Publication II

Lasse Kiviluoto, Patric R. J. Östergård, and Vesa P. Vaskelainen. 2010. Algorithms for finding maximum transitive subtournaments. Espoo, Finland: Aalto University School of Science and Technology. 15 pages. Helsinki University of Technology, Department of Communications and Networking, Report 5/2010. ISBN 978-952-60-3369-3. ISSN 1797-478X.

Algorithms for Finding Maximum Transitive Subtournaments

Lasse Kiviluoto*
Celtius Ltd, Pieni Roobertinkatu 11, 00130 Helsinki, Finland

Patric R. J. Östergård† and Vesa P. Vaskelainen‡§
Department of Communications and Networking
Aalto University, P.O. Box 13000, 00076 Aalto, Finland

Abstract

The problem of finding a maximum clique is a fundamental problem for undirected graphs, and it is a natural question to ask whether there are analogous computational problems for directed graphs. One such problem is that of finding a maximum transitive subtournament in a directed graph. A tournament is an orientation of a complete graph; it is transitive if the occurrence of the arcs xy and yz implies the occurrence of xz. Searching for a maximum transitive subtournaments in a directed graph D is equivalent to searching for a maximum induced acyclic subgraph in $\bar{D}$, which in turn is computationally equivalent to searching for a minimum feedback vertex set in $\bar{D}$. This paper discusses two basic backtrack algorithms and a Russian doll search algorithm for finding a maximum transitive subtournament.

Keywords: backtrack search, clique, directed acyclic graph, feedback vertex set, Russian doll search, transitive tournament.

*Supported by the Academy of Finland, Grant No. 100500.

†Supported in part by the Academy of Finland, Grants No. 107493, 110196, and 130142.

‡Supported by the Academy of Finland under Grant No. 107493 and by the Walter Ahlström Foundation. (Walter Ahlströmin säätiö)

§Corresponding author. E-mail: vesa.vaskelainen@tkk.fi

1 Introduction

The maximum clique problem has attracted a lot of attention along the years; see [2] for an extensive survey. A clique in a graph is a set of mutually adjacent vertices, and a maximum clique is a clique with the largest number of vertices. The following decision problem is NP-complete: Given an undirected graph G and an integer k as the input, determine whether there is a clique of size k in G. In any case, much effort has been put on developing algorithms for the maximum clique problem, since there are several important applications.

The maximum clique problem concerns undirected graphs; it is therefore natural to ask whether there are corresponding problems for directed graphs. For a directed graph, one may obviously look at the maximum clique problem for the underlying undirected graph or the undirected graph with an edge $\{u, w\}$ if and only if both uw and wu are arcs in the original graph. Cliques of the latter type are called *symmetric cliques*. However, both of these cases reduce to the undirected case.

In Section 2 we will use the problem of determining the capacity of communication channels to demonstrate one possible correspondence between certain computational problems in undirected and directed graphs. This leads to the problem of finding a maximum transitive subtournament in a directed graph, which is computationally equivalent to the problems of finding a maximum induced acyclic subgraph and a minimum feedback vertex set. Analogously, in the undirected case we have the maximum clique problem, which is computationally equivalent to the problems of finding a maximum independent set and a minimum vertex cover. Two basic backtrack search algorithms and a Russian doll search algorithm for the maximum transitive subtournament problem are discussed in Section 3. The performance of these algorithms is compared in computational experiments in Section 4.

2 Clique-Type Problems for Directed Graphs

In this section, we develop a correspondence between clique-type problems for undirected and directed graphs via the concept of capacity of certain communication channels and graphs. This application also accentuates the demand for practical exact algorithms for these problems.

Consider an undirected graph G with one vertex for each input symbol of a discrete communication channel and an edge between two vertices if and only if the two symbols cannot lead to the same output symbol (due to channel errors). The graph G is known as the *characteristic graph* of the channel. Alternatively, one may study $\bar{G}$, the *confusion graph* of the channel.

The clique number of $G = (V, E)$, denoted by $\omega(G)$, gives the largest set of symbols that can be used for error-free transmission of one symbol over the channel, and the amount of information is $\log_2 \omega(G)$. If information is transmitted in blocks of size n, then the information rate is

$$\frac{\log_2 \omega(G^n)}{n} = \log_2 \omega(G^n)^{1/n}$$

bits per transmission, where the (co-normal) power graph G^n has the vertex set $V^n = \{(u_1, u_2, \ldots, u_n) : u_i \in V\}$, and $(u_1, u_2, \ldots, u_n)$ and $(v_1, v_2, \ldots, v_n)$ are connected by an edge if and only if there is an i such that $\{u_i, v_i\} \in E$. The *zero-error capacity* [18] of the channel is

$$\sup_{n \geq 1} \log_2 \omega(G^n)^{1/n}$$

bits per transmission. When this problem is studied purely in the context of graphs, one ignores the logarithm and defines the zero-error capacity of G as

$$\sup_{n \geq 1} \omega(G^n)^{1/n}.$$

Gargaro, Körner, and Vaccaro [7] studied directed graphs in an analogous way. With $D = (V, A)$ a directed graph, the (co-normal) power graph D^n has the vertex set $V^n = \{(u_1, u_2, \ldots, u_n) : u_i \in V\}$, and there is an arc from $(u_1, u_2, \ldots, u_n)$ to $(v_1, v_2, \ldots, v_n)$ if and only if there is an i such that $u_i v_i \in A$. The (non-logarithmic) *Sperner capacity* of D is then

$$\sigma(D) := \sup_{n \geq 1} \omega_s(D^n)^{1/n},$$

where $\omega_s(D)$ is the size of the maximum symmetric clique.

A *transitive subtournament*, or *transitive clique*, in a directed graph is a set of vertices that can be listed $v_1, v_2, \ldots, v_m$ such that if $i < j$ then there is an arc $v_i v_j$; note that we do not care about possible arcs in the opposite direction. The size of a maximum transitive subtournament in D is denoted by $\omega_t(D)$. The importance of $\omega_t(D)$ in the study of Sperner capacity is that (see [7, 17] and their references)

$$\sigma(D) = \sup_{n \geq 1} \omega_t(D^n)^{1/n}$$

and as obviously $\omega_s(D) \leq \omega_t(D)$, determination of the size of a maximum transitive subtournament rather than that of a maximum symmetric clique for small powers of D may—and often does—lead to better lower bounds

on $\sigma(D)$. Concrete instances for the problem of determining the size of a maximum transitive subtournament can be found, for example, in [10].

We shall briefly look at some analogies with the undirected case. The following decision problem is NP-complete [6, 8]: Given a directed graph D and an integer k as the input, determine whether there is a transitive subtournament of size k in D.

A clique in an undirected graph G corresponds to an *independent set* in $\bar{G}$, but can something similar be said in the directed case? An answer to this question is provided by the following basic graph-theoretic result.

Theorem 1 *A transitive subtournament in D is an acyclic induced subgraph in $\bar{D}$, and vice versa.*

Proof. In $D = (V, A)$, consider an arbitrary set of vertices $V' \subseteq V$ that is not a transitive subtournament. Equivalently, for any order $v_1, v_2, \ldots, v_m$ of the vertices in V', there are i and j such that $i < j$ and $v_i v_j \notin A$. In the complementary graph $\bar{D} = (V, \bar{A})$, for any order of the vertices in V', there are i and j such that $i < j$ and $v_i v_j \in \bar{A}$. The well-known fact that a directed graph has a topological ordering if and only if it is acyclic proves that the induced subgraph in $\bar{D}$ is cyclic. This completes the proof as we have equivalences in all steps. □

If S is an independent set in an undirected graph $G = (V, E)$, then $S' = V \setminus S$ is a vertex cover of G. For directed graphs we have the following analogy: If S induces an acyclic graph, then $S' = V \setminus S$ covers—that is, contains a vertex of—every cycle of G and is called a *feedback vertex set*.

Finally, we would like to point out that various aspects of clique-type problems, for example, considering weighted graphs, carries over naturally to the problems for directed graphs discussed here. A final example: The chromatic number of an undirected graph is the smallest number of independent sets that partition the vertex set, whereas the dichromatic number [14] of a directed graph is the smallest number of acyclic induced subgraphs that partition the vertex set.

3 Algorithms

Bearing in mind the analogy between the discussed problems for undirected and directed graphs, it is perhaps surprising that in the undirected case virtually all computational studies have been on cliques and independent sets (rather than on vertex covers), whereas for directed graphs most studies have been on feedback vertex sets (rather than on transitive subtournaments

and induced acyclic subgraphs). Since the solutions to these two versions of the problems are complements to each other, it seems reasonable to focus on the version that has a small(er) solution. Consequently, if a graph is almost acyclic, then one should indeed search for a minimum feedback vertex set. However, there are many applications where the alternative formulation is to be preferred, including the application discussed in Section 2. The authors feel that the problem of finding maximum transitive subtournaments (alternatively, induced acyclic subgraphs) should receive more attention in the literature.

Only a handful of papers have been published on exact algorithms for the aforementioned problems for directed graphs. (Note that exact algorithms can be studied and compared in two ways: via theoretical bounds and via their practical performance. We consider algorithm within the latter framework.) Also here the literature is biased toward feedback vertex sets [1, 12, 15, 19]. Funke and Reinelt [5] indeed consider induced acyclic subgraphs, but they concentrate on a variant of the problem where arcs have weights and the aim is to maximize the total arc weight of the induced graph.

We shall here present three algorithms for finding a maximum transitive subtournament: two basic backtrack algorithms and a Russian doll search algorithm. The similarities between these and the algorithms published in [4] and [16] are not incidental. The order of the vertices of the input graph has a (sometimes even significant) impact on the computation times of these algorithms; this issue is discussed further in the Section 4.

We start by a basic backtrack algorithm, Algorithm 1, that builds up a transitive subtournament so that if vertex v is added on level i—this information is saved in the variable a_i—then there must be an arc vw to add vertex w on level j whenever $j > i$. The set of vertices w such that there is an arc vw is denoted by $\Gamma^+(v)$. The parameters U and s of the procedure BASIC1(U, s) stand for the set of remaining vertex candidates and the size of the current transitive subtournament, respectively. The algorithm is invoked with BASIC1$(V, 0)$, where V is the set of all vertices, and the initial value of the global variable *record* is 0.

Algorithm 1 builds up a transitive tournament in the order of its vertices, starting from the vertex that has all other vertices of the tournament as successors. One drawback of Algorithm 1 is that having considered a certain vertex, chosen in line 8, we cannot remove it from U since it can also be considered at later stages when building up the transitive tournament. The next two algorithms to be considered do not have this drawback; on the other hand, in these algorithms we do need a test that determines whether an induced subgraph is a transitive tournament. Actually, because of this test, we consider the complementary graph instead and build up induced acyclic

Algorithm 1 First Basic Backtrack Algorithm

```
procedure BASIC1(U: set, s: integer)
 1: if |U| = 0 then
 2:    if s > record then
 3:       record ← s
 4:       Save the current solution a_1, a_2, ..., a_s
 5:    end if
 6:    return
 7: end if
 8: for i ← 1, 2, ..., |U| do
 9:    U' ← U ∩ Γ+(u_i); The elements of U are denoted by u_1, u_2, ....
10:    if s + 1 + |U'| > record then
11:       a_{s+1} ← u_i
12:       BASIC1(U', s + 1)
13:    end if
14: end for
end procedure
```

subgraphs (Theorem 1); throughout the rest of this section, the maximum induced acyclic subgraph setting is therefore assumed.

Although updating the set of vertex candidates in line 9 of Algorithm 1 is a rather straightforward operation, careful implementation of this step does have some impact on the overall performance as it lies in the core of the algorithm. A thorough consideration of such core steps of backtrack algorithm was carried out by Lam *et al.* [11, 20] in their work on proving nonexistence of projective planes of order 10; in fact, they gave reasons for algorithms to act differently on different levels of the search tree (the levels being split into three regions).

The next algorithms to be considered build up an induced acyclic subgraph, adding the vertices in the order they occur in the directed graph D. Maintaining a set of vertices that do not introduce cycles when added to a solution set is a much more complicated operation than the update operation for Algorithm 1. Consequently, as there is not one obvious way of carrying out this operation, care has to be taken in the evaluation of algorithms to distinguish between the impact of this core part of the algorithm from the impact of the overall algorithm.

For each of the algorithms that build up an induced acyclic subgraph, we consider two different ways of maintaining partial solutions that are induced acyclic subgraphs. In one version (a) it is checked for each added vertex whether it leads to an induced cycle together with the vertices so far in the

solution set; this can be done by breadth-first search (BFS) or depth-first search (DFS) in time $O(|V| + |A|)$. In another version (b) we maintain a set U of vertices that, together with the partial solution, do not form induced cycles. In version (b), updating U after adding a vertex to the solution set is a nontrivial operation. We shall now elaborate briefly on this operation.

When a vertex v is added to the solution set B, we can delete a vertex $u \in U$ whenever there exist vertices $v', v'' \in B$ such that there is a path from v' through v to v'' in B, and arcs $v''u$ and uv' in the original graph D. The latter information can be precalculated into a table that for each triple u, v', v'' of vertices tells whether there are arcs $v''u$ and uv'. With $\Theta(|V|^3)$ bits for this table, $\Theta(|V|^2)$ bits for each u, the question can be answered in constant time. On the other hand, by finding—with BFS or DFS—all vertices V' in B from which there is a directed path to v (v itself is in this set) as well as all vertices V'' to which there is a directed path from v (v is also in this set), we get the desired pairs $(v', v'') \in (V', V'')$. The random instances considered in Section 4 are not constrained by the large space requirement of the auxiliary table. Anyway, other data structures can be utilized should the space requirement be an issue, trading speed for memory.

To test $(v', v'') \in (V', V'')$ against the information about paths $v''uv'$, there are two possibilities. If the number of pairs $(v', v'') \in (V', V'')$ is large, then one may form a string with $\Theta(|V|^2)$ bits and 1s for each pair occurring; this string can be immediately tested against the precalculated strings of the same length. On the other hand, with very few pairs, it is faster to test each pair separately. In the current work we used only the latter approach. Note also that when a certain pair has been tested, then it need not be tested among descendants in the search tree; this information is maintained in the search tree.

With the above mentioned versions (a) and (b), we consider Algorithm 2, which is a basic backtrack algorithm, and Algorithm 3, which a Russian doll search algorithm. To save space, we only give a detailed presentation of one version for each algorithm: version (a) for Algorithm 2 and version (b) for Algorithm 3. In all versions of these two algorithms, we handle the easy case of cycles of length 2 in a direct way through $T(v)$, $v \in V$, which denotes the set of vertices $w \in V$ for which not both vw and wv are arcs.

Algorithm 2 is invoked with BASIC2$(V, 0)$. The global variable *record* has initial value 0. The other variables and parameters are as in Algorithm 1.

Algorithm 3 is a Russian doll search algorithm. A *Russian doll search algorithm* [21] can in general terms be described as an algorithm for solving a problem with n variables through n subproblems, where the first subproblem includes just the nth variable, the ith subproblem the last i variables, and where the solutions of the subproblems are used for pruning in later subprob-

lems. Russian doll search has been applied to the maximum clique problem in [16].

In Algorithm 3, which is invoked with RUSSIANDOLL, the definition $S_i = \{v_i, v_{i+1}, \ldots, v_n\}$ gives the vertex sets of the subproblems. The size of a maximum induced acyclic subgraph in the subgraph induced by S_i is saved in the array $c[i]$. Note that $c[i-1] = c[i]$ or $c[i-1] = c[i]+1$. If a solution corresponding to the latter equality is found then a subproblem has obviously been solved; the variable *found* makes immediate interruption possible. The function *reduce* takes care of reducing the set of candidate vertices in U after a new vertex is added to the solution (in a way the updating operation of the version (b) which is described earlier). The other variables and parameters are as in Algorithm 1.

Algorithm 2 Second Basic Backtrack Algorithm, Version (a)

```
procedure BASIC2(U: set, s: integer)
 1: if |U| = 0 then
 2:    if s > record then
 3:       record ← s
 4:       Save the current solution a_1, a_2, ..., a_s
 5:    end if
 6:    return
 7: end if
 8: while U ≠ ∅ do
 9:    if s + |U| ≤ record then
10:       return
11:    end if
12:    i ← min{j : v_j ∈ U}
13:    U ← U \ {v_i}
14:    a_{s+1} ← v_i
15:    if a_1, a_2, ..., a_{s+1} is acyclic then
16:       BASIC2(U ∩ T(v_i), s + 1)
17:    end if
18: end while
end procedure
```

Algorithm 3 Russian Doll Search Algorithm, Version (b)

```
procedure RDS(U: set, s: integer)
 1: if |U| = 0 then
 2:    if s > record then
 3:       record ← s
 4:       Save the current solution a_1, a_2, ..., a_s
 5:       found ← TRUE
 6:    end if
 7:    return
 8: end if
 9: while U ≠ ∅ do
10:    if s + |U| ≤ record then
11:       return
12:    end if
13:    i ← min{j : v_j ∈ U}
14:    if s + c[i] ≤ record then
15:       return
16:    end if
17:    a_{s+1} ← v_i
18:    U ← U \ {v_i}
19:    U' ← reduce(U, a_1, ..., a_{s+1})
20:    RDS(U', s + 1)
21:    if found = TRUE then
22:       return
23:    end if
24: end while
end procedure
procedure RUSSIANDOLL
25: record ← 0
26: for i = n, n − 1, ..., 1 do
27:    found ← FALSE
28:    RDS(S_i ∩ T(v_i), 1)
29:    c[i] = record
30: end for
end procedure
```

4 Computational Results

In this section we use computational experiments to compare Algorithms 1, 2, and 3, the two latter in versions (a) and (b). Indeed, these algorithms were designed with the aim of optimizing practical performance, and belong to a class of algorithms less amenable to a formal performance analysis [13]. Recall that we are dealing with an optimization version of an NP-complete problem.

The algorithms were implemented in C++ and evaluated using random graphs. Graphs with 50 to 1000 vertices were generated with different arc probabilities—for each ordered pair of vertices, there is an arc with a given probability p. It is important to note that for Algorithm 1 we consider such a random graph D, but for Algorithms 2 and 3 we consider its complement $\bar{D}$. The results are presented in Table 2. We are not aware of any earlier work against which these results could be compared—earlier computational results either consider different versions of the problem or instances with transitive subtournaments (induced acyclic subgraphs) of size close to the order of the graph.

One detail that has been omitted so far is the impact of vertex orderings on the performance of the algorithms. For clique algorithms, the importance of this detail was recognized in the early work by Carraghan and Pardalos [4]. It is therefore no surprise that the choice of ordering turns out to have a crucial impact also for the type of algorithms considered in the current study.

Vertex orderings for clique-type problems are heuristics, many of which tend to order the vertices (in increasing or decreasing order) according to how likely they are to occur in a maximum solution. The best orders are not always intuitive and some experimentation is generally needed to find a good heuristic (for the graphs considered). Basic orderings considered here are those based on the vertex degrees in the underlying undirected graph (U), the outdegrees of the vertices in the directed graph (O), and the indegrees of the vertices in the directed graph (I). The vertices can be ordered ascending ($\nearrow$) or descending ($\searrow$). Also the case with no reordering is considered (NONE).

Thinking in the terms of $\bar{D}$, it seems reasonable that a large number of pairs v', v'' such that there are arcs $v''u$ and uv' make it less likely for u to be in a maximum induced acyclic subgraph (recall that this is related to the number of elements in the table used in version (b) of the algorithms). Therefore we also consider the ordering (P) given by the product of the indegree and the outdegree of the vertices in the complementary graph.

Russian doll search algorithms tend to perform well when the elements of an optimal solution are spread out evenly over the set of all elements. For undirected graphs and the maximum clique problem, this is achieved

by coloring the vertices and grouping the vertices that belong to each color class. Also in the case of induced acyclic subgraphs of $\bar{D}$, we may get a similar ordering (C) by grouping the vertices into symmetric cliques; in other words, these groups are independent sets of the underlying graph of D. Since the heuristic is only relevant for Russian doll search algorithms and not for basic backtrack search, it was only carried out for Algorithm 3.

To compare the different orderings, we utilized the average ranks ranking method [3]. For each parameter pair n, p in Table 2, five instances were generated and tested with each ordering. Orderings were given a rank number, starting from 1, with respect to their performance, and the average of all runs was calculated for each ordering. The results are shown in Table 1. The best ordering for each algorithm is in bold and used in the computations leading to Table 2.

Table 1: Ranking of permutations

Ordering	A1	A2(a)	A2(b)	A3(a)	A3(b)
U$\nearrow$	5.73	**1.77**	**1.56**	**3.64**	**2.89**
U$\searrow$	4.20	8.36	7.97	7.85	8.15
O$\nearrow$	6.41	3.12	3.25	4.51	4.28
O$\searrow$	**3.67**	6.51	6.52	7.20	7.37
I$\nearrow$	4.69	3.13	3.27	4.85	4.13
I$\searrow$	4.91	6.71	6.48	8.23	8.36
P$\nearrow$	6.06	2.17	2.27	3.76	3.95
P$\searrow$	4.29	8.16	8.65	7.92	8.93
C$\nearrow$				3.84	4.85
C$\searrow$				6.56	5.97
NONE	5.04	5.07	5.03	7.64	7.11

In Table 2, we show the average time for 100 random graphs of each parameter set. Once again, recall that for Algorithms 2 and 3 the computations are carried out on the complementary graph. The relatively large number of graphs considered is a consequence of the large standard deviation of computational times which are presented in Table 3. These times are in CPU seconds for a 2-GHz PC with Linux operating system. Also the average size of a solution is given.

Table 2 illustrates that the presented algorithm for finding maximum transitive subtournaments is faster than the one for finding maximum induced acyclic subgraphs in the complement in the region where the solution size is approximately smaller than 20. The Russian doll search approach for finding maximum induced acyclic subgraphs is faster than (the backtrack

Table 2: Evaluation of the algorithms

$\|V\|$	p	max size	A1	A2(a)	A2(b)	A3(a)	A3(b)
50	0.5	10.04	0.01	0.02	0.03	0.01	0.02
50	0.6	12.71	0.06	0.13	0.14	0.07	0.08
50	0.7	16.42	1.73	1.36	1.12	0.64	0.56
50	0.8	22.37	573.54	17.35	11.12	7.76	5.46
50	0.9	33.04	>1000	69.81	71.08	25.62	30.38
100	0.5	12.40	0.57	2.85	2.82	1.73	1.60
100	0.6	15.91	16.41	71.39	41.45	41.92	23.13
100	0.7	21.13	>1000	>1000	>1000	>1000	886.29
200	0.3	9.04	0.18	0.90	1.77	0.59	1.18
200	0.4	11.33	3.13	19.98	24.00	14.24	16.93
200	0.5	14.54	109.00	835.27	537.31	580.27	367.41
500	0.2	8.03	0.99	4.78	29.61	3.38	21.75
500	0.3	10.67	28.52	220.61	669.65	153.36	475.96
1000	0.1	6.55	0.91	2.69	138.55	2.04	104.66
1000	0.2	9.02	49.70	203.24	>1000	148.67	>1000

Table 3: Sample standard deviation for results in Table 2

$\|V\|$	p	max size	A1	A2(a)	A2(b)	A3(a)	A3(b)
50	0.5	0.59	0.00	0.01	0.01	0.01	0.01
50	0.6	0.56	0.03	0.05	0.06	0.03	0.04
50	0.7	0.72	1.24	0.75	0.61	0.39	0.36
50	0.8	0.98	954.64	12.41	9.06	6.96	4.84
50	0.9	1.26	-	74.74	73.70	31.81	34.56
100	0.5	0.49	0.13	0.58	0.73	0.59	0.63
100	0.6	0.47	4.53	20.27	13.16	15.70	10.51
100	0.7	0.54	-	-	-	-	483.27
200	0.3	0.20	0.02	0.13	0.19	0.09	0.21
200	0.4	0.47	0.51	2.70	4.53	3.91	5.36
200	0.5	0.50	20.30	126.32	126.57	159.51	125.60
500	0.2	0.17	0.10	0.15	1.62	0.39	2.72
500	0.3	0.47	4.00	19.93	79.06	42.65	137.26
1000	0.1	0.50	0.09	0.15	8.90	0.52	27.84
1000	0.2	0.14	1.50	4.52	-	16.39	-

search) Algorithm 2 and the overall fastest for dense graphs, for all instances considered in this work.

As the described algorithm resembles the maximum clique algorithm presented in [16], it is natural to ask whether ideas from other maximum clique algorithms could be implemented in algorithms for finding maximum transitive subtournaments. However, most clique algorithms utilize various types of colorings, and it is not clear how to carry over this concept (as for efficient computing) to directed graphs.

We conclude this paper with the remark that the considered algorithms have indeed been applied to instances of the applications discussed in Section 2; an exhaustive determination of the Sperner capacity of small directed graphs has been carried out in [9].

Acknowledgements

The authors thank Brendan McKay for useful discussions.

References

[1] Berghammer, R., Fronk, A.: Exact computation of minimum feedback vertex sets with relational algebra. Fund. Inform. 70, 301–316 (2006)

[2] Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M.: The maximum clique problem. In: Du, D-Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization, Supplement Vol. A, pp. 1–74. Kluwer, Dordrecht (1999)

[3] Brazdil, P.B., Soares, C.: A comparison of ranking methods for classification algorithm selection. In: López de Mántanas, R., Plaza, E. (eds.) Machine Learning: ECML 2000, LNCS 1810, pp. 63–74. Springer, Berlin (2000)

[4] Carraghan, R., Pardalos, P.M.: An exact algorithm for the maximum clique problem. Oper. Res. Lett. 9, 375–382 (1990)

[5] Funke, M., Reinelt, G.: A polyhedral approach to the feedback vertex set problem. In: Cunningham, W.H., McCormick, S.T., Queyranne, M. (eds.) Integer Programming and Combinatorial Optimization, LNCS 1084, pp. 445–459. Springer, Berlin (1996)

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

[7] Gargano, L., Körner, J., Vaccaro, U.: Qualitative independence and Sperner problems for directed graphs. J. Combin. Theory Ser. A 61, 173–192 (1992)

[8] Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum, New York (1972)

[9] Kiviluoto, L., Östergård, P.R.J., Vaskelainen, V.P.: Sperner capacity of small digraphs. Adv. Math. Commun. 3, 125–133 (2009)

[10] Körner, J., Pilotto, C., Simonyi, G.: Local chromatic number and Sperner capacity. J. Combin. Theory Ser. B 95, 101–117 (2005)

[11] Lam, C.W.H., Thiel, L.H., Swiercz, S.: A computer search for a projective plane of order 10. In: Deza, M.M., Frankl, P., Rosenberg, I.G. (eds.) Algebraic, Extremal and Metric Combinatorics, pp. 155–165. Cambridge University Press, Cambridge (1988)

[12] Lloyd, E.L., Soffa, M.L.: On locating minimum feedback vertex sets. J. Comput. System Sci. 37, 292–311 (1988)

[13] McGeoch, C.C.: Experimental algorithmics. Communications of the ACM 50(11), 27–31 (2007)

[14] Neumann-Lara, V.: The dichromatic number of a digraph. J. Combin. Theory Ser. B 33, 265–270 (1982)

[15] Orenstein, T., Kohavi, Z., Pomeranz, I.: An optimal algorithm for cycle breaking in directed graphs. J. Electronic Testing 7, 71–81 (1995)

[16] Östergård, P.R.J.: A fast algorithm for the maximum clique problem. Discrete Appl. Math. 120, 197–207 (2002)

[17] Sali, A., Simonyi, G.: Orientations of self-complementary graphs and the relation of Sperner and Shannon capacities. Europ. J. Combin. 20, 93–99 (1999)

[18] Shannon, C.E.: The zero-error capacity of a noisy channel. IRE Trans. Inform. Theory 2, 8–19 (1956)

[19] Smith, G.W., Walford, R.B.: The identification of a minimal feedback vertex set of a directed graph. IEEE Trans. Circuits and Systems 22, 9–14 (1975)

[20] Thiel, L.H., Lam, C.W.H., Swiercz, S.: Using a CRAY-1 to perform backtrack search. In: Kartashev, L.P., Kartashev, S.I. (eds.) Supercomputing '87: Supercomputer Design, Performance Evaluation and Performance Education, Vol. 3, pp. 92–99. International Supercomputing Institute, St. Petersburg Fla. (1987)

[21] Verfaillie, G., Lemaître, M., Schiex, T.: Russian doll search for solving constraint optimization problems. In: Proc. 13th National Conference on Artificial Intelligence (AAAI-96) pp. 181–187. AAAI Press, Menlo Park (1996)