Solution of a Conjecture of Vandell on Decycling Bipartite Tournaments by Deleting Arcs

Andreas Holtkamp (holtkamp@mathc.rwth-aachen.de)
Lehrstuhl C für Mathematik, RWTH Aachen University, 52056 Aachen, Germany

Lutz Volkmann (volkm@math2.rwth-aachen.de)
Lehrstuhl II für Mathematik, RWTH Aachen University, 52056 Aachen, Germany

Abstract

The *decycling index* of a digraph is the minimum number of arcs whose removal yields an acyclic digraph. The *maximum arc decycling number* $\overline{\nabla}'(m,n)$ is the maximum decycling index among all m-by-n *bipartite tournaments*. Recently, R.C. Vandell determined the numbers $\overline{\nabla}'(2,n)$, $\overline{\nabla}'(3,n)$, and $\overline{\nabla}'(4,n)$ for all positive integers n, as well as $\overline{\nabla}'(5,5)$. In this work we use a computer program to obtain $\overline{\nabla}'(5,6)$, $\overline{\nabla}'(6,6)$, and $\overline{\nabla}'(5,7)$, as well as some results on $\overline{\nabla}'(6,7)$ and $\overline{\nabla}'(5,8)$. In particular, $\overline{\nabla}'(6,6) = 10$, and this confirms a conjecture of Vandell.

Keywords: cycle; digraph; bipartite tournament; decycling index; maximum arc decycling number.
2000 MSC: 05C20

1 Terminology and introduction

By a *cycle* of a digraph we mean a *directed cycle.* The *decycling index* of a digraph D, denoted by $\nabla'(D)$, is the minimum number of arcs whose removal yields an acyclic digraph. An *m-by-n bipartite tournament* is an orientation of a complete bipartite graph $K_{m,n}$. By $\overline{\nabla}'(G)$ we denote the maximum decycling index among all orientations of a graph G. Therefore, $\overline{\nabla}'(m,n) := \overline{\nabla}'(K_{m,n})$ is the maximum decycling index of all m-by-n bipartite tournaments. Furthermore, the *in-degree* and *out-degree* of a vertex $v \in V(D)$ are denoted *id* v and *od* v.

The decycling index was studied for tournaments by Reid [1] and for bipartite tournaments by Vandell [2]. Vandell determined the numbers $\overline{\nabla}'(2,n)$, $\overline{\nabla}'(3,n)$, and $\overline{\nabla}'(4,n)$ for all positive integers n, as well as $\overline{\nabla}'(5,5)$. In this work we determine $\overline{\nabla}'(5,6)$, $\overline{\nabla}'(6,6)$, and $\overline{\nabla}'(5,7)$. Firstly, we find some lower and upper bounds for the mentioned cases, which lead to only two possible values for the maximum arc decycling number in each case. Secondly, we prove further degree conditions for the tournaments with high decycling index. Then, we use the computer program from the appendix to decide between the two possible values, where

the degree conditions help to reduce the runtime of the algorithm. Therefore, the algorithm enumerates all possible bipartite tournaments, checks if the resulting digraph matches the degree condition, and if necessary computes their decycling index. The following results of Vandell [2] are useful for our investigations.

Lemma 1. *For all positive integers m, s, and t,*

$$\overline{\nabla}'(m, s+t) \geq \overline{\nabla}'(m, s) + \overline{\nabla}'(m, t).$$

Theorem 1. $\overline{\nabla}'(2, n) = \lfloor \frac{n}{2} \rfloor$.

Theorem 2. $\overline{\nabla}'(3, n) = \lfloor \frac{2n}{3} \rfloor$.

Lemma 2. $7 \leq \overline{\nabla}'(4, 6) \leq \overline{\nabla}'(4, 7)$.

Theorem 3. *For $n \geq 2$,*

$$\overline{\nabla}'(4, n) = \begin{cases} \lfloor \frac{7n}{6} \rfloor - 1 & \text{if } n \equiv 1,\ 3 \pmod 6, \\ \lfloor \frac{7n}{6} \rfloor & \text{if } n \equiv 0,\ 2,\ 4,\ 5 \pmod 6. \end{cases}$$

Theorem 4. $\overline{\nabla}'(5, 5) = 6$.

Either deleting all arcs going into or out of an arbitrary vertex v, leaves a digraph without any cycle through v. Therefore, the following lemma is immediate.

Lemma 3. *Let T be an m-by-n bipartite tournament with partite sets X, Y with $|X| = m \geq 2$, $|Y| = n \geq 1$, and $v \in X$. Then*

$$\nabla'(T) \leq \min\{d^+(v), d^-(v)\} + \overline{\nabla}'(m-1, n).$$

This directly leads to the following result.

Corollary 1. *For all positive integers i, m and n with $m - i \geq 1$ we have*

$$\overline{\nabla}'(m, n) \leq \overline{\nabla}'(m-i, n) + i \cdot \left\lfloor \frac{n}{2} \right\rfloor.$$

Furthermore, by combining Theorem 2 and Corollary 1 we obtain a trivial upper bound for the general case.

Corollary 2. *For all positive integers m and n with $m \geq 3$ and $n \geq 2$ we have*

$$\overline{\nabla}'(m, n) \leq \left\lfloor \frac{2n}{3} \right\rfloor + (m-3) \left\lfloor \frac{n}{2} \right\rfloor.$$

To obtain a trivial lower bound for the general case, we can decompose one partite set of an m-by-n bipartite tournament into disjoint subsets of order 2 and one subset of order 3. Thus, using Lemma 1, Theorem 1 and Theorem 2 we obtain:

Corollary 3. *For all positive integers m and n with $3 \leq m \leq n$ we have*

$$\overline{\nabla}'(m, n) \geq \left\lfloor \frac{2n}{3} \right\rfloor + \left(\left\lfloor \frac{m-3}{2} \right\rfloor \right) \cdot \left\lfloor \frac{n}{2} \right\rfloor.$$

2 The 5-by-6 case

According to Lemma 2, we have $\overline{\nabla}'(4,6) \geq 7$, and so we deduce from Lemma 1 our first proposition.

Proposition 1. $\overline{\nabla}'(5,6) \geq 7$.

Proposition 2. $\overline{\nabla}'(5,6) \leq 8$.

Proof. Let $X = \{x_1, x_2, \ldots, x_5\}$ and $Y = \{y_1, y_2, \ldots, y_6\}$ be the partite sets of a 5-by-6 bipartite tournament T. Then $\min\{\mathrm{id}\, y_1, \mathrm{od}\, y_1\} \leq 2$. By Lemma 3 and Theorem 4 we obtain

$$\overline{\nabla}'(5,6) \leq \overline{\nabla}'(5,5) + 2 = 8.$$

□

The proof of Proposition 2 shows the next corollary immediately.

Corollary 4. *Let $X = \{x_1, x_2, \ldots, x_5\}$ and $Y = \{y_1, y_2, \ldots, y_6\}$ be the partite sets of a bipartite tournament T. If $\mathrm{id}\, y_i \leq 1$ or $\mathrm{od}\, y_i \leq 1$ for any $i \in \{1, 2, \ldots, 6\}$, then $\nabla'(T) \leq 7$.*

Using Corollary 4, Proposition 1 and the computer program in the appendix, we obtain the next result.

Theorem 5. $\overline{\nabla}'(5,6) = 7$.

3 The 6-by-6 case

Proposition 3. $\overline{\nabla}'(6,6) \geq 10$.

Proof. In view of Lemma 1 and Theorems 1 and 3, we obtain

$$\begin{aligned} \overline{\nabla}'(6,6) &= \overline{\nabla}'(6,4+2) \\ &\geq \overline{\nabla}'(6,4) + \overline{\nabla}'(6,2) \\ &= 7 + 3 = 10. \end{aligned}$$

□

Theorem 6. $\overline{\nabla}'(6,6) = 10$.

Proof. Let $X = \{x_1, x_2, \ldots, x_6\}$ and $Y = \{y_1, y_2, \ldots, y_6\}$ be the partite sets of a bipartite tournament T. Then there exists a vertex x_i, say x_1, such that $\mathrm{id}\, x_1 \leq 3$. By Lemma 3 and Theorem 5 we obtain

$$\overline{\nabla}'(6,6) \leq \overline{\nabla}'(5,6) + 3 = 10.$$

Now Proposition 3 leads to the desired result. □

Theorem 6 confirms a conjecture of Vandell [2].

	y_1	y_2	y_3	y_4	y_5	y_6	y_7
x_1	-	-	$\oplus$	-	-	$\oplus$	$\oplus$
x_2	-	$\oplus$	-	-	$\oplus$	-	$\oplus$
x_3	+	-	$\oplus$	+	-	-	-
x_4	+	+	-	$\ominus$	-	+	-
x_5	-	-	-	+	$\oplus$	-	-

Table 1: Tournament T_1. A +/- entry in line i and row j in the table specifies an arc $x_i \to y_j$ / $x_i \leftarrow y_j$. Removing the nine circled arcs decycles T_1.

4 The 5-by-7 case

Proposition 4. $\overline{\nabla}'(5,7) \geq 8$.

Proof. In view of Lemma 1 and Theorems 1 and 4, we obtain

$$\begin{aligned} \overline{\nabla}'(5,7) &= \overline{\nabla}'(5,5+2) \\ &\geq \overline{\nabla}'(5,5) + \overline{\nabla}'(5,2) \\ &= 6+2=8. \end{aligned}$$

□

Proposition 5. $\overline{\nabla}'(5,7) \leq 9$.

Proof. Let $X = \{x_1, x_2, \ldots, x_5\}$ and $Y = \{y_1, y_2, \ldots, y_7\}$ be the partite sets of a bipartite tournament T. Then $\min\{\mathrm{id}\, y_1, \mathrm{od}\, y_1\} \leq 2$. By Lemma 3 and Theorem 5 we obtain

$$\overline{\nabla}'(5,7) \leq \overline{\nabla}'(5,6) + 2 = 9.$$

□

Theorem 7. $\overline{\nabla}'(5,7) = 9$.

Proof. The computer program in the appendix shows that $\nabla'(T_1) = 9$ for the 5-by-7 bipartite tournament T_1 in Figure 1 and Table 1. Thus, Proposition 5 leads to

$$\overline{\nabla}'(5,7) = 9.$$

□

There are many further 5-by-7 bipartite tournaments T with the property $\nabla'(T) = 9$. In the next proposition we give some sufficient conditions for 5-by-7 bipartite tournaments T such that $\nabla'(T) \leq 8$.

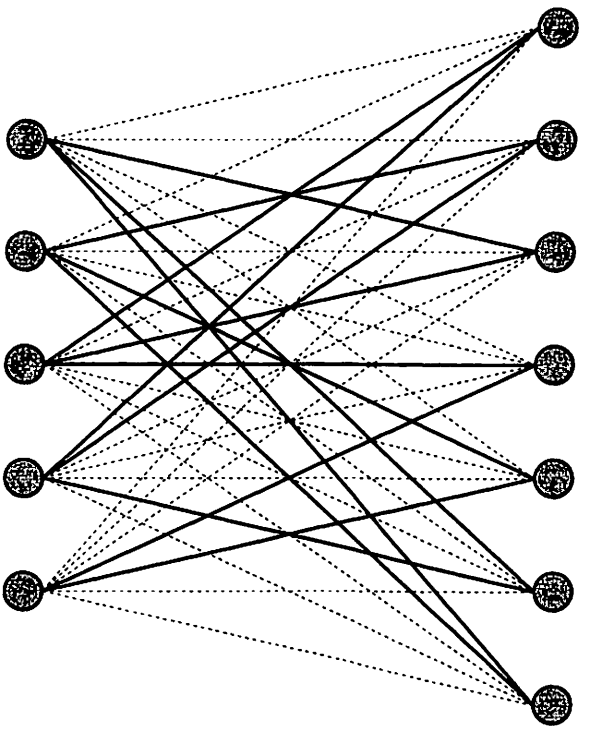

Figure 1: Drawing of tournament T_1. Arcs from left to right (right to left) are depicted as continuous (dashed) lines.

Proposition 6. *Let* $X = \{x_1, x_2, \ldots, x_5\}$ *and* $Y = \{y_1, y_2, \ldots, y_7\}$ *be the partite sets of a bipartite tournament* T.

If $\mathrm{id}\, y_i \leq 1$ *or* $\mathrm{od}\, y_i \leq 1$ *for any* $i \in \{1, 2, \ldots, 7\}$, *then* $\nabla'(T) \leq 8$.

If $\mathrm{id}\, x_i \leq 1$ *or* $\mathrm{od}\, x_i \leq 1$ *for any* $i \in \{1, 2, \ldots, 5\}$, *then* $\nabla'(T) \leq 8$.

If there are two distinct indices $i, j \in \{1, 2, \ldots, 5\}$ *such that* $\mathrm{id}\, x_i \leq 2$ *or* $\mathrm{od}\, x_i \leq 2$ *and* $\mathrm{id}\, x_j \leq 2$ *or* $\mathrm{od}\, x_j \leq 2$, *then* $\nabla'(T) \leq 8$.

Proof. If $\mathrm{id}\, y_i \leq 1$ or $\mathrm{od}\, y_i \leq 1$ for any $i \in \{1, 2, \ldots, 7\}$, then Lemma 3 and Theorem 5 lead to $\nabla'(T) \leq \overline{\nabla}'(5, 6) + 1 = 8$.

If $\mathrm{id}\, x_i \leq 1$ or $\mathrm{od}\, x_i \leq 1$ for any $i \in \{1,2,\ldots,5\}$, then Lemma 3 and Theorem 3 imply that $\nabla'(T) \leq \overline{\nabla}'(4,7)+1 = 8$.

Assume that there are two distinct indices $i,j \in \{1,2,\ldots,5\}$ such that $\mathrm{id}\, x_i \leq 2$ or $\mathrm{od}\, x_i \leq 2$ and $\mathrm{id}\, x_j \leq 2$ or $\mathrm{od}\, x_j \leq 2$. By Lemma 3 and Theorem 2 we obtain $\nabla'(T) \leq \overline{\nabla}'(3,7)+4 = 8$. □

5 The 6-by-7 case

Proposition 7. $\overline{\nabla}'(6,7) \geq 11$.

Proof. In view of Lemma 1 and Theorems 2 and 3, we obtain

$$\begin{aligned} \overline{\nabla}'(6,7) &= \overline{\nabla}'(6,4+3) \\ &\geq \overline{\nabla}'(6,4)+\overline{\nabla}'(6,3) \\ &= 7+4 = 11. \end{aligned}$$

□

Proposition 8. $\overline{\nabla}'(6,7) \leq 12$.

Proof. Let $X = \{x_1,x_2,\ldots,x_6\}$ and $Y = \{y_1,y_2,\ldots,y_7\}$ be the partite sets of a bipartite tournament T. Then $\min\{\mathrm{id}\, x_1, \mathrm{od}\, x_1\} \leq 3$. By Lemma 3 and Proposition 5 we obtain

$$\overline{\nabla}'(6,7) \leq \overline{\nabla}'(5,7)+3 \leq 9+3 = 12.$$

□

The proof of Proposition 8 and Lemma 3 lead immediately to the next propositions.

Proposition 9. *Let $X = \{x_1,x_2,\ldots,x_6\}$ and $Y = \{y_1,y_2,\ldots,y_7\}$ be the partite sets of a bipartite tournament T. If $\mathrm{id}\, x_i \leq 2$ or $\mathrm{od}\, x_i \leq 2$ for any $i \in \{1,2,\ldots,6\}$, then $\nabla'(T) \leq 11$.*

Proposition 10. *Let $X = \{x_1,x_2,\ldots,x_6\}$ and $Y = \{y_1,y_2,\ldots,y_7\}$ be the partite sets of a bipartite tournament T. If $\mathrm{id}\, y_i \leq 1$ or $\mathrm{od}\, y_i \leq 1$ for any $i \in \{1,2,\ldots,7\}$, then $\nabla'(T) \leq 11$.*

Proof. Assume, without loss of generality, $\mathrm{id}\, y_1 \leq 1$. Deleting the arc going into y_1 leaves a digraph without any cycle through y_1. According to Theorem 6, we obtain $\nabla'(T) \leq \overline{\nabla}'(6,6)+1 = 11$. □

6 The 5-by-8 case

Proposition 11. $\overline{\nabla}'(5,8) \geq 10$.

Proof. In view of Lemma 1 and Theorem 3, we obtain

$$\begin{aligned} \overline{\nabla}'(5,8) &= \overline{\nabla}'(5,4+4) \\ &\geq \overline{\nabla}'(5,4) + \overline{\nabla}'(5,4) \\ &= 5+5 = 10. \end{aligned}$$

□

Proposition 12. $\overline{\nabla}'(5,8) \leq 11$.

Proof. Let $X = \{x_1, x_2, \ldots, x_5\}$ and $Y = \{y_1, y_2, \ldots, y_8\}$ be the partite sets of a bipartite tournament T. Then $\min\{\mathrm{id}\, y_1, \mathrm{od}\, y_1\} \leq 2$. By Lemma 3 and Proposition 5, we obtain

$$\overline{\nabla}'(5,8) \leq \overline{\nabla}'(5,7) + 2 \leq 9 + 2 = 11.$$

□

The proof of Proposition 12 and Lemma 3 lead immediately to the next proposition.

Proposition 13. *Let $X = \{x_1, x_2, \ldots, x_5\}$ and $Y = \{y_1, y_2, \ldots, y_8\}$ be the partite sets of a bipartite tournament T. If $\mathrm{id}\, y_i \leq 1$ or $\mathrm{od}\, y_i \leq 1$ for any $i \in \{1, 2, \ldots, 8\}$, then $\nabla'(T) \leq 10$.*

7 The computer program

The computer program we used to determine the maximum arc decycling numbers $\overline{\nabla}'(m,n)$ for fixed integers m and n is programmed in C. In general, it uses three different algorithms to solve the problem. First, all possible m-by-n bipartite tournaments are generated. Second, for a fixed bipartite tournament T and a fixed integer s all possible subsets $S \subseteq E(T)$ of the arcs of T with cardinality s are generated. Third, for a fixed bipartite tournament T and an arc set S it is decided whether or not $T - S$ contains a cycle. Clearly, if every m-by-n bipartite tournament can be decycled by deleting an arc set of cardinality s, then $\overline{\nabla}'(m,n) \leq s$. On the other hand, if there exists an m-by-n bipartite tournament that cannot be decycled by removing an arc set of cardinality s, then $\overline{\nabla}'(m,n) > s$.

Furthermore, by using additional information like Corollary 4 and Proposition 6 it is possible to reduce significantly the processing time of the program, since

only a fraction of all possible 5-by-6 and 5-by-7 bipartite tournaments need to be computed. Corollary 4 for example states that a 5-by-6 bipartite tournament T with $\nabla'(T) > 7$ needs to have id $v \geq 2$ and od $v \geq 2$ for all vertices v of the partite set with cardinality six. Analogously, Proposition 6 determines that in a 5-by-7 bipartite tournament T with $\nabla'(T) > 8$ all vertices must have an in- and out-degree of at least two, and in the smaller partite set all but one vertex must have in- and out-degrees of at least three. Therefore, whether a generated tournament fits this regularity condition will be decided by a call of the function *bool in_out_degree_greater_equal(...)* of class *digraph*.

In the appendix the source code of our program for the 5-by-6 case is shown. The program types out a notification every time a tournament is found, which cannot be decycled by removing seven arcs. After the entire term some statistics on the computation will be displayed, showing how many 5-by-6 bipartite tournaments could be decycled by removing seven arcs. The program of the 5-by-7 case looks very similar and is therefore omitted. It only uses different assignments of a few constants and some minor changes on the regularity criteria due to Proposition 6 in comparison to Corollary 4.

Finally, let us consider the runtime of the algorithm above. The number of arcs in an m-by-n bipartite tournament is $m \cdot n$. Let d be the number of arcs to be deleted from these tournaments. Thus, $2^{m \cdot n}$ different m-by-n bipartite tournaments need to be computed, and for each of those digraphs all possible combinations of d arcs out of $m \cdot n$ are consecutively removed. Our algorithm that decides whether or not the remaining digraphs contain a directed cycle runs in $O(m \cdot (m+n)^2)$. For larger digraphs we could use topological sorting here, which runs in $O(n+m)$. But for the digraphs considered in this work our algorithm has an average runtime of about four times faster than topological sorting, because it makes use of the bipartite structure of the graph and it stops as soon as the first cycle is found. Altogether, the runtime of this algorithm can be estimated by $O(2^{m \cdot n} \cdot (m \cdot n)^d \cdot m \cdot (m+n)^2)$. By using the additional information from Corollary 4 and Proposition 6 we were in both cases able to significantly reduce the number of tournaments to be computed, in fact to obtain all possible 5-by-6 and 5-by-7 bipartite tournaments (up to isomorphism) we reduced the number of edges that need to be oriented from 30 to 23 and from 35 to 26, respectively. Of course, this directly decreases the runtime of our algorithm, and even for the remaining digraphs only a fraction of those match the exact regularity criteria tested after the generation of the 2^{23} and 2^{26} bipartite tournaments.

8 Conclusions

Due to the exponential complexity of the problem it is very hard to compute the maximum arc decycling numbers of m-by-n bipartite tournaments. As our program obtained the results on 5-by-6 and 5-by-7 bipartite tournaments in about 20 minutes (5-by-6 case) and about four hours (5-by-7 case), respectively, the computation of the 5-by-8 and 6-by-7 cases would each take more than a year of processing time on a common computer. In every case it is recommondable to look for results like Proposition 7-10 and Proposition 11-12 that immensely reduce the runtime.

By using parallelisation techniques and more sophisticated machines it would certainly be possible to obtain the next few maximum arc decycling numbers in reasonable time. But as m and n grow larger the complexity will become intractable, and it will be more and more difficult to find suitable lower and upper bounds for the maximum arc decycling numbers, thus, this approach becomes inapplicable. The more promising approach is now to make use of these results in a mathematical proof to obtain a general result on $\overline{\nabla}'(5,n)$ or $\overline{\nabla}'(6,n)$, and, if necessary for the proofs, compute the numbers $\overline{\nabla}'(5,8)$ and $\overline{\nabla}'(6,7)$ like described above.

References

[1] K.B. Reid, On sets of arcs containing no cycles in a tournament, Canad. Math. Bull. 12 (1969), pp. 261–264.

[2] R.C. Vandell, Decycling bipartite tournaments by deleting arcs, Ars Combin., to appear.

Appendices

Source code: *main.cpp*

```
#include <cstdlib>
#include <iostream>
#include <time.h>
#include "digraph.h"
#include <math.h>
```

```
// The program deals with orientations of a complete bipartite
// graph K(m,n). The question is, how many arcs need at least to
// be deleted from such an arbitrary digraph D[m][n] for the
// remaining digraph to be acyclic (free of any oriented cycles).

using namespace std;

int main(int argc, char *argv[])
{
        const int number_deleted_arcs=7;
        const int start_tournament=0;
        const int end_tournament=8388608; // 2^23=8388608

        digraph* tournament= new digraph(6,5,number_deleted_arcs);
        bool has_cycle;
        int n=0;
        int sum_regular_tournaments=0;
        int sum_of_not_decycled_tournaments=0;
        int abort=0;
        for (int i=(tournament->m_D-1);i>(tournament- >m_D
                        -number_deleted_arcs-1);--i) abort+=i;

        // Without loss of generality in order to gain all tournaments
        // of the 6-by-5 bipartite tournament with in- and out-degree
        // of at least two in the first partite set (down to isomorphism
        // according to Corollary 2.3), the following seven arcs are specified
        // with a fixed value.
        tournament->set_arc(1,1,1);
        tournament->set_arc(1,2,1);
        tournament->set_arc(1,3,0);
        tournament->set_arc(1,4,0);
        tournament->set_arc(1,5,0);
        tournament->set_arc(2,1,0);
        tournament->set_arc(2,5,1);

        cout << "Starting with tournament: " << start_tournament << endl;

// There are only 23 arcs undetermined. Therefore the number of
// different tournaments to look at is 2^23=8388608. For each
// tournament we will at first set the arcs of the digraph according to the
// number of the tournament. Afterwards we will check, if these tournaments
// fulfill the in- and out-degree of at least two in the partite set A.
// If it does, we will then try to decycle it by deleting seven arcs. Therefore we
```

```
// need to check 30 over 7 combinations of arcs, i.d. ~two million combinations.
// If we find a combination of arcs whose deletion decycles the current
// tournament, we can abort and continue looking at the next tournament.
for (int t_counter=start_tournament;t_counter<end_tournament;++t_counter)
{
        // Set the arcs of the tournament according to the number of the
        // tournament.
        for (int i=3;i<7;++i) for (int j=1;j<6;++j)
                {
                n=((i-3)*5)+j-1;
                tournament->set_arc(i,j,((t_counter%(((int)pow(((float)2),
                        ((float)(n+1))))))/(((int)pow(((float)2),((float)n))))));
                }
        tournament->set_arc(2,2,((t_counter%2097152)/1048576));
        tournament->set_arc(2,3,((t_counter%4194304)/2097152));
        tournament->set_arc(2,4,(t_counter/4194304));
        tournament->update_arcs(); // Update the arc sets.

// If the generated tournament has in- and out-degrees greater than two
// for vertices in the first partite set, then check whether or not
// it can be decycled by removing seven arcs.
if (tournament->in_out_degree_greater_equal(2,6))
{
        ++sum_regular_tournaments;

        // Remove the first seven arcs from the digraph
        int a[number_deleted_arcs];
        for (int i=0;i<number_deleted_arcs;++i) a[i]=i;
        tournament->delete_arcs(a);
        int sum_a=0;
        for (int i=0;i<number_deleted_arcs;++i) sum_a+=a[i];

        // Check if the remaining digraph is now decycled.
        has_cycle=tournament->has_cycle();

        // If it is not decycled, continue looking at all other possible
        // combinations of 7 arcs out of 30 to be deleted. I.e. ~2 million.
        while ((has_cycle) && (sum_a<abort))
                {

                // Restore the formerly deleted arcs.
                tournament->reset_arcs();
```

```
            // Compute the next combination of arcs to be deleted.
            bool added=false;
            int i=(number_deleted_arcs-1);
            while ((!added) && (i>=0))
                {
                if (a[i]<((tournament->m_D - number_deleted_arcs)+i))
                    {
                    added=true;
                    ++a[i];
                    int k=1;
                    for (int j=(i+1);j<number_deleted_arcs;++j)
                        { a[j]=(a[i]+k); ++k; }
                    }
                --i;
                }

            // Delete the above computed combination of arcs.
            tournament->delete_arcs(a);

            // Check if the remaining digraph is decycled.
            sum_a=0;
            for (int i=0;i<number_deleted_arcs;++i) sum_a+=a[i];
            has_cycle=tournament->has_cycle();
            }

        // Restore the formerly deleted arcs.
        sum_of_not_decycled_tournaments+=((int)has_cycle);
        tournament- >reset_arcs();
        if (has_cycle)
            {
            cout << "!Tournament not decycled by " << number_deleted_arcs
                << " arcs: " << t_counter << endl;
            cout << "Arc set:" << endl;
            for (int i=0;i<tournament->n_D;++i)
                for (int j=0;j<tournament->n_D;++j)
                    if (tournament->D[i][j])
                        cout << i << "->" << j << "; ";
                    cout << endl;
            }
        }
}

        // After the whole compution give out some statistics about the
```

```
        // results on how many of the tournaments could be decycled by
        // removing seven arcs.
        cout << sum_regular_tournaments;
        cout << " tournaments found with od x>=2 and id x>=2 for x in A.";
        cout << endl << (sum_of_not_decycled_tournaments);
        cout << " out of those not decycled by " << number_deleted_arcs
                                        << " removed arcs." << endl;
        cout << "Tournaments visited: " << start_tournament << " to "
                                        << end_tournament << endl;
        delete tournament;

        system("PAUSE");
        return EXIT_SUCCESS;
}
```

Source code: *digraph.h*

```
#ifndef DIGRAPH_H
#define DIGRAPH_H

#include <vector>

class digraph
{
    public:
        // class constructor
        digraph(int _n_A, int _n_B, int _deleted_arcs);
        // class destructor
        ~digraph();

        std::vector< std::vector<bool> > D; // adjacency matrix of the digraph
D
        std::vector<int> arcs;          // to adress the (numbered) arcs of D di-
rectly
        std::vector<int> deleted_arcs;  // to adress the deleted arcs of D directly
        int n_A;        // n(A) – order of partition set A
        int n_B;        // n(B) – order of partition set B
        int n_D;        // n(D) – order of the digraph D (no. of vertices)
        int n_sqr;      // n(D)^2 – for reducing no. of multiplications
        int m_D;        // m(D) – size of the digraph D (no. of arcs)
        int number_deleted_arcs;  // the number of arcs to be deleted from D
```

```
        // for description of function see "digraph.cpp"
        void set_arc(int x, int y, bool direction);
        void update_arcs();
        bool has_cycle();
        void delete_arcs(int* a);
        void reset_arcs();
        bool in_out_degree_greater_equal(int k, int vertices);
};

#endif // DIGRAPH_H
```

Source code: *digraph.cpp*

```
#include "digraph.h" // class's header file
#include <iostream>

using namespace std;

// class constructor
// Creates a complete bipartite (simple) digraph D[][] without any loops
// and with "int _n_A" and "int _n_B" being the orders of the partition
// sets A and B. After construction all arcs of D[][] are directed
// from partition set A to partition set B.
// Furthermore, the size of the subset of arcs that are to be
// deleted is determined by the argument "int _deleted_arcs", and
// the according arc set "deleted_arcs[..]" is initialized.
digraph::digraph(int _n_A, int _n_B, int _deleted_arcs)
{
        n_A=_n_A;
        n_B=_n_B;
        n_D=(n_A+n_B);
        n_sqr=(n_D*n_D);
        D.resize(n_D);
        for (int i=0;i<n_D;++i)
        D[i].resize(n_D);
        for (int i=0;i<n_D;++i) for (int j=0;j<n_D;++j)
                D[i][j]=((i<n_A)&&(j>=n_A));
        m_D=(n_A*n_B);
        arcs.resize(m_D);
        number_deleted_arcs=_deleted_arcs;
```

```
        deleted_arcs.resize(number_deleted_arcs);
}

// class destructor
digraph::~digraph()
{
}

// For x in 1,..,n_A and y in 1,..,n_B the function sets the arc
// from x to y in the digraph D[][] according to the given "direction".
// The old arc is overwritten. If "direction==true" then x- >y is the
// arc between x and y, and if "direction==false" then y- >x is.
void digraph::set_arc(int x, int y, bool direction)
{
  if (direction) {D[(x-1)][(y+n_A-1)]=1; D[(y+n_A-1)][(x-1)]=0;}
    else { D[(x-1)][(y+n_A-1)]=0; D[(y+n_A-1)][(x-1)]=1; }
}

// After the construction of the digraph or after some arcs
// of D[][] have changed, the function "update_arcs(..)" enumerates
// the arcs of D[][] within the vector "arcs[..]", so that arc
// number i can be directly adressed by the value given in "arcs[i]".
void digraph::update_arcs()
{
        int counter=0;
        for (int i=0;i<n_D;++i)
                for (int j=0;j<n_D;++j)
                        if ((D[i][j]) && (counter<m_D))
                                {
                                arcs[counter]=((n_D*i)+j);
                                ++counter;
                                }
}

// The function "has_cycle()" determines whether or not
// the bipartite digraph D[][] contains an oriented cycle
// and returns the according Boolean value.
bool digraph::has_cycle()
{
        bool cycle=false;
        bool mark1[n_D];
        bool mark2[n_D];
```

```
        int i,abort;
        if (n_A<=n_B)
                { i=0; abort=n_A-1; }
        else { i=n_A; abort=(n_A+n_B-1); }

        while ((i<abort) && (!cycle))
                {
                for (int k=0;k<n_D;++k)
                    { mark1[k]=false; mark2[k]=false; }
                mark1[i]=true;
                int j=0;
                int l=0;
                while ((l<(n_sqr)) && (!cycle))
                        {
                        if ((mark1[j]) && (!(mark2[j])))
                                {
                                for (int k=0;k<n_D;++k)
                                        if (D[j][k])
                                                {
                                                if (k==i) cycle=true;
                                                else mark1[k]=true;
                                                }
                                mark2[j]=true;
                                }
                        ++j; j=(j%n_D); ++l;
                        }
                ++i;
                }
        if (cycle) { return true;}
        else { return false; }
}

// The array "int* a" contains the numbers of the arcs
// that are to be deleted from D[][]. The deleted arcs
// are stored within the arc set "deleted_arcs[..]".
void digraph::delete_arcs(int* a)
{
        for (int i=0;i<number_deleted_arcs;++i)
                deleted_arcs[i]=arcs[a[i]];
        for (int i=0; i<number_deleted_arcs; ++i)
                D[(deleted_arcs[i]/n_D)][(deleted_arcs[i]%n_D)]=false;
}
```

```
// The arcs formerly deleted by the function "delete_arc(..)"
// and stored within the arc set "deleted_arcs[..]" are restored.
void digraph::reset_arcs()
{
        for (int i=0; i<number_deleted_arcs; ++i)
                D[(deleted_arcs[i]/n_D)][(deleted_arcs[i]%n_D)]=true;
}

// Determines whether or not the in- and out-degrees of the first
// "vertices" number of vertices in D[][] are greater or equal than
// the number "k". E.g. a call like
// "in_out_degree_greater_equal(2,n_A)" would check if the in-
// and out-degrees of the (first n_A) vertices of the first partition set
// are greater or equal two.
bool digraph::in_out_degree_greater_equal(int k, int vertices)
{
        int n=0;
        bool regular=true;
        while ((regular) && (n<vertices))
            {
            int row_sum=0;
            int column_sum=0;
            for (int i=0;i<n_D;++i)
                { row_sum+=D[n][i]; column_sum+=D[i][n]; }
            regular=((row_sum>=k) && (column_sum>=k));
            ++n;
            }
        return (regular);
}
```