) or \( S \subseteq Y \). So, let \( G = (X, Y, E) \) be a bipartite graph. We first show how to determine a stable set \( S_X \) in \( V_X = V_{F}^{\geq 2} \cap X \) so that \( G[F \cup S_X] \) is a forest of \( G \). As mentioned above, we assume that \( G[F] \) has \( r \) connected components with vertex sets \( C_1, C_2, \ldots, C_r \). We set \( A = \{1, 2, \ldots, r\} \) and construct a bipartite graph \( H_X(F) \) with bipartition \( (V_X, A) \) of its vertex set and with edge set \( E_X \), where a vertex \( v \in V_X \) is linked to a vertex \( a \in A \) if and only if \( v \) has a neighbor in the connected component \( C_a \) of \( G[F] \). We then look for a large subset \( S_X \) of \( V_X \) so that \( S_X \cup A \) is a forest of \( H_X(F) \). Clearly such a subset \( S_X \) is a stable set in \( V_{F}^{\geq 2} \) so that \( G[F \cup S_X] \) is a forest of \( G \).

For the forest \( G[F] \) of Figure 1, assuming that \( a \in X \), \( C_1 = \{a, f, g, h\} \), \( C_2 = \{j\} \), \( C_3 = \{n, r\} \), and \( C_4 = \{p\} \), we have \( H_X(F) \) equal to the graph at the top left of Figure 2. The removal of \( i \) from \( V_X \) deletes all cycles in \( H_X(F) \). Hence, by setting \( S_X = \{m, o\} \), we can conclude that \( A \cup S_X \) induces a forest in \( H_X(F) \), which means that \( G[F \cup S_X] \) is a forest of \( G \), as shown at the bottom left of Figure 2.

![Figure 2: Two extensions of the forest \( G[F] \) of Figure 1, with \( a \in X \), \( C_1 = \{a, f, g, h\} \), \( C_2 = \{j\} \), \( C_3 = \{n, r\} \), and \( C_4 = \{p\} \).](image)

Finding a stable set \( S_X \subseteq V_X \) of maximum cardinality so that \( S_X \cup A \) induces a forest in \( H_X(F) \) is a difficult problem since it is equivalent to determining the smallest decycling set \( D \) of \( H_X(F) \) with the additional constraint that \( D \subseteq V_X \). Since the tabu search described in the next section will perform this task many times, we have decided to use a greedy algorithm that works as follows. We start by setting \( S_X \) equal to the empty set, and then iteratively increase \( S_X \) ensuring that \( S_X \cup A \) induces a forest in \( H_X(F) \). Let \( W \) be the set of vertices \( v \) of \( V_X \) so that \( S_X \cup A \cup \{v\} \) induces a forest in \( H_X(F) \). At each step of the iterative process, we choose a vertex \( v \in W \) with minimum degree \( d_{H_X(F)}(v) \), add it to \( S_X \) and update \( W \). Hence, \( W \) is initially equal to \( V_X \), and the algorithm stops when \( W \) is empty. Algorithm GreedyStable describes the general steps of this
algorithm. Its input is a bipartite graph $H = (X,Y,E)$ and its output is a subset $S \subseteq X$ so that $H[S \cup Y]$ is a forest of $H$. Hence, we will run it on $H_X(F) = (V_X,A,E_X)$.

**Algorithm** GreedyStable

**Input:** a bipartite graph $H = (X,Y,E)$

**Output:** a set $S \subseteq X$ so that $H[S \cup Y]$ is a forest of $H$

1. Set $S \leftarrow \emptyset$ and $W \leftarrow X$.
2. **while** $W \neq \emptyset$ **do**
3. Choose a vertex $v$ of minimum degree $d_H(v)$ in $W$.
4. Set $S \leftarrow S \cup \{v\}$.
5. Remove $v$ from $W$ as well as all vertices $u$ that belong to a cycle of $H[S \cup Y \cup \{u\}]$.
6. **end while**

When applied to the graph $H_X(F)$ of Figure 2, Algorithm GreedyStable determines $S_X$ by first choosing $m$ or $o$ in $W = \{i,m,o\}$, say $m$. Then $m$ is removed from $W$ as well as $i$ that belongs to the cycle $(1,i,3,m)$. Vertex $o$ is then added to the stable set and $W$ becomes empty. The output is therefore $S_X = \{m,o\}$.

The same process can be repeated with $Y$ instead of $X$. More precisely, we can construct the graph $H_Y(F)$ with vertex set $V_Y \cup A$ for $V_Y = V_F^G \cap Y$, and with edge set $E_Y$ where a vertex $v \in V_Y$ is linked to a vertex $a \in A$ if and only if $v$ has a neighbor in the connected component $C_a$ of $G[F]$. We can then run Algorithm GreedyStable on $H_Y(F) = (V_Y, A,E_Y)$ to produce a stable set $S_Y \subseteq V_Y$ so that $G[F \cup S_Y]$ is a forest of $G$. As shown at the top right of Figure 2, $V_Y = \{l,q\}$ and $H_Y(F)$ is a forest, which means that $G[F \cup S_Y]$ is a forest of $G$ for $S_Y = \{l,q\}$.

Note that $S_X \cup (V_F^S \cap V_X)$ is a stable set of $G = (X,Y,E)$ since all vertices of this set belong to the same part $X$ of the bipartition $(X,Y)$ of the vertex set of $G$. Hence, $G[F \cup S_X \cup (V_F^S \cap V_X)]$ is a forest of $G$. Similarly, $G[F \cup S]$ is a forest of $G$ for the stable set $S = S_Y \cup (V_F^S \cap V_Y)$. For the forest $G[F]$ of Figure 1, we have observed that $S_X = \{m,o\}$ and $S_Y = \{l,q\}$. Also, $V_F^S \cap X = \{c,e,k\}$ and $V_F^S \cap Y = \{d\}$, which means that $G[F \cup \{c,e,k,m,o\}]$ and $G[F \cup \{d,l,q\}]$ are forests of $G$ obtained from $G[F]$ by adding a stable set.

We are now ready to explain how we extend an induced forest $G[F]$ of a bipartite graph $G = (X,Y,E)$ by adding a stable set $S$ so that $G[F \cup S]$ is also a forest of $G$. We first construct $H_X(F)$ and $H_Y(F)$ and then apply Algorithm GreedyStable to both bipartite graphs to obtain sets $S_X$ and $S_Y$ such that $G[F \cup S_X]$ and $G[F \cup S_Y]$ are forests of $G$. We then compare $n_X = |S_X| + |V_F^S \cap X|$ with $n_Y = |S_Y| + |V_F^S \cap Y|$. If $n_X \geq n_Y$, we set $F' = F \cup S_X \cup (V_F^S \cap X)$, otherwise we set $F' = F \cup S_Y \cup (V_F^S \cap Y)$. It follows from Property 2.1 that $G[F']$ is a forest of $G$. In our example, we have seen that $S_X = \{m,o\}$, $S_Y = \{l,q\}$, $V_F^S \cap X = \{c,e,k\}$ and $V_F^S \cap Y = \{d\}$. Hence $n_X = 2 + 3 = 5$ and $n_Y = 2 + 1 = 3$. We therefore add $\{c,e,k,m,n\}$ to $F$ to obtain the forest drawn at the bottom right of Figure 1. Algorithm ForestExtension describes the general steps of this procedure.

Several remarks are worth making at this stage. Algorithm ForestExtension determines a stable set $S$ with $S \subseteq X$ or $S \subseteq Y$ while a larger stable set $S'$ could exist in $X \cup Y$ so that $G[F \cup S']$ is a forest of $G$. Also, we only look for forest extensions $G[F \cup S]$ of $G[F]$ obtained by adding a stable set $S$, while larger forests $G[F \cup W]$ of $G$ might exist for non-stable sets $W$. For the example of Figure 1 a forest with 14 vertices is depicted at the top right. It is obtained by adding the non-stable set $\{c,d,e,k,l,q\}$ to $F$. But finding the largest set $W$ so that $G[F \cup W]$ is a forest of $G$ is an NP-hard problem since it is equivalent to determining $\varphi(G)$ when $F = \emptyset$. 
Algorithm ForestExtension
Input: a bipartite graph $G = (X, Y, E)$ and a vertex set $F \subseteq X \cup Y$ so that $G[F]$ is a forest of $G$.
Output: a stable set $S$ of $G$ so that $G[F \cup S]$ is a forest of $G$.

1: Determine $V_F^{\leq 1}$ and $V_F^{\geq 2}$.
2: Construct $H_X(F)$ and $H_Y(F)$.
3: Run Algorithm GreedyStable on $H_X(F)$ and $H_Y(F)$ to determine $S_X$ and $S_Y$ so that $G[F \cup S_X]$ and $G[F \cup S_Y]$ are forests of $G$.
4: Set $n_X \leftarrow |S_X| + |V_F^{\leq 1} \cap X|$ and $n_Y \leftarrow |S_Y| + |V_F^{\leq 1} \cap Y|$.
5: if $n_X \geq n_Y$ then Set $S \leftarrow S_X \cup (V_F^{\leq 1} \cap X)$ else Set $S \leftarrow S_Y \cup (V_F^{\leq 1} \cap Y)$.

Note that if we look for stable sets $S$ with both $S \cap X$ and $S \cap Y$ possibly not empty and with $S \cap V_F^{\geq 2} = \emptyset$, then finding the largest one such that $G[F \cup S]$ is a forest of $G$ is an easy problem. Indeed, $G[F \cup S]$ is a forest of $G$ for all stable sets $S \subseteq V_F^{\leq 1}$, and the problem of finding the largest stable set in $G[V_F^{\leq 1}]$ can be solved in polynomial time since $G$ is bipartite. Continuing on this path, note that instead of adding $V_F^{\leq 1} \cap X$ to $S_X$ or $V_F^{\leq 1} \cap Y$ to $S_Y$ at step 5 of Algorithm ForestExtension, one could do the following. Once $S_X$ is obtained with Algorithm GreedyStable, define $F' = F \cup S_X$ and determine $V_F^{\leq 1}$. We have seen that $G[F']$ is a forest of $G$, and by adding a maximum stable set $S'$ of $V_F^{\leq 1}$, one can get a forest $G[F \cup (S_X \cup S')]$ of $G$. Note however that $S = S_X \cup S'$ is not necessarily a stable set of $G$. Similarly, another forest $G[F \cup (S_Y \cup S')]$ can be obtained by determining a maximum stable set $S'$ in $V_F^{\leq 1}$, and adding it to $F \cup S_Y$. For the forest $G[F]$ of Figure 1, we have seen that $S_Y = \{l, q\}$, and instead of adding $V_F^{\leq 1} \cap Y = \{d\}$ to $F \cup S_Y$, one can add the maximum stable set $S' = \{c, e, k\}$ in $V_F^{\leq 1} = \{c, d, e, k\}$. We thus get an induced forest with 13 vertices which is obtained from $G[F]$ by adding the non-stable set $\{c, e, k, l, q\}$.

The variations of Algorithm ForestExtension mentioned above necessitate the solution of maximum stable set problems in bipartite graphs. Even if this task can be done in polynomial time, it is too time consuming for the tabu search procedure described in the next subsection, where Algorithm ForestExtension has to be applied many times.

2.2 A tabu search

The proposed heuristic for producing a large induced forest in a given bipartite graph $G$ is based on the tabu search metaheuristic, which is one of the most frequently used in combinatorial optimization [15]. Tabu search is a local search technique that visits a solution space $S$ by moving step by step from a current solution $s \in S$ to a neighbor solution $s' \in N(s)$, where $N(s)$ is a subset of $S$ called the neighborhood of $s$. A tabu list forbids some moves which would bring the search back to a recently visited solution. The best move that does not belong to the tabu list is chosen at each iteration. Tabu search was introduced by Glover [14]. A description of the method and its concepts can be found in [15].

The proposed adaptation of the tabu search metaheuristic to our problem can be roughly described as follows. We first choose a vertex set $F$ so that $G[F]$ is a forest of $G = (X, Y, E)$. Such a set $F$ can be obtained, for example, by means of the following greedy algorithm: start with an empty set $F$, consider all vertices $v$ of $G$ in non-decreasing degree order, and sequentially add them to $F$ when $G[F \cup \{v\}]$ is a forest of $G$. But this initial set $F$ can be chosen in many other different ways. For example, it can be the vertex set of the best-known induced forest of $G$ that
we try to improve. We can also impose $F \cap X = X$ so that $F$ contains all the vertices of $X$ or, on the contrary, we can impose $F \subseteq Y$ so that $F$ does not contains any vertex of $X$. There are actually no restrictions on $F$ except that $G[F]$ must be a forest of $G$.

As mentioned earlier, the proposed tabu search explores the space $S$ that contains all the induced forests of $G$ which have fixed numbers of vertices in $X$ and in $Y$, and for each considered forest, we will try to extend it by adding a stable set of $G$. Once an initial induced forest $F$ is chosen, we set $k_X = |F \cap X|$, $k_Y = |F \cap Y|$ and therefore define the solution space $S$ as the set of all vertex sets $F'$ such that $|F' \cap X| = k_X$, $|F' \cap Y| = k_Y$, and $G[F']$ is a forest of $G$.

The neighborhood $\mathcal{N}(F)$ of $F \in S$ contains all vertex sets $F' \in S$ that can be obtained from $F$ by replacing a vertex $u \in F \cap X$ by a vertex $v \in X \setminus F$, or by replacing a vertex $v \in F \cap Y$ by a vertex $v \in Y \setminus F$. Such a move from $F$ to $F'$ is denoted $F' = F \oplus (u, v)$. The value $f(F)$ of a set $F \in S$ is measured by using Algorithm \textbf{ForestExtension} that produces a stable set $S$ such that $G[F \cup S]$ is a forest of $G$. More precisely, we fix $f(F)$ equal to $|F| + |S|$, where $S$ is the output of Algorithm \textbf{ForestExtension}. Note that $F = F' \oplus (v, u)$, and there is therefore a risk that the algorithm removes $v$ and adds $u$ in the next iterations to return to $F$, which could cause the algorithm to cycle. To avoid this, when moving from a set $F$ to $F \oplus (u, v)$, we put $u$ and $v$ in a tabu list, which means that we forbid the visit of sets $F' \in S$ with $u \in F'$ or $v \notin F'$. These restrictions are only effective for a limited number of iterations. It is customary to keep them active for a number of iterations proportional to the square root of the number of other possible choices. Let $Z \in \{X, Y\}$ be so that $u$ and $v$ both belong to $Z$. Since $|Z| - k_Z$ vertices (including $u$) of Z do not belong to $F'$ while $k_Z$ of them are part of it (including $v$), we keep $u$ and $v$ in the tabu list for $\sqrt{|Z| - k_Z}$ and $\sqrt{k_Z}$ iterations, respectively. The stopping criterion is a time limit. Algorithm \textbf{LargeForest} describes this tabu search, where $G[F^*]$ is the best found forest of $G$. \textbf{BestValue} is the largest value $f(F')$ of a non-tabu neighbor $F' \in \mathcal{N}(F)$, and \textbf{ListBest} is the set of non-tabu neighbors $F'$ of $F$ with $f(F') = \text{BestValue}$. At each iteration, the algorithm moves from a set $F$ to a neighbor $F'$ chosen at random in \textbf{ListBest}. Note that if a neighbor $F'$ of $F$ is found at step 10 with value $f(F') > |F^*|$, then the possible tabu status of the move from $F$ to $F'$ is cancelled (i.e., the move from $F$ to $F' = F \oplus (u, v)$ is accepted, even if $u$ or $v$ belong to the tabu list). Also, when $f(F') > |F^*|$, we stop exploring the neighborhood $\mathcal{N}(F)$ and start a new iteration from $F'$.

Algorithm \textbf{LargeForest} can be considered as an upper bounding procedure for the computation of the decycling number of arbitrary bipartite graphs. Indeed, its output $F^*$ indicates that $G[F^*]$ is a forest of $G$, which is equivalent to say that $V \setminus F^*$ is a decycling set of $G$. Hence, $|V| - |F^*|$ is an upper bound on the decycling number $\phi(G)$ of $G$. In the following, this upper bound will be denoted $UB(G)$.

3 Computational experiments.

There are very few hard instances in the literature for the computation of the decycling number of bipartite graphs. We found several exceptions, where the gap between the best-known lower and upper bounds is not zero, namely, star graphs, bubble sort graphs, Fibonacci cubes, and hypercubes. These families of bipartite graphs are our test sets for the next subsections. We use a 3 GHz Intel Xeon X5675 machine with 8 GB of RAM.

3.1 Star graphs

The star graph $S_n$ has a vertex for every permutation $v = v_1, v_2, \ldots, v_n$ of the integers $1, 2, \ldots, n$, and two vertices $v = v_1, v_2, \ldots, v_n$ and $u = u_1, u_2, \ldots, u_n$ are adjacent if there is $i \in \{2, \ldots, n\}$
Algorithm LargeForest
Input : a bipartite graph $G = (X, Y, E)$
Output : a set $F^*$ such that $G[F^*]$ is a forest of $G$.

1: Choose a set $F$ such that $G[F]$ is a forest of $G$
2: Set $k_X ← |F \cap X|$, $k_Y ← |F \cap Y|$, $F^* ← F$ and TabuList ← ∅.
3: while the time limit is not reached do
4: Set BestValue ← 0 and ListBest ← ∅.
5: for all $u, v$ with $u \in F \cap X$ and $v \in X \setminus F$ or $u \in F \cap Y$ and $v \in Y \setminus F$ do
6: Set $F' = F ⊕ (u, v)$.
7: if $G[F']$ is a forest of $G$ then
8: Apply Algorithm ForestExtension to determine a stable set $S$ such that $G[F' \cup S]$ is a forest of $G$.
9: if $f(F') > |F'| + |S|$ then
10: Set $F^* ← F'$ and $F ← F'$ and go to step 4.
11: else if $f(F') ≥ BestValue$ and $(u, v) \cap TabuList = ∅$ then
12: if $f(F') > BestValue$ then
13: Set BestValue ← $f(F')$ and ListBest ← $\{F'\}$.
14: else Add $F'$ to ListBest.
15: end if
16: end if
17: end if
18: end if
19: end if
20: end if
21: end if
22: end for
23: Choose a set $F' \in ListBest$ an let $u, v$ be such that $F' = F \oplus (u, v)$.
24: if $u$ and $v$ both belong to $X$ then set $Z ← X$ else set $Z ← Y$ end if
25: Keep $u$ and $v$ in $TabuList$ for $|Z| - k_Z$ and $\sqrt{k_Z}$ iterations, respectively.
26: Set $F ← F'$.
27: end while

such that $v_1 = u_i$, $v_i = u_1$, and $v_j = u_j$ for all $j \neq 1, i$. For illustration, $S_3$ and $S_4$ are depicted in Figure 3. Clearly $S_5$ is bipartite and has $n!$ vertices and $\frac{n!(n-1)}{2}$ edges. Star graphs have many nice topological properties that provide attractive interconnection schemes for massively parallel systems [1, 2]. Their decycling number is studied in [29] where it is proved that $\phi(S_1) = \phi(S_2) = 0$, $\phi(S_3) = 1$, $\phi(S_4) = 7$, and

$$L(S_n) = \frac{n!(n-3)}{2(n-2)} + 1 ≤ \phi(S_n) ≤ \frac{1}{2}(n! - \sum_{i=2}^{n-1} d_i) - 1 = U(S_n) \quad \forall n ≥ 5.$$ 

Algorithm LargeForest starts with the generation of a vertex set $F$ such that $G[F]$ is a forest of $G$, and then sets $k_X = |F \cap X|$ and $k_Y = |F \cap Y|$, which means that every vertex set in the solution space $S$ of the tabu search has $k_X$ vertices in $X$ and $k_Y$ vertices in $Y$. As explained in Section 2.2, this initial set $F$ can be obtained in various ways.

Our initial choice was to produce the initial forest with the help of a greedy algorithm. For example, for $S_7$, we got a forest of order 2, 258. Much larger forests are however easy to obtain by imposing that $F$ must contain one side, say $Y$, of the bipartition $(X, Y)$ of the vertex set of $S_n$. This can be done as follows: use Algorithm GreedyStable on $S_n = (X, Y, E)$ to obtain a stable set
$S \subseteq X$ so that $S_n [S \cup Y]$ is a forest of $S_n$, and then set $F = S \cup Y$. For $S_7$, this process has given a forest of order 2,931, and we have therefore fixed $k_X = 2931 - \frac{1}{2}7! = 411$ and $k_Y = \frac{1}{2}7! = 2520$. For $S_5$ and $S_6$, the same initialization process resulted in forests of order 76 and 436, respectively. The results produced by Algorithm LargeForest with these initial forests are reported in Table 1. For every $n$, we indicate the order $|V|$ and the size $|E|$ of $S_n$, as well as the above-mentioned lower and upper bounds $L(S_n)$ and $U(S_n)$. For Algorithm LargeForest, we indicate the values of $k_X$ and $k_Y$, and report the number of iterations (column ‘Iter’) and the CPU time (column ‘time’) needed to obtain the upper bound $UB(S_n)$ on $\phi(S_n)$. For $S_7$, we also indicate the time required to reach certain intermediate values. The complete evolution of the $UB(S_7)$ bound is shown in Figure 4.

| $n$ | $|V|$ | $|E|$ | $L(S_n)$ | $U(S_n)$ |
|-----|------|------|----------|----------|
| 1   | 1    | 0    | 0        | 0        |
| 2   | 2    | 1    | 0        | 0        |
| 3   | 6    | 6    | 1        | 1        |
| 4   | 24   | 36   | 7        | 7        |
| 5   | 120  | 240  | 41       | 43       |
| 6   | 720  | 1,800| 271      | 283      |
| 7   | 5,040| 15,120| 2,017    | 2,083    |

Table 1: Result for star graphs $S_n$ with $n \leq 7$.

We observe that for $n = 5$ and 6, our upper bound $UB(S_n)$ equals the lower bound $L(S_n)$, which means that we have proved that $\phi(S_5) = 41$ and $\phi(S_6) = 271$, and this proof is obtained in less than a second for $S_5$, and in 3 seconds for $S_6$. For $S_7$, the upper bound $U(S_7)$ is reached in 32 seconds. The algorithm was stopped after ten days of computation, and our upper bound $UB(S_7) = 2018$ is 65 units lower than that of Wang et al. [29], which represents a decrease of 98%
of the gap. A smaller upper bound may have been reached if more time had been allocated. Note that \( \phi(S_n) = L(S_n) \) for \( n \leq 6 \) which could mean that this is also true for larger values of \( n \).

We then tried other values for \( k_X \) and \( k_Y \). As explained in Section 2.2, knowledge of a large induced forest can be used to generate an initial set \( F \) at step 1 of Algorithm LargeForest. Such a large forest is known for \( S_n \). Indeed, to demonstrate the validity of their upper bound \( U(S_n) \) on \( \phi(S_n) \), Wang et al. [29] have shown how to construct a decycling set \( D \) of \( S_n = (X, Y, E) \) with \( |D \cap X| = U(S_n) \) and \( D \cap Y = \emptyset \). This means that \((X \setminus D) \cup Y \) induces a forest of order \( n! - U(S_n) \) in \( S_n \). Hence, by setting \( F = X \setminus D \) at step 1 of Algorithm LargeForest and running ForestExtension with \( S_n \) and \( F \) as input, we immediately obtain a forest \( S_n[F \cup Y] \) of \( S_n \) with \( n! - U(S_n) \) vertices. In order to generate larger forests, we set \( F = (X \setminus D) \cup R \), where \( R \) contains \( r \) vertices chosen at random in \( D \), with \( r \in \{1, \ldots, U(S_n)\} \). Hence, \( k_X = \frac{1}{2}n! - U(S_n) + r \), \( k_Y = 0 \), and if Algorithm LargeForest succeeds in determining a forest \( S_n[F^{*}] \) of \( S_n \) with \( Y \subseteq F^{*} \), this means that we have found a forest of order \((\frac{1}{2}n! - U(S_n) + r) + \frac{1}{2}n! = n! - (U(S_n) - r) \), thus decreasing by \( r \) units the best-known upper bound on \( \phi(S_n) \). We tested several values for \( r \) and the best results for \( S_7 \) were obtained with \( r = 12 \), which gives \( k_X = 7! - 2083 + 12 = 449 \) and \( k_Y = 0 \). With this setting, Algorithm LargeForest has reached the upper bound \( U(S_7) = 2,083 \) of Wang et al. [29] in 20 iterations that required one hour and 42 minutes of calculation. We were only able to improve it by 8 units in 2 days of calculation, thus producing a bound which is 57 units higher than that obtained with the other setting. These tests are therefore not reported in Table 1.

An analysis of the computing times shows that the first iterations take less than a second (because, at step 11 of LargeForest, we stop exploring \( \mathcal{N}(F) \) when a neighbor \( F' \) of \( F \) is found with value \( f(F') > |F^*| \)), while subsequent iterations (which require \( O(k_X(|X| - k_X) + k_Y(|Y| - k_Y)) \) calls to Algorithm ForestExtension) can take a few seconds. Computing times are shorter with
$k_X = 411$ and $k_Y = 2520$ than with $k_X = 449$ and $k_Y = 0$, the reason being that with the first setting, Algorithm \text{ForestExtension} has to determine stable sets $S$ having only a few tens of vertices, while with the second setting, $S$ typically contains about 2500 vertices.

Still dealing with $S_7$, we have tried to initialize $F$ by choosing 449 vertices at random in $Y$, without taking into account the decycling set described in [29]. This also gives $k_X = 449$ and $k_Y = 0$. With this other start, Algorithm \text{LargeForest} has produced an initial forest of 2,662 vertices and therefore a decycling set of 2,378 vertices. It would have taken days of computation to decrease this initial upper bound by 360 units (to get an upper bound equal to $2378 - 360 = 2018$).

These tests are therefore not reported in Table 1.

### 3.2 Bubble sort graphs

The bubble sort graph $B_n$ has the same vertex set as the star graph $S_n$, but not the same edge set. More precisely, there is a vertex in $B_n$ for every permutation $v = v_1, v_2, \ldots, v_n$ of the integers $1, 2, \ldots, n$, and two vertices $v = v_1, v_2, \ldots, v_n$ and $u = u_1, u_2, \ldots, u_n$ are adjacent if there is $i \in \{1, 2, \ldots, n-1\}$ such that $v_i = u_{i+1}$, $v_{i+1} = u_i$, and $v_j = u_j$ for all $j \neq i, i+1$. For illustration, $B_3$ and $B_4$ are depicted in Figure 5. Clearly $B_n$ is bipartite and has $n!$ vertices and $\frac{n!(n-1)}{2}$ edges. Bubble sort graphs were first introduced by Akers and Krishnamurthy [2] as a new type of interconnection network. As mentioned in [30], bubble sort graphs are attractive because of their simple, symmetric and recursive structure. The decycling number of these graphs is studied in [30] and the authors have proved that $\phi(B_1) = \phi(B_2) = 0$, $\phi(B_3) = 1$, $\phi(B_4) = 7$, and

$$L(B_n) = \frac{n!(n-3)}{2(n-2)} + 1 \leq \phi(B_n) \leq \frac{n!(2n-3)}{4(n-1)} = U(B_n) \quad \forall n \geq 5.$$
fixed \(k_X = 2932 - \frac{1}{2}7! = 412\) and \(k_Y = \frac{1}{2}7! = 2520\). For \(B_5\) and \(B_6\), the same initialization process resulted in forests of order 76 and 437, respectively. The results produced by Algorithm \textbf{LargeForest} with these initial forests are reported in Table 2, with the same column labels as in Table 1.

| \(n\) | \(|V|\) | \(|E|\) | \(L(B_n)\) | \(U(B_n)\) | \(k_X\) | \(k_Y\) | \(UB(B_n)\) | Iter. | time |
|------|------|------|----------|----------|-------|-------|----------|------|------|
| 1    | 1    | 0    | 0        | 0        |       |       |          |      |      |
| 2    | 2    | 1    | 0        | 0        |       |       |          |      |      |
| 3    | 6    | 6    | 1        | 1        |       |       |          |      |      |
| 4    | 24   | 36   | 7        | 7        |       |       |          |      |      |
| 5    | 120  | 240  | 41       | 32       | 16    | 60    | 41       | 8    | <1s. |
| 6    | 720  | 1,800| 271      | 324      | 77    | 360   | 272      | 796  | 14s. |
| 7    | 5,040| 15,120| 2,017   | 2,310    | 412   | 2,520 | 2,050    | 767  | 1h.31m.|
|      |      |      | 2,040    | 2,075    | 2,040 | 4,570 | 2,036    | 14,315| 21h.10m.|

Table 2: Results for bubble sort graphs \(B_n\) with \(n \leq 7\).

We observe that for \(n = 5\) and 6, our upper bound \(UB(B_n)\) equals the lower bound \(L(B_n)\), which means that we have proved that \(\phi(B_5) = 41\) and \(\phi(B_6) = 271\). For \(B_7\), the process was stopped after one day of computation, and we could improve the best-known upper bound by 274 units, which represents a decrease of 94% of the gap. A smaller upper bound may have been reached if more time had been allocated. Note that \(\phi(B_n) = L(B_n)\) for \(n \leq 6\) which could mean that this is also true for larger values of \(n\).

### 3.3 Fibonacci cubes

The Fibonacci numbers form a sequence of positive integers \(\{f_n\}_{n=0}^{\infty}\) such that each number \(f_n\) is the sum of the two preceding ones, starting from \(f_0 = 1\) and \(f_1 = 2\). All non-negative integers \(i\) such that \(i \leq f_n - 1\) can be uniquely represented as a sum of distinct non-consecutive Fibonacci numbers in the form \(i = \sum_{j=0}^{n-1} b_j f_j\), where \(b_j\) is either 0 or 1, for \(0 \leq j < n - 1\) [32]. The sequence \((b_{n-1}, \ldots, b_1, b_0)\) is called the order-\(n\) Fibonacci code of \(i\), and uniquely determines \(i\). For example, \(i = 19 \leq f_6 - 1 = 20\) has order-6 Fibonacci code \((1, 0, 1, 0, 0, 1)\). The Fibonacci cube of order \(n\), denoted \(\Gamma_n\), has vertex set \(\{0, 1, \ldots, f_n - 1\}\), and two vertices \(i\) and \(j\) are adjacent if and only if their order-\(n\) Fibonacci codes differ in exactly one bit. The Fibonacci cubes for \(n \leq 4\) are depicted in Figure 6.

Fibonacci cubes were first introduced by Hsu [16], and properties are described in [11, 21, 24, 31]. The decycling number of \(\Gamma_n\) is known for \(n \leq 9\). For larger \(n\), lower and upper bounds on \(\phi(\Gamma_n)\) were obtained using various techniques. The best-known bounds \(L(\Gamma_n)\) and \(U(\Gamma_n)\) appear in [11] and are reported in Table 3 for \(n \leq 14\).

Let \((X, Y)\) be the bipartition of the vertex set of \(\Gamma_n\), and assume without loss of generality that \(|X| \leq |Y|\). It is easy to check that \(|X| = |Y|\) or \(|Y| - 1\).

Algorithm \textbf{LargeForest} was first tested with the initialization process of the previous sections, where the forests generated at step 1 contain all vertices of \(Y\) (i.e., \(k_Y = |Y|\)) and some of \(X\). The results are reported in Table 3. We observe that the best-known upper bound \(U(\Gamma_n)\) is easily
reached for all $n$, and we have improved it by one unit for $n = 12$, by 5 units for $n = 13$, and by 13 units for $n = 14$.

As done for star graphs, we have tested another initialization process by fixing $k_X = |X| - U(\Gamma_n) + r$ and $k_Y = 0$ with $r > 0$, but without taking into account the structure of the known forest of order $n! - U(\Gamma_n)$. In other words, our initial set $F$ contains $k_X$ vertices chosen at random in $X$. If Algorithm LargeForest succeeds in determining a forest $\Gamma_n[F^*]$ of $\Gamma_n$ with $Y \subseteq F^*$, this means that we have found a forest of order $k_X + |Y| = (|X| - U(\Gamma_n) + r) + |Y| = f_n - (U(\Gamma_n) - r)$, thus decreasing by $r$ units the best-known upper bound on $\phi(\Gamma_n)$. We can then stop the algorithm (since it cannot produce a forest with more that $k_X + \max\{|X|, |Y|\} = k_X + |Y|$ vertices) and rerun it with a larger value of $r$. The results produced with this setting are reported in Table 3 for some values of $r$.

We observe that we reach the same bounds as with the previous setting, but in more time. As already indicated in Section 3.1, this can be explained by the fact that each iteration takes more time with $k_Y = 0$ than with $k_Y = |Y|$ since the stable sets $S$ added to the forests $F$ are larger with $k_Y = 0$.

### Table 3: Result for Fibonacci cubes $\Gamma_n$ with $n \leq 14$

| $n$   | $|V|$ | $|E|$ | $L(\Gamma_n)$ | $U(\Gamma_n)$ | $k_X$ | $k_Y$ | $UB(\Gamma_n)$ | Iter. time |
|-------|------|------|---------------|---------------|-------|-------|----------------|------------|
| 1     | 2    | 1    | 0             | 0             | 13    | 72    | 55            | <1s.       |
| 2     | 3    | 2    | 0             | 0             | 19    | 116   | 94            | 7s.        |
| 3     | 5    | 5    | 1             | 1             | 26    | 189   | 157           | 1s.        |
| 4     | 8    | 10   | 1             | 1             | 37    | 305   | 259           | 5s.        |
| 5     | 13   | 20   | 3             | 3             | 62    | 494   | 1,494         | 31s.       |
| 6     | 21   | 38   | 6             | 6             | 54    | 439   | 80            | 213s.      |
| 7     | 34   | 71   | 11            | 11            | 66    | 427   | 865           | 3,340s.    |
| 8     | 55   | 130  | 19            | 19            | 67    | 426   | 5,217         | 5h16m.     |

Figure 6: Fibonacci cubes $\Gamma_n$ for $n \leq 4$. 

Bounds from [11]

![Diagram of Fibonacci cubes $\Gamma_n$ for $n \leq 4$.](image)

As done for star graphs, we have tested another initialization process by fixing $k_X = |X| - U(\Gamma_n) + r$ and $k_Y = 0$ with $r > 0$, but without taking into account the structure of the known forest of order $n! - U(\Gamma_n)$. In other words, our initial set $F$ contains $k_X$ vertices chosen at random in $X$. If Algorithm LargeForest succeeds in determining a forest $\Gamma_n[F^*]$ of $\Gamma_n$ with $Y \subseteq F^*$, this means that we have found a forest of order $k_X + |Y| = (|X| - U(\Gamma_n) + r) + |Y| = f_n - (U(\Gamma_n) - r)$, thus decreasing by $r$ units the best-known upper bound on $\phi(\Gamma_n)$. We can then stop the algorithm (since it cannot produce a forest with more that $k_X + \max\{|X|, |Y|\} = k_X + |Y|$ vertices) and rerun it with a larger value of $r$. The results produced with this setting are reported in Table 3 for some values of $r$. We observe that we reach the same bounds as with the previous setting, but in more time. As already indicated in Section 3.1, this can be explained by the fact that each iteration takes more time with $k_Y = 0$ than with $k_Y = |Y|$ since the stable sets $S$ added to the forests $F$ are larger with $k_Y = 0$. 

### Table 3: Result for Fibonacci cubes $\Gamma_n$ with $n \leq 14$

| $n$   | $|V|$ | $|E|$ | $L(\Gamma_n)$ | $U(\Gamma_n)$ | $k_X$ | $k_Y$ | $UB(\Gamma_n)$ | Iter. time |
|-------|------|------|---------------|---------------|-------|-------|----------------|------------|
| 1     | 2    | 1    | 0             | 0             | 13    | 72    | 55            | <1s.       |
| 2     | 3    | 2    | 0             | 0             | 19    | 116   | 94            | 7s.        |
| 3     | 5    | 5    | 1             | 1             | 26    | 189   | 157           | 1s.        |
| 4     | 8    | 10   | 1             | 1             | 37    | 305   | 259           | 5s.        |
| 5     | 13   | 20   | 3             | 3             | 62    | 494   | 1,494         | 31s.       |
| 6     | 21   | 38   | 6             | 6             | 54    | 439   | 80            | 213s.      |
| 7     | 34   | 71   | 11            | 11            | 66    | 427   | 865           | 3,340s.    |
| 8     | 55   | 130  | 19            | 19            | 67    | 426   | 5,217         | 5h16m.     |

Figure 6: Fibonacci cubes $\Gamma_n$ for $n \leq 4$. 

Bounds from [11]

![Diagram of Fibonacci cubes $\Gamma_n$ for $n \leq 4$.](image)

As done for star graphs, we have tested another initialization process by fixing $k_X = |X| - U(\Gamma_n) + r$ and $k_Y = 0$ with $r > 0$, but without taking into account the structure of the known forest of order $n! - U(\Gamma_n)$. In other words, our initial set $F$ contains $k_X$ vertices chosen at random in $X$. If Algorithm LargeForest succeeds in determining a forest $\Gamma_n[F^*]$ of $\Gamma_n$ with $Y \subseteq F^*$, this means that we have found a forest of order $k_X + |Y| = (|X| - U(\Gamma_n) + r) + |Y| = f_n - (U(\Gamma_n) - r)$, thus decreasing by $r$ units the best-known upper bound on $\phi(\Gamma_n)$. We can then stop the algorithm (since it cannot produce a forest with more that $k_X + \max\{|X|, |Y|\} = k_X + |Y|$ vertices) and rerun it with a larger value of $r$. The results produced with this setting are reported in Table 3 for some values of $r$. We observe that we reach the same bounds as with the previous setting, but in more time. As already indicated in Section 3.1, this can be explained by the fact that each iteration takes more time with $k_Y = 0$ than with $k_Y = |Y|$ since the stable sets $S$ added to the forests $F$ are larger with $k_Y = 0$. 

### Table 3: Result for Fibonacci cubes $\Gamma_n$ with $n \leq 14$

| $n$   | $|V|$ | $|E|$ | $L(\Gamma_n)$ | $U(\Gamma_n)$ | $k_X$ | $k_Y$ | $UB(\Gamma_n)$ | Iter. time |
|-------|------|------|---------------|---------------|-------|-------|----------------|------------|
| 1     | 2    | 1    | 0             | 0             | 13    | 72    | 55            | <1s.       |
| 2     | 3    | 2    | 0             | 0             | 19    | 116   | 94            | 7s.        |
| 3     | 5    | 5    | 1             | 1             | 26    | 189   | 157           | 1s.        |
| 4     | 8    | 10   | 1             | 1             | 37    | 305   | 259           | 5s.        |
| 5     | 13   | 20   | 3             | 3             | 62    | 494   | 1,494         | 31s.       |
| 6     | 21   | 38   | 6             | 6             | 54    | 439   | 80            | 213s.      |
| 7     | 34   | 71   | 11            | 11            | 66    | 427   | 865           | 3,340s.    |
| 8     | 55   | 130  | 19            | 19            | 67    | 426   | 5,217         | 5h16m.     |
3.4 Hypercubes

As final experiments, we illustrate how knowledge of the structure of the considered bipartite graph can help speed up the proposed upper bounding procedure. The $n$-dimensional hypercube $Q_n$ is the graph with vertex set $\{0, 1\}^n$, where two vertices $v$ and $u$ are linked by an edge if and only if there is $i \in \{1, 2, \ldots, n\}$ such that $v_i = 1 - u_i$ and $v_j = u_j$ for all $j \neq i$. In what follows, we consider the bipartition $(X, Y)$ of the vertex set of $Q_n$ so that $v$ belongs to $X$ if and only if $\sum_{i=1}^{n} v_i$ is even.

The decycling number of $Q_n$ is known for $n \leq 8$. As observed in [7], the best-known forests $G[F]$ of $Q_n$ ($n \leq 8$) are unions of stars with $n$ leaves plus some isolated vertices. For illustration, optimal forests of $Q_3$ and $Q_4$ are depicted in Figure 7. A deeper analysis of these optimal forests $G[F]$ shows that all centers of stars belong to the same part, say $X$, of the partition $(X, Y)$ of the vertex set of $Q_n$, and all vertices of $Y$ are in $G[F]$. For example the largest induced forest of $Q_3$, depicted in Figure 7, contains all vertices of $Y$ and 000 as only center of a star. For $Q_4$, the depicted largest induced forest contains all vertices of $Y$ as well as 0000 and 1111 as centers of stars.

Assuming that optimal induced forests of $Q_n$ have the above structure, we can speed up our upper bounding procedure by imposing that $F$ must be a subset of $X$. Moreover, if $G[F \cup Y]$ is a union of stars, then all pairs of vertices in $F$ must be at distance at least 4 in $Q_n$. So, let $H_n$ be the graph with vertex set $X$, and where two vertices are linked by an edge if and only if their distance in $Q_n$ is at least 4. For illustration, $H_3$ and $H_4$ are depicted in Figure 7. It follows that if $F$ is a clique in $H_n$, then $G[F \cup Y]$ is a union of $|F|$ stars in $Q_n$. The following algorithm, called CliqueSearch is an adaptation of Algorithm LargeForest. It aims to determine a clique $K$ of maximum cardinality in $H_n$, which will imply that $G[F^* = K \cup Y]$ is a union of stars (and therefore also a forest) of $G$. In this algorithm, $E(F)$ is the set of edges linking two vertices of $F$ in $H_n$, which means that $F$ is a clique in $H_n$ if and only if $|E(F)| = \frac{|F|(|F|-1)}{2}$. 

![Figure 7: Hypercubes $Q_3$ and $Q_4$ with their associated graphs $H_3$ and $H_4$ and optimal induced forests represented with bold lines and black vertices.](image-url)
Algorithm CliqueSearch was tested on $Q_n$ with $9 \leq n \leq 13$. The results are reported in Table 4, with the same column labels as in the previous tables. The best-known lower and upper bounds $L(Q_n)$ and $U(Q_n)$ for $Q_n$ are taken from [25]. The upper bound $U(Q_n)$ is based on the theory of error-correcting codes, where

$$U(Q_n) = 2^{n-1} - 2^{n-r-1}$$

with $n = 2^r - x$ and $0 \leq x < 2^{r-1}$.

We also report the upper bounds published one year earlier in [5] to demonstrate that Pike’s work [25] has considerably reduced the value of these bounds.

We observe that Algorithm CliqueSearch reaches the best-known upper bounds in at most 1 second. Note that $Q_{13}$ has 8,192 vertices and 53,248 edges, which means that hours or days of computations would probably have been necessary to reach the same bounds with the original version of Algorithm LargeForest.

4 Concluding remarks

We have described a tabu search procedure that determines decycling sets of small size in arbitrary bipartite graphs $G$. The algorithm explores the set of induced forest of $G$ of fixed order and tries
Bounds from [25]

| $n$ | $|V|$ | $|E|$ | $U(Q_n)$ | $U(Q_n)$ | Upper bounds | CliqueSearch | Iter. time |
|-----|------|------|--------|--------|--------------|-------------|-----------|
| 1   | 2    | 1    | 0      | 0      | 0            |             |           |
| 2   | 4    | 4    | 1      | 1      | 0            |             |           |
| 3   | 8    | 12   | 3      | 3      | 0            |             |           |
| 4   | 16   | 32   | 6      | 6      | 0            |             |           |
| 5   | 32   | 80   | 14     | 14     | 0            |             |           |
| 6   | 64   | 192  | 28     | 28     | 0            |             |           |
| 7   | 128  | 448  | 56     | 56     | 0            |             |           |
| 8   | 256  | 1,024| 112    | 112    | 0            |             |           |
| 9   | 512  | 2,304| 225    | 236    | 237          | 236         | 89 <1s.  |
| 10  | 1,024| 5,120| 456    | 472    | 493          | 472         | 903 1s.  |
| 11  | 2,048| 11,264| 922   | 952    | 1,005        | 952         | 212 1s.  |
| 12  | 4,096| 24,576| 1,862 | 1,904  | 2,029        | 1,904       | 605 1s.  |

Table 4: Result for hypercubes $Q_n$ with $n \leq 13$.

to extend these forests to larger ones by adding stable sets. We have shown that the procedure is effective on challenging families of bipartite graphs. Indeed, we have decreased most of the best-known upper bounds on $\phi(G)$, thus narrowing the gap to the best-known lower bounds. On four occasions we even managed to close this gap, which allows us to state that $\phi(B_5) = \phi(S_5) = 41$ and $\phi(B_6) = \phi(S_6) = 271$. The updated bounds for the tested graphs with a strictly positive gap are as follows, the old upper bounds being shown in parentheses:

$2017 \leq \phi(S_7) \leq 2018$ (2083)

$2017 \leq \phi(B_7) \leq 2036$ (2310)

$139 \leq \phi(\Gamma_{12}) \leq 157$ (158)

$225 \leq \phi(\Gamma_{13}) \leq 259$ (264)

$364 \leq \phi(\Gamma_{14}) \leq 426$ (439).

Most of the largest forests produced by our algorithm contain all vertices of one part of the bipartition $(X, Y)$ of the vertex set of $G$. Assuming $|X| \leq |Y|$, this has led us to fix $k_Y = 0$ or $k_Y = |Y|$, which helps speed up the algorithm since the moves from a solution $F$ to a solution $F' = F \oplus (u, v)$ are restricted to pairs $u, v$ of vertices that both belong to $X$. Such forests correspond to decycling sets fully contained in $X$. Notice however that it is easy to construct bipartite graphs $G = (X, Y, E)$ with no decycling set of minimum size fully contained in $X$ or $Y$. In the example of Figure 8, the only optimal decycling set contains two adjacent vertices $x$ and $y$ which therefore belong to different parts of the bipartition of the vertex set. Algorithm **LargeForest** is flexible enough to allow the discovery of any induced forest of $G$ (and therefore of any decycling set of $G$). Indeed, a forest $G[F^*]$ of $G$ can be generated with $k_X = |F^* \cap X|$ and $k_Y \leq |F^* \cap Y|$ : Algorithm **LargeForest** has to determine a set $F$ so that $F \cap X = F^* \cap X$, $F \cap Y \subseteq F^* \cap Y$ and $G[F]$ is a forest of $G$, and Algorithm **GreedyStable** can then extend $F$ to $F^*$ by adding the stable set $F^* \setminus F \subseteq Y$.

We have observed in Section 3.4 that knowledge of the structure of the considered bipartite graph can help speed up the proposed upper bounding procedure. For example, several authors have realized that optimal forests in $Q_n$ are unions of stars. With this assumption, we have been able to match the best-known upper bounds for $\phi(Q_n)$ ($n \leq 13$) in at most one second, while these graphs have up to 8,192 vertices. Hence, an analysis of the structure of the best-known induced forests of a bipartite graph can facilitate the generation of large induced forests.
Every iteration of Algorithm LargeForest requires $O(k_X(|X| - k_X) + k_Y(|Y| - k_Y))$ calls to Algorithm ForestExtension. In order to speed up the algorithm, we can fix an upper limit $k_{max}$ on $k_X$ and $k_Y$. If this limit has a too low value, it can prevent Algorithm LargeForest from generating an induced forest of $G$ of maximum order. For example, if $k_{max} = 0$, then the output $F^*$ of Algorithm LargeForest will be a stable set with max{$|X|$, $|Y|$} vertices. For the bubble sort graph $B_7$, this would lead to an upper bound of 2,520 on $\phi(B_7)$ while it is very easy to generate decycling sets of $B_7$ with less than 2,100 vertices. If an optimal forest has $x$ vertices in $X$ and $y$ vertices in $Y$, then $k_{max}$ should at least be equal to $\min\{x, y\}$. In summary, small values for $k_{max}$ make it possible to speed up Algorithm LargeForest, but finding the smallest value of $k_{max}$ which makes it possible to generate an optimal induced forest is a real challenge.

We want to mention that we did not seek to optimize the code by using structures making it possible to efficiently manage the fusion of connected components when adding a vertex to a forest, or their splitting when removing a vertex. This would certainly reduce the computing times somewhat. The aim of this article was to demonstrate that the proposed algorithm makes it possible to generate large induced forests in bipartite graphs and thus obtain small decycling sets.

Note finally that the smallest decycling sets generated by our algorithm for star graphs, bubble sort graphs, and Fibonacci cubes are archived online at www.gerad.ca/~alainh/decycling.html.

Acknowledgements

The author would like to thank Serge Bisaillon and Phil Herman for their help and support.

References


480 Hertz Decycling Bipartite Graphs


