Acta Universitatis Sapientiae

Informatica

Volume 4, Number 1, 2012

Sapientia Hungarian University of Transylvania
Scientia Publishing House

Antal Bege
(1962–2012)

Our colleague, friend, editor-in-chief of Acta Universitatis Sapientiae, associate professor Antal Bege passed away unexpectedly on March 22, 2012. He was only 49.

After finishing his studies in Mathematics at Babeş-Bolyai University in Cluj he became a teacher in his former school in Miercurea Ciuc. After the regime change in 1989 he joined the Faculty of Mathematics and Computer Science, at Babeş-Bolyai University. He worked there for almost two decades, then went over to Sapientia Hungarian University of Transylvania, Department of Mathematics and Informatics in Târgu-Mureş in 2008. This is where he became the head of the department and the editor-in-chief of the academic journal Acta Universitatis Sapientiae. Naturally, he did his best in all these qualities.

Among his research interests we can mention Number Theory (arithmetical functions), Nonlinear Analysis and Discrete Mathematics. He published 13 textbooks and monographs both in Hungarian and Romanian, as well as a lot of scientific papers.

He was extremely evenhanded person, appreciated by all his colleagues and students, a man of poise and an eternal stayer. With a terrible feeling of pain and loss, we say goodbye to our friend. We shall treasure his memory.

Editorial Board

Contents

Acta Univ. Sapientiae, Informatica, **4,** 1 (2012) 7–16

Concept of the abstract program

Tibor GREGORICS
Eötvös Loránd University, Faculty of Informatics
Budapest
email: `gt@inf.elte.hu`

Abstract. The aim of this paper is to alter the abstract definition of the program of the theoretical programming model which has been developed at Eötvös Loránd University for many years in order to investigate methods that support designing correct programs. The motivation of this modification was that the dynamic properties of programs appear in the model. This new definition of the program gives a hand to extend the model with the concept of subprograms while the earlier results of the original programming model are preserved.

1 Introduction

It is a well-known aspiration to exclusively use methods which guarantee the correctness of the developed program with respect to the problem [1, 5, 6] posed, that is, what makes it essential to find abstract mathematical definition of programming notions. At Eötvös Loránd University, a relational model of programming which treats the most important, fundamental notions of programming—problem, program, state space, variable, data type, etc.—in a uniform and consistent way [2, 3, 7] has been built for thirty years.

The base of the abstract definitions of programming notions is the state space and their tool is the mathematical relation. The problem, for example, is a homogeneous binary relation over the state space that maps from the possible initial states to the appropriate goal states. The program is defined as

Computing Classification System 1998: F.3.1
Mathematics Subject Classification 2010: 68N30
Key words and phrases: programming model, state space, program constructions, specification, correctness

all of its executions, so it can be described by the relation that maps from any state to the executions starting from one, where an execution is the sequence of the states concerned. The effect of a program is also a homogeneous binary relation where its domain contains the states from which the program surely terminates, i.e. the executions starting from these states are finite, and maps to states where these executions stop. The definition of the solution (when a program is said to solve a problem) establishes the main connection between the relations of the problem and the effect of the program. This theoretical approach makes it possible to investigate the concept and correctness of several kinds of program-designer methods such as program synthesis, analogous programming, etc.

The main characteristic of this relational model is the special static point of view in which the concept of the program is a mapping and not a working, nonetheless it is originally dynamic. The advantage of this static point of view is the simple definition of the solution. Although it is not self-evident if the number of the components of the state space of a problem and that of a program are different. To prove the correctness, in this case, the program and the problem must be transformed into a new common state space [4].

In the present model however, the state space is persistent, i.e. all variables of the program are global and static (the scope and life time of the variables involve the whole program), the so-called local variables cannot be created and destroyed during an execution. In absence of local variables, the concept of real subprograms cannot be introduced into our programming model. (Only macros can be defined.) Namely, the power of a subprogram is what permits creating new local auxiliary variables as well as parameter variables, and it is not supported by the present concept of the program. It is not a serious problem if simply the correctness is wanted to be proved but it is a great difficulty if our aim is to design a program. Unfortunately, the programming method, which is based on our programming model to synthetize correct programs, cannot produce subprograms with parameter variables, so their creation remains in the sphere of the implementation instead of the designing. It is an additional disadvantage that, without the concept of subprograms, recursive calling cannot be used in our programming model.

The aim of this paper is to remedy this shortcoming of the definition of the program, and to preserve the earlier results of the programming model.

2 State space

All definitions of the programming model are based on the state space. The concept of the state space has already been interpreted in several ways. For many people, the state space is a model of a von Neumann type computer; others, e.g. Dijkstra [1], associate this notion with the problem to be solved, where a state is a compound of the values of the main data types. So, the program is "outside" of the state space operating on it. In our programming model, this second meaning is used. In [2], the notion of the state space is a Cartesian product of the value set of data types. The only mistake of this obvious definition is that it imposes an order on the components. In [3], where the ***state space*** is a direct product, this mistake is repaired.

Here it is the definition of the direct product and the notions related to it.

Let I be a finite set named as index set. Let A_i's $(i \in I)$ be arbitrary sets. The set $\underset{i \in I}{\times} A_i ::= \{x : I \to \bigcup_{i \in I} A_i | \forall i \in I : x(i) \in A_i\}$ is the direct product of the sets A_i $(i \in I)$.

The A_i's are the components of the direct product $\underset{i \in I}{\times} A_i$. The number of the components is finite. The direct product is the empty set if the set I is empty.

Let $A = \underset{i \in I}{\times} A_i$ and $B = \underset{j \in J}{\times} B_j$ be direct products where I and J are finite sets, A_i's $(i \in I)$ and B_j's $(j \in J)$ are arbitrary sets. The A and B are equivalent if there exists a bijection $\nu : I \to J$ so that $\forall i \in I : A_i = B_{\nu(i)}$. In other words, B is the renamed A.

If $J \subseteq I$ and $\forall j \in J : B_j = A_j$ then B is the subspace of A, i.e. $B \leqslant A$.

The function $pr_B : A \to B$ is a projection if $B \leqslant A$ and $\forall a \in A : pr_B(a) = a|_J$. Not only one state but a set or sequence of states can also be projected if it is done one by one. If $J = \{j\}$ then the function pr_B is named as j variable and j is the variable name.

3 The new concept of the program

The most important notion of the programming model is the program. In our explanation, the program is not a collection of some statements that can be executed on a computer. The statements could only describe a program but the program is the complex of its executions. An execution is a sequence of states. A program, by definition, can always begin, i.e. at least one execution has to start from each state. If several executions start from the same start state, it means that the program is non-deterministic: nobody knows which execution will happen.

In the original programming model, every state of the progam belongs to the same state space. Now we are going to permit that the state space can be permanently changed; the inner states of the executions may have got new components.

The other novelty of our new definition is the idea of the ***base state space*** of the program. The first state (start state) of all executions and the last, if the execution is finite, are in this base state space.

Before giving the formal definition of the program, some notions must be introduced. Let H^{**} denote the set of all finite and infinite sequences of the elements of set H. H^{∞} includes the infinite sequences; H^{*} contains the finite ones. So, $H^{**} = H^{*} \cup H^{\infty}$ and $H^{*} \cap H^{\infty} = \emptyset$. The length of the sequence $\alpha \in H^{**}$ is $|\alpha|$, the value of which is ∞ if the sequence is infinite.

Definition 1 *Let A be the so-called base state space and $\bar{A}$ be the set of all states which belong to the state spaces B whose subspace is A, i.e. $\bar{A} = \bigcup_{A \leqslant B} B$. The relation $S \subseteq A \times \bar{A}^{**}$ is a **program** over A, if*

1. $\mathcal{D}_S = A$ *(domain of S),*
2. $\forall \alpha \in \mathcal{R}_S \cap \bar{A}^{*} : \alpha_{|\alpha|} \in A$ *(the last state of the finite executions of the range of S),*
3. $\forall a \in A$ *and* $\forall \alpha \in S(a) : |\alpha| \geqslant 1$ *and* $\alpha_1 = a$.

The variables of the base state space are the ***base variables***; the other variables are the ***auxiliary variables*** of the program.

The concept of the program allows to create and destroy new components (variables) during an execution, so the state space changes dynamically. All new components have to be destroyed at the termination, at the very latest, but the base variables should never be removed. The current state always contains the components of the base state space.

The following three definitions will outline that the base state space of any program is denoted arbitrarily. The base variables of a program can be extended, restricted or renamed without changing the essence of the program.

The base variables of any program can be renamed easily without the execution of the program is changed. However, if the new name of a base variable would be identical to an auxiliary variable's name then this auxiliary variable is given a new, unique name.

Definition 2 *Let the statespace* A *and* B *be equivalent, i.e. if* $A = \bigtimes_{i\in I} A_i$ *and* $B = \bigtimes_{j\in J} B_j$ *then there exists a bijection* $\nu : I \to J$ *so that* $\forall i \in I : A_i = B_{\nu(i)}$*. Let* S *be a program over the state space* A*. Let the name of the auxiliary variables of* S *be* K *(*$K \cap I = \emptyset$*) and a bijection* $\mu : K \to L$ *so that* $L \cap J = \emptyset$*. The program over* B *is called the* ***variable renaming of S onto B*** *if its executions are identical to the executions of* S*, but in their all states, the variable name* $i \in I$ *is replaced by* $\nu(i)$ *and the variable name* $k \in K$ *is replaced by* $\mu(k)$*.*

Not only can the name of the variables be changed but also the number of the components of the base state space: this state space can be extended with new variables, or some base variables can be degraded to auxiliary varibles.

The next definition shows that the base state space can be extended with a new component. The states of the executions can be extended with this new component if it is not the auxiliary variable of the original program. Otherwise if this component was an auxiliary variable, then it becomes a base variable; it should not be created and destroyed it, its life expands over the whole execution.

Definition 3 *Let* S *be a program over the state space* $A = \bigtimes_{i\in I} A_i$ *and the variable name* k *where* $k \notin I$*. (This* k *may denote one of the auxiliary variables of* S *or a totally new variable.) Let* C *be the statespace* $\bigtimes_{i\in I\cup\{k\}} A_i$*. The* ***extension of S onto C*** *is the program (denoted by* S'*) whose base state space is* C *and for all* $c \in C$ *:*

$$\begin{aligned} S'(c) = \{\gamma \in \overline{C}^{**} \mid \exists \alpha \in S(pr_A(c)) : |\alpha| = |\gamma| \wedge \forall i \in [2..|\gamma|] : \\ \text{if } k \notin \mathcal{D}_{\alpha_i} \text{ then } \gamma_i|_I = \alpha_i \text{ and } \gamma_i(k) = \gamma_{i-1}(k) \\ \text{if } k \in \mathcal{D}_{\alpha_i} \text{ then } \gamma_i = \alpha_i\}. \end{aligned}$$

It is easy to generalize this definition in case the state space A is a subspace of the state space C.

The following definition shows how a base variable can be degraded to an auxiliary variable: the first step creates it and the last step destroys it.

Definition 4 *Let* S *be a program over the state space* $A = \bigtimes_{i\in I} A_i$ *and let* C *be the subspace of A. The* ***restriction of S onto C*** *is the program (denoted by* $\overline{S}$*) whose base state space is* C *and for all* $c \in C$*:*

$$\begin{aligned} \overline{S}(c) = \{ <c, \alpha, pr_C(\alpha_{|\alpha|})> \in \overline{C}^* | \alpha \in S(a) \cap \overline{A}^* \text{ where } c = pr_C(a) \} \\ \cup \{ <c, \alpha> \in \overline{C}^\infty | \alpha \in S(a) \cap \overline{A}^\infty \text{ where } c = pr_C(a) \}. \end{aligned}$$

The base state space of a program can be renamed, extended or restricted several times, thus the base state space can be totally transformed. But these transformations can change only the base state space and not the program. The essence of the executions of the program remains the same. The renamed, extended and restricted program hardly differs from its original version.

Definition 5 *Two programs are **identical** if they can transform into the same program, using extensions, restrictions and renaming.*

4 Conclusions

We have introduced a new concept of the program. Now let us observe what kinds of effects this modification has on the original programming model.

4.1 Concept of dynamic program

The new concept of the program smuggles the dynamic property of the program back to the static programming model [2, 3] developed at Eötvös Loránd University. Certainly, the fact that a program can create and destroy auxiliary variables is not a brilliant discovery. The difficulty of our investigation was to find out how this dynamic property can be embedded into the programming model described in the static point of view.

The most important element of our concept is the base state space. It can be seen as the interface of the program. It determines the components through which the program can communicate, and can make contact with its environment. Only the base variables can get value from outside before the program starts, and their value can be asked after the termination.

The program becomes very flexible because its base state space can be changed easily. This property is reflected in the fact that the same program can possess different interfaces depending on the problem to be solved. The aim of a program can be changed by altering its base state space whereas its operation cannot be changed.

4.2 Concept of solution

Despite the above modification, the definition of the solution can be preserved [2, 3, 7]. The only thing that must be done is to redefine the concept of the ***effect*** of the program because the definition of the solution relies on it.

Now we will repeat all definitions that are important to describe the concept of solution.

Let the problem generally be a relation $F \subseteq A \times A$ where A is a state space. Let S be a program over A. The relation $p(S) \subseteq A \times A$ is the effect of S if

1. $\mathcal{D}_{p(S)} = \{a \in A \mid S(a) \subseteq A^*\}$
2. $\forall a \in \mathcal{D}_{p(S)} : p(S)(a) = \{b \in A \mid \exists \alpha \in S(a) : \alpha_{|\alpha|} = b\}$.

The program S is said to solve the problem F if

1. $\mathcal{D}_F \subseteq \mathcal{D}_{p(S)}$
2. $\forall a \in \mathcal{D}_F : p(S)(a) \subseteq F(a)$

The definition of the ***solution*** supposes that the state space of the problem is identical to the base state space of the program. As the base state space can be chosen arbitrarily, the program can be extended, restricted or renamed in order that its base state space is identical to the state space of the problem. Since programs of this kind are identical, if one of them can solve the problem, so can the others.

Moreover, if the effects of the programs S_1 and S_2 are identical over all common base state spaces ($p(S_1) = p(S_2)$) and if one of them can solve a problem, so can the others. In this case these programs are called ***equivalent***. This relation is reflexive, symmetrical and transitive, thus it is an equivalence relation. Consequently, if one program belonging to an equivalence class can solve a problem, then every program derived from this class can also solve it.

In the original programming model, the definition of the solution must be generalized [4] when the state spaces of the problem and the program are different. Now, our new concept is used to avoid this. It is enough to fit the base state space of the program to the state space of the problem and transform the program onto this common state space.

4.3 Earlier results

The original programming model contains many important results that can be used in the verification of programs. It is apparent that these results are not based directly on the concept of the program; except for the effect of the program. The definition of the effect of the program has not been changed because the new definition of the program fixes that the start state and the end state of the finite execution are in the same state space (that is, the base state space). Accordingly, all earlier results, namely Dijkstra's weakest precondition [1], the theorem of the specification or the derivation rules [1, 3] of the program constructions [3] are used in unalterable forms.

Certainly, in the definitions of program constructions, the components (programs and conditions) which are the parts of the construction have to be on the same base state space. Accordingly, before making one of the constructions, this common base state space has to be agreed on.

4.4 Subprograms

The concept of real subprograms, which have got parameter variables, could not be introduced into the original programming model because the program has got an unvarying state space, thus local variables cannot be created during the execution, so parameter variables, which are local variables, cannot be used. Now, the situation is changed. In our new programming model, the concept of the subprograms may be defined.

First and foremost the subprogram is a program. It can be executed independently; it has got own base state space. Its only speciality is that it can be built into another program. (This is the host program.)

Each program, including subprograms, is equivalent to an assignment. This assignment is appropriate to identify the subprogram that is equivalent to it. Accordingly, in an arbitrary program description language, a subprogram can be denoted with this assignment. The variables of this assignment, which are the formal parameters, form the base state space of the subprogram. The variables at the left-hand side of the assignment are the output parameter variables; the ones at the right-hand side are the input parameter variables. (For simplicity's sake, we shall restrict our consideration to assignments where a parameter variable does not occur more than once.)

This assignment can be also used as a calling statement in the description of the host program. Certainly, in the calling statement, the parameter variables can be substituted by actual parameters (arguments), which are the expressions (often, they are only variables) of the host program. An output variable cannot be changed by an expression more general than a variable. The number and the type of the formal and current parameters must be the same. At this calling statement, the execution of the host program is interrupted, the control switches over the subprogram, the values of current parameters are given to the formal parameter variables, and the subprogram is executed. After the termination of the subprogram, the values of the output parameter variables are recopied to the appropriate current parameter variables. (Here, two kinds of parameter passing have been defined: passing by value and passing by value-result but other passing methods may be introduced.)

The base state space of the subprogram is the interface between the subprogram and the host program. At the beginning of the subprogram, the variables of this base state space will get their initial values from the current state of the host program. At the end of it these variables give their values to the variables of the host program. The parameter variables and other local variables of the subprogram are created when the subprogram is called and destroyed at the termination of the subprogram.

In addition, the usage of subprograms permits recursive callings because a subprogram can call itself. After each calling, the parameter and the local variables of the subprogram are created again and again without the error message "out of memory" because the memory of the abstract program is infinite.

During planning, it is convenient that all the current variables of the host program are treated as global with respect to the subprogram that is called by the host program. The applied programming style and the facilities of the selected programming language determine if these global variables can be used directly in the subprogram or not. Obviously, their use has to be forbidden if we want to guarantee the independency of the subprogram.

We can make difference between the calling statement and the calling expression. In the latter case, only the right-hand side of the subprogram's head appears inside an expression of the host program, which contains actual parameters instead of formal parameter variables. After the termination, the result value of the left-hand side of the head is given back to the place of the calling expression in the host program.

Certainly, the abstract program description language which is used for planning has to be extended with the notation of the subprogram and of its calling including the connection between the actual parameters and the parameter variables.

To sum up the introduction of the concept of the subprograms makes it possible to design subprograms during planning and not only during implementation.

Acknowledgements

This paper is supported by the European Union and co-financed by the European Social Fund (grant agreement no. TÁMOP 4.2.1/B-09/1/KMR-2010-0003).

I would like to thank Ákos Fóthi for his pieces of useful advices.

References

[1] E. W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, New York, 1976. ⇒7, 9, 14

[2] Á. Fóthi, A Mathematical Approach to Programming, *Ann. Univ. Sci. Budapest. Sect. Comput.*, **9** (1988) 105–114. ⇒7, 9, 12

[3] Á. Fóthi, *Bevezetés a programozáshoz*, ELTE Eötvös Kiadó. 2005 (in Hungarian). ⇒7, 9, 12, 14

[4] Á. Fóthi, Z. Horváth, J. Nyéky-Gaizler, A Relational Model of Transformation in Programming, *Proc. 3th International Conference on Applied Informatics*, Eger-Noszvaj, Hungary, August 24–28. 1997. ⇒8, 13

[5] C. A. Hoare, Proof of correctness of data representations, *Acta Inform.*, **1** (1972) 271–281. ⇒7

[6] D. Gries, *The Science of Programming*, Springer, Berlin, 1981. ⇒7

[7] Workgroup on Relational Models of Programming – Á. Fóthi et al., Some concepts of a Relational Model of Programming, L. Varga ed., *Proc. 4th Symposium on Programming Languages and Software Tools*, Visegrád, Hungary, June 8-14, 1995, 434–446. ⇒7, 12

Received: March 14, 2012 • Revised: April 14, 2012

Acta Univ. Sapientiae, Informatica, **4,** 1 (2012) 17–32

Sum of squares representation for the Böttcher-Wenzel biquadratic form

Lajos LÁSZLÓ
Department of Numerical Analysis, Eötvös Loránd University, Hungary
email: `laszlo@numanal.inf.elte.hu`

Abstract. We find the minimum scale factor, for which the nonnegative Böttcher-Wenzel biquadratic form becomes a sum of squares (sos). To this we give the primal and dual solutions for the underlying semidefinite program. Moreover, for special matrix classes (tridiagonal, backward tridiagonal and cyclic Hankel matrices) we show that the above form is sos. Finally, we conjecture sos representability for Toeplitz matrices.

1 Introduction

The Böttcher-Wenzel inequality ([1], [2], [3], [7], [6]) states that for real square matrices P, Q the biquadratic form

$$BW \equiv 2\left(||P||^2||Q||^2 - \mathrm{trace}^2(P^TQ)\right) - ||PQ - QP||^2 \qquad (1)$$

is nonnegative, with the Frobenius norm used. Replacing the factor 2 by $2+\gamma_n$, it is natural to ask for the minimum γ_n such that

$$(2+\gamma_n)\left(||P||^2||Q||^2 - \mathrm{trace}^2(P^TQ)\right) - ||PQ - QP||^2$$

is a sum of (polynomial) squares. We answer this question in Theorem 1 by showing that the minimum value is $(n-2)/2$.

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 90C22, 15A45, 65F15
Key words and phrases: sum of squares, Böttcher-Wenzel biquadratic form, semidefinite programming, strict complementarity

For simplicity, we use one-subscript notation for the entries of P and Q, described by means of the "small" index matrix

$$\mathrm{IND} = ((i-1)n+j)_{i,j=1}^{n}.$$

Then P and Q can be generated by vectors p and q of dimension $m = n^2$ as

$$P(i,j) = p\,(\mathrm{IND}(i,j)),\ 1 \le i \le j \le n.$$

Introducing an index matrix will be especially useful in Sections 3 to 5, where tridiagonal, backward tridiagonal, cyclic Hankel and general Toeplitz matrices will be investigated. For these special cases we prove (for Toeplitz matrices: conjecture) that the corresponding BW form is a sum of squares (sos).

It is quite odd that although (real) Hankel matrices are symmetric, thus normal, hence nonnegativity easily follows [1], this does not imply that a sos form also exists (except if $n = 3$, Example 9). On the other hand, Toeplitz matrices are usually not normal, yet the corresponding BW form is sos, at least according to our well-grounded Conjecture 15 at the end of the paper.

2 The case of general matrices

Let P, Q be arbitrary $n \times n$ real matrices with entries

$$P = (p_{(i-1)n+j})_{i,j}^{n}, \quad Q = (q_{(i-1)n+j})_{i,j}^{n},$$

as indicated above. (Notice that we use this indexing technique for simplicity.) It turns out [5] that the above forms depend only on the variables

$$z_{i,j} = p_i q_j - q_i p_j,\ 1 \le i < j \le n,$$

i.e. on the skew symmetric matrix

$$Z = pq^{T} - qp^{T}$$

of order n^2, a benefit of including the term $\mathrm{trace}^2(P^T Q)$. Indeed, we have

$$||Z||^2 = 2\left(||P||^2||Q||^2 - \mathrm{trace}^2(P^T Q)\right),$$

and all entries of the commutator $[P,Q]$ obviously are linear forms of the $z_{i,j}$-s.

Let us formulate the primal and dual semi-definite programming problems (see e.g. in [8]) for the eigenvalue optimization:

$$\min \{\mathrm{tr}(CX) : \ X \geq 0, \ \mathrm{tr}(A_i X) = 0, \ 1 \leq i \leq M, \ \mathrm{tr}(X) = 1\} \quad (\mathrm{Primal})$$
$$\max \{y_{M+1} : \ S \equiv C - \sum_{t=1}^{M} y_t A_t - y_{M+1} I \geq 0\} \quad (\mathrm{Dual})$$

where C, S, X, A_t and the identity $I = I_N$ are all real symmetric Nth order matrices, C and $(A_t)_1^M$ are given, the primal matrix X, the dual (slack) matrix S and vector y are the solutions of the program, $\mathrm{tr}(AB) \equiv \mathrm{trace}(AB)$ denotes the scalar product of the symmetric matrices A and B, and $\geq$ stands for the semi-definite ordering: $A \geq B$ iff $A - B$ is positive semi-definite.

The quantities $z_{i,j}$ will play the role of 'candidate monomials' (better to say, differences, and hereafter called candidates) with ordering

$$z = (z_{1,2}, \ z_{1,3}, \ z_{2,3}, \ z_{1,4}, \ \ldots, \ z_{1,n}, \ \ldots, z_{n-1,n})^T.$$

The indices can be read from the "big" index matrix

$$\mathrm{POS} = \begin{pmatrix} 0 & 1 & 2 & 4 & 7 & \ldots \\ \cdot & 0 & 3 & 5 & 8 & \ldots \\ \cdot & \cdot & 0 & 6 & 9 & \ldots \\ \cdot & \cdot & \cdot & 0 & 10 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

of order n^2 to be

$$(i,j) \sim k \equiv i + \frac{(j-1)(j-2)}{2}, \quad 1 \leq i < j \leq n^2.$$

Note that $z_{i,i} = 0$ for all i, and that the entries below the diagonal are omitted due to $z_{i,j} = -z_{j,i}$, enabling us to reduce the number of unknowns. Also note that IND is related to P and Q, while POS is connected with Z.

As an example, we give the biquadratic form BW as a quadratic form of the quantities $(z_{i,j})$ for $n = 3$. Observe that $\|Z\|^2 = \|Z\|_F^2 = 2\sum_{1\leq i<j\leq 9} z_{i,j}^2$.

Example 1. For $n = 3$ the objective takes the form

$$\begin{aligned} BW &= \|Z\|^2 - (z_{2,4} + z_{3,7})^2 - (z_{1,2} + z_{2,5} + z_{3,8})^2 - (z_{1,3} + z_{2,6} + z_{3,9})^2 \\ &- (-z_{1,4} - z_{4,5} + z_{6,7})^2 - (-z_{2,4} + z_{6,8})^2 - (-z_{3,4} + z_{5,6} + z_{6,9})^2 \\ &- (-z_{1,7} - z_{4,8} - z_{7,9})^2 - (-z_{2,7} - z_{5,8} - z_{8,9})^2 - (-z_{3,7} + z_{6,8})^2. \end{aligned}$$

(Note that the form BW can be thought of as a function of the matrices (P, Q), the vectors (p, q), the matrix Z, or of the vector z.) We give now all the (quadratic) relations holding for the variables $(z_{i,j})_{1\le i<j\le n^2}$ as

$$z_{i,j}\, z_{k,l} + z_{i,l}\, z_{j,k} - z_{i,k}\, z_{j,l} = 0, \quad 1 \le i < j < k < l \le n^2. \tag{2}$$

These easily checked relations define $M = \binom{n^2}{4}$ symmetric constraint matrices A_t, each having exactly 6 nonzero (off-diagonal) entries. For instance, equation

$$z_{2,3}\, z_{4,5} + z_{2,5}\, z_{3,4} - z_{2,4}\, z_{3,5} = 0$$

defines an A_t with nonzero entries in positions $(3, 10)$, $(8, 6)$, $(5, 9)$ and their transposes, see the matrix POS. Now we can state our main theorem.

Theorem 1 *The minimum value of* γ_n*, for which (1) is a sum of squares is*

$$\gamma_n = \frac{n-2}{2}.$$

Proof. We give the optimal primal and dual solutions and describe the main characteristics of the optimal dual matrix. Since the objectives coincide, the strong duality theorem yields the desired result.

By fixing the order of the variables $(z_{i,j})$ above, matrix C is uniquely determined. To get the (slack) matrix $S = C - \sum y_t A_t$, we use the following strategy. Note that we not only give the set (A_t) of active constraints (as e.g. when taking the half Newton-polytope), but also give their coefficients (y_t).

Strategy A. Assume the commutator $[P, Q]$ contains an entry $(z_{i,j}+z_{k,l}+\dots)$ with i, j, k, l distinct and $i < j,\ k < l$. Then the quadratic form $z^T C z$ associated with C necessarily contains a term $2\,z_{i,j}\, z_{k,l}$. We 'halve' this term, and leave one $z_{i,j}\, z_{k,l}$ unchanged as is, while apply for the other term the basic quadratic relation (2). By using the correct sign, this defines a constraint A_t and the corresponding dual variable y_t for some t. Finally, let $y_{m+1} = -\frac{n-2}{2}$.

Now we give the obtained primal and dual solutions. In view of the quite combinatorial character of the problem, we do not detail each block, instead we give some explanations and important cross-references (control sums) and for matrix S we provide Table 1. with all essential informations.

The Primal Problem

Before defining the optimal primal matrix X, we note that its rank is $\binom{n}{2}$. For indices $(i, j) : 1 \le i < j \le n$ we define the vectors $v_{i,j}$ of dimension $\binom{n^2}{2}$

to have $4(n-1)$ nonzero coordinates (four 2's and $4(n-2)$ ± 1's) using row_i, row_j, column_i and column_j of the index matrix IND, cf. Example 2 below.

Next we form the matrix of these vectors

$$V = [v_{1,2},\ v_{1,3},\ v_{2,3},\ \dots,\ v_{n-1,n}],$$

and define matrix $X_0 = V V^T = \sum v_{i,j} v_{i,j}^T$ with the following properties:

X_0 is a symmetric matrix of order $N = \binom{n^2}{2}$ and rank $\binom{n}{2}$. The $v'_{i,j}$'s are orthogonal with norm square $\|v_{i,j}\|^2 = 4 \cdot 4 + 4(n-2) \cdot 1 = 4(n+2)$. The trace of X_0 is

$$\operatorname{tr}(X_0) = \sum_{i<j} \operatorname{tr}(v_{i,j} v_{i,j}^T) = \sum_{i<j} \|v_{i,j}\|^2 = 4(n+2)\binom{n}{2},$$

thus by defining

$$X = \left(4(n+2)\binom{n}{2}\right)^{-1} X_0$$

we get a trace 1 matrix. The eigenvalues of X_0 are $4(n+2)$, those of X are $\binom{n}{2}^{-1}$ (hence $\binom{n}{2}^{-1} X$ is a projection). The $v_{i,j}$'s are also eigenvectors of C:

$$C v_{i,j} = (2-n)/2\ v_{i,j}.$$

Furthermore we have

$$\operatorname{tr}(CX_0) = \sum_{i<j} \operatorname{tr}(C v_{i,j} v_{i,j}^T) = \sum_{i<j} v_{i,j}^T C v_{i,j} = \frac{2-n}{2} \sum_{i<j} \|v_{ij}\|^2,$$

and finally, the primal objective equals

$$\operatorname{tr}(CX) = \frac{\operatorname{tr}(CX_0)}{\operatorname{tr}(X_0)} = \frac{2-n}{2}. \tag{3}$$

The Dual Problem

The matrix $S = C - \sum y_t A_t$ resulting from Strategy A is positive semi-definite and decomposes into some blocks given in Table 1. (Observe that all eigenvalues and diagonal entries of $2S$ are integer – a reason for the factor 2.)

Here we list the important facts and control sums concerning the blocks of S, and in Example (3) we give further hints for understanding the construction.

ROW-control: an element in the last column of row i is the scalar product of row 1 and row i. For instance, the number of zero eigenvalues—the defect of S—equals to $\binom{n}{2}2 + 1(n-1) = n^2 - 1$.

1	No of blocks	$\binom{n}{2}$	1	$3\binom{n}{4}$	$\binom{n}{2}$	Total
2	Block sizes	$6n-8$	$\binom{n}{2}$	4	1	
3	Eig=0	2	n-1	–	–	n^2-1
4	Eig=4	1	–	–	–	$n(n-1)/2$
5	Eig=n	$2n-4$	$\binom{n-1}{2}$	1	–	$(n^2-1)(n-2)(n+4)/8$
6	Eig=$n+2$	$3n-5$	–	2	2	$(n^2-1)(n-2)(n+4)/4$
7	Eig=$n+4$	$n-3$	–	1	–	$n(n-1)(n^2+n-2)/8$
8	Eig=$2n+2$	1	–	–	–	$n(n-2)/2$
9	Diag	$n(+2)$	$n-2$	$n+2$	$n+2$	$(n^3-n)(n^2+2n-4)/2$

Table 1: Decomposition of the matrix "2S"

EIG-control: the last column (the number of eigenvalues, rows 3 to 8) sums up to $\binom{n^2}{2}$, the order of the matrix $2S$.

DIAG-control: The sum of the elementwise products of row 1, 2 and 9,

$$\binom{n}{2}\Big(2n*n+4(n-2)(n+2)\Big)+1\binom{n}{2}(n-2)+3\binom{n}{4}4(n+2)+\binom{n}{2}1(n+2)$$

equals to $\binom{n}{2}(n+1)(n^2+2n-4)$, the trace of $2S$. (In the blocks of order $6n-8$ there are $2n$ diagonal elements "n", and $4(n-2)$ diagonal elements "$(n+2)$".)

TRACE-control: the trace of the coefficient matrix C equals

$$\operatorname{tr}(C)=2\binom{n^2}{2}-n(n-1)-(n^2-n)n=n(n-1)^2(n+1).$$

The first subtrahend comes from the diagonal of the commutator $[P,Q]$, the second from their off-diagonal elements. Due to $\operatorname{diag}(S)=\operatorname{diag}(C)+\gamma_n I$, the connection between the traces of matrices C and S is

$$\operatorname{tr}(2S)=2\Big(\operatorname{tr}(C)+\frac{n-2}{2}\binom{n^2}{2}\Big).$$

The number of all constraints is $\binom{n^2}{4}$, while that of active constraints equals

$$n\binom{n-1}{2}+n(n-1)\Big(\binom{n-2}{2}+2(n-2)\Big)=\binom{n}{2}(n^2-4)\equiv 3(n+2)\binom{n}{3}.$$

Here the first term is associated with the main diagonal of $R\equiv[P,Q]$ (by virtue of $z_{i,i}=0$ there are only $n-1$ terms in $R(i,i)$), while the rest comes from the off-diagonal of the commutator R (where there always are two terms for which the basic relations do not apply, see the Example 2).

Thus S is positive semi-definite with defect $n^2 - 1$, and its eigenvalues range in the interval $[0, n+1]$. To sum up, the primal objective (3) coincides with the dual objective y_{M+1}, the negative of γ_n, which proves the theorem. $\square$

There holds no strict complementarity, for $\operatorname{rank}(X) = \binom{n}{2} < n^2 - 1 = \operatorname{def}(S)$.

Example 2 *To define the primal matrix X take the four scalar products*

$$\langle \mathrm{row}_i, \mathrm{col}_j \rangle, \quad \langle \mathrm{col}_i, \mathrm{row}_j \rangle, \quad \langle \mathrm{row}_i, \mathrm{row}_j \rangle, \quad \langle \mathrm{col}_i, \mathrm{col}_j \rangle$$

in the index matrix IND where each of the four products determine n coordinates in $v_{i,j}$ as follows. If $n = 3$ and $i = 1$, $j = 2$, then $\mathrm{row}_1 = [1, 2, 3]$, $\mathrm{col}_2 = [2, 5, 8]^T$ *which yields by $(1, 2) \sim 1$, $(2, 5) \sim 8$, $(3, 8) \sim 24$ the coordinates* $1, 8, 24$, *see also matrix POS. Similarly we calculate the other three triples, giving together*

$$1,\ 8,\ 24;\ \ 4,\ 10,\ 21\ (!);\ \ 4,\ 8,\ 13;\ \ 1,\ 10,\ 28.$$

The repeated elements (1, 4, 8, 10) denote positions with value 2. The exclamation sign refers to an entry -1 (since $(7, 6)$ must be inverted to $(6, 7) \sim 21$). To sum up, we get

$$\begin{aligned} v_{1,2} = \quad & (2, 0,\ 0, 2, 0, 0, 0, 2, 0, 2, 0, 0, 1, 0, 0, 0, 0, 0, \\ & 0, 0, -1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0)^T. \end{aligned}$$

Example 3 *Hints for obtaining the dual matrix. We give some details for the $\binom{n}{2}$ most important blocks of order $6n-8$. There is a one-to-one correspondence between these blocks and ordered pairs (i, j), $1 \le i < j \le n$. To collect the indices for the block containing $z_{i,j}$, we have to consider the $4(n-1)$ terms in*

$$\langle \mathrm{row}_i, \mathrm{col}_j \rangle, \quad \langle \mathrm{col}_i, \mathrm{row}_j \rangle, \quad \langle \mathrm{row}_i, \mathrm{row}_j \rangle, \quad \langle \mathrm{col}_i, \mathrm{col}_j \rangle$$

(the same as for $v_{i,j}$ above!) and further $2(n-2)$ terms in the products

$$\mathrm{IND}(i, j) * \mathrm{diag}(\neq i, j), \quad \mathrm{IND}(j, i) * \mathrm{diag}(\neq i, j).$$

Here $\mathrm{diag}(\neq i, j)$ *stands for the $n-2$ entries of the diagonal of IND, differing from i, j. As in* Example 2, *choosing $n = 3$, $i = 1$, $j = 2$, vector diag($\neq 1, 2$) reduces to the $(3, 3)$ entry 9, thus we get (using index matrices IND and POS)*

$$((1, 2), (3, 3)) \sim (2, 9) \sim 30 \quad \text{and} \quad ((2, 1), (3, 3)) \sim (4, 9) \sim 32.$$

Hence the diagonal block containing row 1 (related to $z_{1,2}$) also contains rows 30 and 32. The whole index set at issue is $[1, 4, 8, 10, 13, 21, 24, 28, 30, 32]$, *and the corresponding block is the* 10×10 *(irreducible) matrix*

$$\begin{pmatrix} 3 & 0 & -2 & 0 & -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 3 & 0 & -2 & 0 & 1 & 0 & -1 & 0 & 0 \\ -2 & 0 & 3 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & -2 & 0 & 3 & -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 & 5 & 0 & 0 & -1 & -1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 3 & -1 & 0 & 0 & 0 \\ -1 & 0 & -1 & 0 & 0 & -1 & 3 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & -1 & 0 & 0 & 5 & 1 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 5 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 5 \end{pmatrix}$$

with eigenvalues $(0, 0, 3, 3, 4, 5, 5, 5, 5, 8)$.

3 Tridiagonal (and backward tridiagonal) matrices

In a former paper [4] we have shown that for nth order matrices P, Q with only nonzero entries in row 1 and column n the BW form is sos, however in case of (additional) main diagonal elements this is no more true. Therefore one would guess that $3n + O(1)$ nonzero elements cannot be allowed, however the result below shows that the answer depends on the position of these elements.

We shall use an index matrix given e.g. for $n = 3$ as IND $= \begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 5 \\ 0 & 6 & 7 \end{pmatrix}$.

Lemma 4 *For tridiagonal* P, Q *the BW form is sos, especially we have*

$$\begin{aligned} BW &= 2\sum_{i<j} z_{i,j}^2 \\ &- \sum (z_{3i-4,3i-3} - z_{3i-1,3i})^2 - \sum z_{3i-2,3i-1}^2 - \sum z_{3i,3i+3}^2 \\ &- \sum (z_{3i-2,3i-1} + z_{3i-1,3i+1})^2 - \sum (z_{3i-2,3i} + z_{3i,3i+1})^2 \\ &= \sum (z_{3i-4,3i-1} + z_{3i-3,3i})^2 + (z_{3i-4,3i} - z_{3i-3,3i-1})^2 \\ &+ \sum (z_{3i-2,3i-1} - z_{3i-1,3i+1})^2 + \sum (z_{3i-2,3i} - z_{3i,3i+1})^2 \\ &+ \sum z_{3i,3i+2}^2 + 2\sum z_{3i-2,3i+1}^2 + 2\sum_{i+5\le j} z_{i,j}^2 - \sum z_{3i-1,3i+3}^2. \end{aligned}$$

Remark 5 *The first equality gives the biquadratic form at issue, the second one is the claim: the sum of squares representation. (The negative terms in the last row are evidently canceled.)*

n	$\lambda=0$	$\lambda=1$	$\lambda=2$	$\lambda=3$	2-bl.	act.	rk(X)
2	3	0	3	0	2	0	3
3	7	1	12	1	6	1	7
4	11	2	30	2	10	2	11
5	15	3	57	3	14	3	15
6	19	4	93	4	18	4	19
7	23	5	138	5	22	5	23
8	27	6	192	6	26	6	27

Table 2: “Tridiagonal matrices”

Although SDP is not needed here, for the identity of the Lemma can be proved directly, we yet give some facts. The eigenvalues of the dual matrix S for the actual semidefinite programming problem are integers $(0, 1, 2, 3)$ in this case, too. This is so because matrix S decomposes into at most second order blocks of the form $\left(\begin{smallmatrix}1 & 1\\ 1 & 1\end{smallmatrix}\right)$ and $\left(\begin{smallmatrix}2 & -1\\ -1 & 2\end{smallmatrix}\right)$.

Table 2 illustrates the main features of the underlying semidefinite program. First the number of the eigenvalues of S are given, then the number of 2×2 blocks in S (the number of scalar blocks is not shown), the number of the active $(y_t \neq 0)$ constraints, and finally, the rank of X. (The $n-2$ active constraints correspond to the positions $(i-1, i), (i, i-1), (i, i+1)$ and $(i+1, i)$ in IND.)

It is easy to get a formula for these quantities, e.g. the number of eigenvalues $\lambda_i = 2$ can be determined by subtracting the number of all other eigenvalues from the order $(3n-2)(3n-3)/2$ of S. The result is $3 + 9\binom{n-1}{2}$.

Note that strict complementarity does hold: the number of zero eigenvalues of S coincides with rank(X), the number of nonzero eigenvalues of X.

Backward tridiagonal matrices

They have many similar properties, except that the case n odd is worse: while for n even all the eigenvalues of S are integers (lying in $[0, 4]$), for n odd this does not hold, therefore we write '–' instead. Also, in this case there are (apart from the scalar and 2×2 blocks) 4×4 blocks, too. All this information is contained in Table 3 from where one can see that for n even we again have strict complementarity, as in the tridiagonal case.

n	$\lambda=0$	$\lambda=1$	$\lambda=2$	$\lambda=3$	$\lambda=4$	act	2-bl.	4-bl.	rk(X)
2	3	0	3	0	0	2	0	0	3
3	8	-	-	-	-	3	3	7	5
4	13	4	25	0	3	6	3	8	13
5	20	-	-	-	-	11	6	17	18
6	25	6	81	2	6	14	6	18	25
7	32	-	-	-	-	30	9	27	30
8	37	8	173	4	9	22	9	28	37

Table 3: "Backward tridiagonal matrices"

Example 6 *We calculate the number act of active constraints:*

$$\text{act} = \begin{cases} 5n-12, & n \textit{ even} \\ 5n-8, & n \textit{ odd} \end{cases}$$

The number of terms in a typical row of the product of backward tridiagonal matrices usually equals $(0,\dots,0,1,2,3,2,1,0,\dots,0)$. *However, in case of the commutator* $PQ-QP$ *there are some minor changes: for odd order 1, for even order 2 main diagonal entries contain only two terms (instead of 3), due to the identity* $z_{i,j}+z_{j,i}=0$. *On the other hand, if* n *is even, there are two opposite entries (with indices* $(k,k+1)$ *and* $(k+1,k)$*, where* $k=n/2$*) which do not generate any constraint, for the corresponding indices are not distinct.*

Now we easily calculate the number asked, which is e.g. for $n=6$ *equal to* $5n-12=18$. *To this consider the matrix*

$$\begin{pmatrix} 2 & 2 & 1 & 0 & 0 & 0 \\ 2 & 3 & 2 & 1 & 0 & 0 \\ 1 & 2 & 2 & * & 1 & 0 \\ 0 & 1 & * & 2 & 2 & 1 \\ 0 & 0 & 1 & 2 & 3 & 2 \\ 0 & 0 & 0 & 1 & 2 & 2 \end{pmatrix}$$

with the number of terms in a given position of $[P,Q]$, *and take* $\binom{e}{2}$ *for any entry* $e>1$. *They sum up to* $2*\left(6\binom{2}{2}+\binom{3}{2}\right)=18$. *The general case is similar.*

4 Cyclic Hankel matrices

When investigating Hankel matrices, we find that – except for the case $n = 3$, see below – they do not generate sos BW forms. However, cyclic ones behave well. We make use of the small index matrix (given for $n = 3$): IND $= \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{pmatrix}$.

Theorem 7 *For cyclic Hankel matrices* P, Q *the BW form is a sum of squares.*

Proof. Using the above-defined index matrix with $(1, 2, \dots, n)$ as first row and $(n, 1, \dots, n-1)^T$ as last column, we obtain

$$\|P\|^2 = n\|p\|^2, \quad \|Q\|^2 = n\|q\|^2, \quad \mathrm{trace}(P^TQ) = np^Tq,$$

consequently

$$2\left(\|P\|^2\|Q\|^2 - \mathrm{trace}^2(P^TQ)\right) = 2n^2(\|p\|^2\|q\|^2 - (p^Tq)^2).$$

The commutator $[P, Q]$ is a skew symmetric cyclic Toeplitz matrix having

$$k = k_n = \left[\frac{n-1}{2}\right]$$

different entries $t_i = t_i^{(n)}$ with row one as

$$\begin{array}{ll} (0, t_1, \dots, t_k, -t_k, \dots, -t_1) & (n \text{ odd}) \\ (0, t_1, \dots, t_k, 0, -t_k, \dots, -t_1) & (n \text{ even}). \end{array}$$

Thus the subtrahend is $\|R\|^2 = 2n \sum t_i^2$, and the whole BW form equals

$$2n\left(n\sum_{i<j}^{n} z_{i,j}^2 - \sum_{1}^{k} t_i^2\right).$$

Observe now that all terms in

$$t_i = t_i^{(n)} = \sum_{j=1}^{n-i} z_{j,i+j} - \sum_{j=1}^{i} z_{j,n-i+j}$$

are distinct ($i = 1, \dots, k$), hence the Cauchy-Schwarz inequality in conjunction with $nk = n\left[\frac{n-1}{2}\right] \le \binom{n}{2}$ imply

$$\sum_{i=1}^{k} t_i^2 \le \sum_{i=1}^{k} n\Big(\sum_{j=1}^{n-i} z_{j,i+j}^2 + \sum_{j=1}^{i} z_{j,n-i+j}^2\Big) \le n\sum_{i<j} z_{i,j}^2,$$

which proves the theorem. The last inequality turns into equality for n odd.
$\square$

Remark 8 *The case* $n = 4$ *is especially interesting. Then the commutator is*

$$\begin{pmatrix} 0 & t & 0 & -t \\ -t & 0 & t & 0 \\ 0 & -t & 0 & t \\ t & 0 & -t & 0 \end{pmatrix}$$

with $t = t_1 = t_1^{(4)} = z_{1,2} + z_{2,3} + z_{3,4} - z_{1,4}$, *therefore the formula*

$$\begin{aligned} & 4\,(z_{1,2}^2 + z_{1,3}^2 + z_{2,3}^2 + z_{1,4}^2 + z_{2,4}^2 + z_{3,4}^2) = \\ + \quad & (z_{1,2} + z_{2,3} - z_{1,4} + z_{3,4})^2 + (z_{1,2} - z_{2,3} + z_{1,4} + z_{3,4})^2 \\ + \quad & (z_{1,2} + z_{1,3} - z_{2,4} - z_{3,4})^2 + (z_{1,2} - z_{1,3} + z_{2,4} - z_{3,4})^2 \\ + \quad & (z_{1,3} + z_{2,3} + z_{1,4} + z_{2,4})^2 + (z_{1,3} - z_{2,3} - z_{1,4} + z_{2,4})^2, \end{aligned}$$

(a consequence of Eulers identity) yields the sos-representation needed.

Example 9 *The case of (general) third order Hankel matrices. The index matrix IND is now* $\left(\begin{smallmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{smallmatrix}\right)$, *the order of* C, S *and the constraint matrices* $\{A_t\}$ *is* $\binom{5}{2} = 10$, *the number of the* A_t*-s is* $\binom{5}{4} = 5$. *By help of vector*

$$z = (z_{1,2},\ z_{1,3},\ z_{2,3},\ z_{1,4},\ z_{2,4},\ z_{3,4},\ z_{1,5},\ z_{2,5},\ z_{3,5},\ z_{4,5})^T$$

and matrix C *the objective can be written as* $BW = z^T C z = \operatorname{tr}(C z z^T)$, *which becomes – by means of a standard SDP relaxation – trace*(CX). *Our MATLAB program yields* $y = (0, 0, 1, 0, 0, 0)$, *i.e. only one constraint will be active, giving*

$$S = C - y_3 A_3 = \begin{pmatrix} 1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & \{1\} \\ 0 & 2 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 \\ -1 & 0 & 4 & 0 & 0 & -2 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & \{-1\} & 0 & 0 \\ 0 & -1 & 0 & 0 & 3 & 0 & \{1\} & 0 & -1 & 0 \\ -1 & 0 & -2 & 0 & 0 & 4 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & \{1\} & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \{-1\} & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 2 & 0 \\ \{1\} & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

(In the original C *the six entries in braces are zero.) The last zero in* y *indicates the sos representability. To obtain the concrete sos form, we calculated the*

eigen-decomposition of the three blocks

$$B_1 = \begin{pmatrix} 1 & -1 & -1 & 1 \\ -1 & 4 & -2 & -1 \\ -1 & -2 & 4 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}, B_2 = \begin{pmatrix} 2 & -1 & 0 & -1 \\ -1 & 3 & 1 & -1 \\ 0 & 1 & 1 & 0 \\ -1 & -1 & 0 & 2 \end{pmatrix}, B_3 = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$$

with integer eigenvalues

$$E_1 : \begin{pmatrix} 0 & 0 & 4 & 6 \end{pmatrix}, \qquad E_2 : \begin{pmatrix} 0 & 1 & 3 & 4 \end{pmatrix}, \qquad E_3 : \begin{pmatrix} 1 & 3 \end{pmatrix}$$

and (unnormalized, integer, columnwise) eigenvectors

$$V_1 : \begin{pmatrix} 2 & -1 & 1 & 0 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 0 & 3 & 1 & 0 \end{pmatrix}, V_2 : \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & -3 \\ -1 & 2 & 0 & -1 \\ 1 & 1 & -1 & 1 \end{pmatrix}, V_3 : \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

We sum up the result: with $z_{i,j} = p_i q_j - q_i p_j$, $1 \le i < j \le 5$ *the following identity holds for the BW form generated by two third order Hankel matrices:*

$$\begin{aligned}
&2z_{1,2}^2 + 3z_{1,3}^2 + 6z_{2,3}^2 + 2z_{1,4}^2 + 4z_{2,4}^2 + 6z_{3,4}^2 + z_{1,5}^2 + 2z_{2,5}^2 + 3z_{3,5}^2 + 2z_{4,5}^2 \\
&-(z_{1,3} + z_{2,4} + z_{3,5})^2 - (z_{1,2} + z_{2,3} + z_{3,4})^2 - (z_{2,3} + z_{3,4} + z_{4,5})^2 = \\
&\quad (z_{1,2} - z_{2,3} - z_{3,4} + z_{4,5})^2 + 3(z_{2,3} - z_{3,4})^2 \\
&+\frac{1}{6}(z_{1,3} + 2z_{1,5} + z_{3,5})^2 + \frac{3}{2}(z_{1,3} - z_{3,5})^2 + \frac{1}{3}(z_{1,3} - 3z_{2,4} - z_{1,5} + z_{3,5})^2 \\
&+\frac{1}{2}(z_{1,4} + z_{2,5})^2 + \frac{3}{2}(z_{1,4} - z_{2,5})^2.
\end{aligned}$$

5 Toeplitz matrices

In this section P and Q will be arbitrary real Toeplitz matrices. Observe that the main diagonal entries don't play any role (to prove this use temporarily the more standard notation $P = (p_{j-i})$ and $Q = (q_{j-i})$, then the (i, j) entry in $R = [P, Q]$ equals $\sum z_{k-i,j-k}$, while $z_{0,j-i} + z_{j-i,0} = 0$). Hence we can reduce the number of variables to get e.g. for $n = 3$ the index matrix $\mathrm{IND} = \begin{pmatrix} 0 & 1 & 2 \\ 3 & 0 & 1 \\ 4 & 3 & 0 \end{pmatrix}$.

Another speciality is that now there occur repeated terms as well. To handle these, introduce the multiplicity vector μ of dimension m by

$$\mu_i = \{\text{the number of occurrences of } p_i \text{ in } P,\ 1 \le i \le m\}.$$

Then the following easily proved representation holds.

Lemma 10

$$2\left(\|P\|^2\|Q\|^2 - \text{trace}^2(P^TQ)\right) = 2\sum_{i=1}^{m-1}\sum_{j=i+1}^{m} \mu_i\mu_j z_{i,j}^2,$$

and (since the commutator is skew persymmetric), $\|R\|^2 = 2\sum_{i=1}^{2}\sum_{j=1}^{n-i} r_{i,j}^2$.

In view of the lemma, we define the symmetric matrix C by help of equation $z^TCz = \frac{1}{2}\,BW(p,q)$. Then there are $m = 2(n-1)$ possible nonzero elements, the candidate vector z has dimension $N = \binom{m}{2}$, and the total number of constraints (A_t) is $M = \binom{m}{4}$. For this problem we formulate a 'quasi-optimal' strategy of choosing the dual variables.

Strategy B. Since the entries of $[P,Q]$ are linear forms in $(z_{i,j})$, their squares figuring in $\|R\|^2$ involve some mixed products of the form $\pm 2\,z_{i,j}z_{k,l}$. Whenever finding such a term with distinct $\{i,j,k,l\}$, we increase the actual value of y.

It turns out that Strategy B works for n, $3 \le n \le 7$, however, for $n \ge 8$ the dual matrix $S = C - \sum y_tA_t$ will have (more and more) negative eigenvalues.

Lemma 11 *For orders* n *not exceeding 7, the matrix* S *is p.s.d, i.e. for these values the BW form is sos. Some further properties of* S *of arbitrary order* n *are: the minimum off-diagonal entry of* S *is* $-\lfloor\frac{n-1}{2}\rfloor$; *the defect of* S, *i.e. the multiplicity of zero as eigenvalue is* $n-1$; *the maximal diagonal entry and also the maximal eigenvalue is* $n(n-2)$. *Moreover,* S *is a direct sum of two types of submatrices of the following order:*
– type (a): $2,4,6,\dots,2(n-2)$; *(denote by* B *the largest block here)*
– type (b): $1,1,2,2,\dots,n-2,n-2,n-1$.
The orders of these matrices sum up to $(n-1)(2n-3)$, *the order of* S.

The largest block B of type (a) is crucial. It has a decomposition $B = \begin{pmatrix} D & H \\ H & D\end{pmatrix}$, with D diagonal, H Hermitian (i.e. symmetric), both of order $n-2$. The diagonal elements of D are $(i(i+1))$ in reverse order: $((n-2)(n-1),\dots,6,2)$. Matrix H is also of a special structure: the elements on the border are -1, those on the 'neighboring' border are -2, and so on. This matrix is p.s.d. for $n \le 7$, but has at least one negative eigenvalue for $n \ge 8$.

Remark 12 *The further submatrices of type (a) also are critical, e.g. the next one (of order* $2(n-3)$*) has a similar form with diagonal elements* $(i(i+2))$ *in* D, *while* H *is the same (of the appropriate size). Therefore there is a second negative eigenvalue for* n, $14 \le n \le 20$, *and so on. In general, the symmetric matrices* H *are of the same form, and the diagonal entries of* D *are* $(i(i+k))_i$.

Example 13 *Matrices of order* 5. *In this case* P *and* Q *have* $m = 2(n-1) = 8$ *nonzero elements, the candidate vector* z *has dimension* $\binom{m}{2} = 28$, *the total number of constraints is* $\binom{m}{4} = 70$, *and the number of active constraints is* 14.

It always suffices to examine the first row and the first column of R, *for all other entries are contained in these, e.g.* $R(1,1) = z_{1,5} + z_{2,6} + z_{3,7} + z_{4,8}$, *and* $R(2,2) = z_{2,6} + z_{3,7}$. *The number of the active constraints for* $n = 5$, *coming from row 1 and column 1 is indeed 6 + 2 (3 + 1) = 14, as stated above. This can be proved by induction, by noting that*

$$\binom{n-1}{2} + 2\left\{\binom{n-2}{2} + \binom{n-3}{2} + \cdots + \binom{2}{2}\right\} = \frac{1}{6}(n-1)(n-2)(2n-3).$$

As regards the y *coordinates, since the product* $2z_{2,6}z_{3,7}$ *occurs two times (as the above formulae show), we write* -2 *in the suitable positions (overwriting the* -1*-s) to get* $S(13,17) = S(3,21) = -2$, *and so on.*

Now we give another example illustrating the role of the crucial block B.

Example 14 *The case* $n = 8$. *The matrices* D *and* H *are now:*

$$D = \begin{pmatrix} 42 & 0 & 0 & 0 & 0 & 0 \\ 0 & 30 & 0 & 0 & 0 & 0 \\ 0 & 0 & 20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 12 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}, \quad H = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -2 & -2 & -2 & -2 & -1 \\ -1 & -2 & -3 & -3 & -2 & -1 \\ -1 & -2 & -3 & -3 & -2 & -1 \\ -1 & -2 & -2 & -2 & -2 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{pmatrix}.$$

The characteristic polynomial of the block $B = \begin{pmatrix} D & H \\ H & D \end{pmatrix}$ *factorizes into* p_1p_2, *where* $p_1(x) = x^6 - 100x^5 + 536x^4 - 53472x^3 + 327472x^2 - 575680x - 145152$, *and* p_1 *has a negative zero: – 0.2228. (All other zeroes of* p_1 *and* p_2 *are positive.)*

Finally we mention that although the above strategy works only up to $n = 7$, the standard semidefinite program yields results indicating that BW can be sos for some larger orders, too, hence we guess that BW is sos in general. The difficulty is that the corresponding dual variables (y_t) of the program are not recognizable real numbers. Nevertheless we formulate the following.

Conjecture 15 *The Böttcher-Wenzel form (1) generated by two real Toeplitz matrices is sos, i.e. a sum of squares of polynomials, now: quadratic forms. Give – if possible – a rational certification, i.e. rational parameters* (y_t) *such that* $S = C - \sum y_tA_t$ *is positive semidefinite.*

References

[1] A. Böttcher, D. Wenzel, How big can the commutator of two matrices be and how big is it typically?, *Linear Algebra Appl.* **403** (2005) 216–228. ⇒17, 18

[2] A. Böttcher, D. Wenzel, The Frobenius norm and the commutator, *Linear Algebra Appl.* **429** (2008) 1864–1885. ⇒17

[3] K. M. R. Audenaert, Variance bounds, with an application to norm bounds for commutators, *Linear Algebra Appl.* **432** (2010) 1126–1143. ⇒17

[4] L. László, On sum of squares decomposition for a biquadratic matrix function, *Ann. Univ. Sci. Budapest. Sect. Comput.* **33** (2010) 273–284. ⇒24

[5] L. László, On a structured semidefinite program, *Acta Univ. Sapientiae, Math.*, **3**, 1 (2011) 77–89. ⇒18

[6] Z. Lu, Normal scalar curvature conjecture and its applications, *Funct. Anal.* **261** (2011) 1284–1308. ⇒17

[7] S-W. Vong, X-Q. Jin, Proof of Böttcher and Wenzel's conjecture, *Oper. Matrices*, **2**, 3 (2008) 435–442. ⇒17

[8] M. J. Todd, Semidefinite optimization, *Acta Numerica* **10** (2001) 515–560, also in: http://people.orie.cornell.edu/ miketodd/soa5.pdf ⇒19

Received: February 8, 2012 • Revised: March 22, 2012

Counting (k, l)-sumsets in groups of prime order

Vahe SARGSYAN
Moscow State University
email: vahe_sargsyan@ymail.com

Abstract. A subset A of a group $\mathbf{G}$ is called (k, l)-*sumset*, if $A = kB - lB$ for some $B \subseteq \mathbf{G}$, where $kB - lB = \{x_1 + \cdots + x_k - x_{k+1} - \cdots - x_{k+l} : x_1, \ldots, x_{k+l} \in B\}$. Upper and lower bounds for the number (k, l)-sumsets in groups of prime order are provided.

1 Introduction

Let p be a prime number and k, l be nonnegative integers with $k+l \geq 2$. Write $\mathbf{Z}_p$ for the group of residues modulo p. A subset $A \subseteq \mathbf{Z}_p$ is called (k, l)-*sumset*, if $A = kB - lB$ for some $B \subseteq \mathbf{Z}_p$, where $kB - lB = \{x_1 + \cdots + x_k - x_{k+1} - \cdots - x_{k+l} : x_1, \ldots, x_{k+l} \in B\}$. Write $\mathbf{SS}_{k,l}(\mathbf{Z}_p)$ for the collection of (k, l)-sumsets in $\mathbf{Z}_p$.

B. Green and I. Ruzsa in [1] proved

$$p^2 2^{p/3} \ll |\mathbf{SS}_{2,0}(\mathbf{Z}_p)| \leq 2^{p/3+\theta(p)}$$

where $\theta(p)/p \to 0$ as $p \to \infty$ and $\theta(p) \ll p(\log\log p)^{2/3}(\log p)^{-1/9}$ (hereafter logarithms are to base two).

The aim of this work is to obtain bounds for the number $|\mathbf{SS}_{k,l}(\mathbf{Z}_p)|$. We prove

Theorem 1 *Let p be a prime number and k,l be nonnegative integers with $k + l \geq 2$. Then there exists a positive constant $C_{k,l}$ such that*

$$C_{k,l} 2^{p/(2(k+l)-1)} \leq |SS_{k,l}(Z_p)| \leq 2^{(p/(k+l+1))+(k+l-2)+o(p)}. \tag{1}$$

Computing Classification System 1998: G.2.1
Mathematics Subject Classification 2010: 68R05
Key words and phrases: sumset, characteristic function, partition, glanular set

2 Definitions and auxiliary results

Let $\mathbf{R}$ be the set of real numbers, $f_i : \mathbf{Z}_p \to \mathbf{R}$, $i = 1, \dots, m$, and $x \in \mathbf{Z}_p$. We set

$$(f_1 * \dots * f_m)(x) =$$
$$= \sum_{x_1 \in \mathbf{Z}_p} \cdots \sum_{x_{m-1} \in \mathbf{Z}_p} f_1(x_1) \dots f_{m-1}(x_{m-1}) f_m(x - x_1 - \dots - x_{m-1}) \tag{2}$$

and

$$\widehat{f}(x) = \sum_{y \in \mathbf{Z}_p} f(y) e^{2\pi i \frac{xy}{p}}.$$

The function $\widehat{f}(x)$ is called *Fourier transform* of f.

Lemma 2 *We have*

$$(f_1 \widehat{* \dots *} f_m)(x) = \widehat{f_1}(x) \dots \widehat{f_m}(x). \tag{3}$$

Proof. By definition

$$(f_1 \widehat{* \dots *} f_m)(x) = \sum_{y \in \mathbf{Z}_p} (f_1 * \dots * f_m)(y) e^{2\pi i \frac{yx}{p}} =$$
$$= \sum_{y \in \mathbf{Z}_p} \sum_{y_1 \in \mathbf{Z}_p} \cdots \sum_{y_{m-1} \in \mathbf{Z}_p} f_1(y_1) \dots f_{m-1}(y_{m-1}) \times$$
$$\times f_m(y - y_1 - \dots - y_{m-1}) \cdot e^{2\pi i \frac{y_1 x}{p}} \dots e^{2\pi i \frac{y_{m-1} x}{p}} \cdot e^{2\pi i \frac{(y - y_1 - \dots - y_{m-1})x}{p}} =$$
$$= \sum_{y_1 \in \mathbf{Z}_p} f_1(y_1) \cdot e^{2\pi i \frac{y_1 x}{p}} \cdots \sum_{y_{m-1} \in \mathbf{Z}_p} f_{m-1}(y_{m-1}) \cdot e^{2\pi i \frac{y_{m-1} x}{p}} \times$$
$$\times \sum_{y \in \mathbf{Z}_p} f_m(y - y_1 - \dots - y_{m-1}) \cdot e^{2\pi i \frac{(y - y_1 - \dots - y_{m-1})x}{p}} = \widehat{f_1}(x) \dots \widehat{f_m}(x).$$

$\square$

Denote the characteristic function of a set A by $\chi_A(x)$. Let $A_1, \dots, A_m$ be non-empty subsets of $\mathbf{Z}_p$. Then $(\chi_{A_1} * \dots * \chi_{A_m})(x)$ will be the number of vectors $(x_1, \dots, x_m) \in A_1 \times \dots \times A_m$ such that $x \equiv x_1 + \dots + x_m \pmod{p}$. Set $A_1 + \dots + A_m = \{x_1 + \dots + x_m \pmod{p} : x_1 \in A_1, \dots, x_m \in A_m\}$. We define $S_{h,m}(A_1, \dots, A_m) = \{x \in \mathbf{Z}_p : (\chi_{A_1} * \dots * \chi_{A_m})(x) \geq h\}$, where $h > 0$. Further, for any integer i and any $A \subseteq \mathbf{Z}_p$ denote the set $\underbrace{A + \dots + A}_{i}$ by iA, and the set $\{p - x : x \in A\}$ by $-A$.

Theorem 3 (Cauchy-Davenport, [2]). *Let $A_1,\dots,A_m$ be non-empty subsets of $\boldsymbol{Z}_p$. Then $|A_1+\cdots+A_m| \geq \min(p, |A_1|+\cdots+|A_m|-(m-1))$.*

Theorem 4 (Pollard, [3]). *Let A_1, A_2 be non-empty subsets of $\boldsymbol{Z}_p$. Then*

$$|S_{1,2}(A_1,A_2)|+\cdots+|S_{t,2}(A_1,A_2)| \geq t\min(p, |A_1|+|A_2|-t),$$

where $t \leq \min(|A_1|, |A_2|)$.

Theorems 3, 4 imply the following two statements.

Lemma 5 *Let $A_1,\dots,A_m$ non-empty subsets of $\boldsymbol{Z}_p$. Then*

$$|S_{1,m}(A_1,\dots,A_m)|+\cdots+|S_{t,m}(A_1,\dots,A_m)| \geq$$

$$\geq t\min(p, |A_1|+\cdots+|A_m|-t-m+2),$$

where $t \leq \min(|A_1|,\dots,|A_m|)$.

Proof. Without loss of generality we assume $|A_1| = \min(|A_1|,\dots,|A_m|)$. By Theorem 4 we have

$$|S_{1,2}(A_1,(A_2+\cdots+A_m))|+\cdots+|S_{t,2}(A_1,(A_2+\cdots+A_m))| \geq$$

$$\geq t\min(p, |A_1|+|A_2+\cdots+A_m|-t), \tag{4}$$

where $t \leq |A_1|$.
On the other hand by Theorem 3 we have

$$|A_2+\cdots+A_m| \geq \min(p, |A_2|+\cdots+|A_m|-(m-2)). \tag{5}$$

Substituting (5) in (4), we obtain

$$|S_{1,m}(A_1,\dots,A_m)|+\cdots+|S_{t,m}(A_1,\dots,A_m)| \geq$$

$$\geq |S_{1,2}(A_1,(A_2+\cdots+A_m))|+\cdots+|S_{t,2}(A_1,(A_2+...+A_m))| \geq$$

$$\geq t\min(p, |A_1|+\cdots+|A_m|-t-m+2).$$

□

Lemma 6 *Let $A_1,\dots,A_m$ be non-empty subsets of $\boldsymbol{Z}_p$ and $h \leq \min(|A_1|,\dots,|A_m|)$. Then*

$$|S_{h,m}(A_1,\dots,A_m)| \geq \min(p, |A_1|+\cdots+|A_m|-m+2)-2(hp)^{1/2}.$$

Proof. Note that $|S_{i,m}(A_1,\dots,A_m)| \geq |S_{j,m}(A_1,\dots,A_m)|$ for $i \leq j$. Choose $h \leq t \leq \min(|A_1|,\dots,|A_m|)$. By Lemma 5 we have

$$t\min(p,|A_1|+\cdots+|A_m|-t-m+2) \leq$$
$$\leq |S_{1,m}(A_1,\dots,A_m)|+\cdots+|S_{t,m}(A_1,\dots,A_m)| \leq$$
$$\leq hp + t|S_{h,m}(A_1,\dots,A_m)|.$$

Putting $t=(hp)^{1/2}$, we get

$$\min(p,|A_1|+\cdots+|A_m|-m+2)-2(hp)^{1/2} \leq$$
$$\leq \min(p,|A_1|+\cdots+|A_m|-m-(hp)^{1/2}+2)-(hp)^{1/2} \leq$$
$$\leq |S_{h,m}(A_1,\dots,A_m)|.$$

□

Lemma 7 *Set* $\boldsymbol{T}_{r,s}(\boldsymbol{Z}_p)=\{A \subset \boldsymbol{Z}_p : |A| \leq p/(r+1)s\}$. *Then there exists* s *such that*

$$|\boldsymbol{T}_{r,s}(\boldsymbol{Z}_p)| \leq 2^{p/(r+1)}. \tag{6}$$

Proof. Let n, m be positive integers, $1 \leq m \leq n$. Then (see Lemma 6.8, [4])

$$\sum_{0\leq i\leq m}\binom{n}{i} \leq \left(\frac{en}{m}\right)^m. \tag{7}$$

We choose s such that

$$es(r+1) \leq 2^s. \tag{8}$$

Then by (7) we have (putting $n=p$ and $m=p/(r+1)s$)

$$|\mathbf{T}_{r,s}(\mathbf{Z}_p)| = \sum_{0\leq i\leq p/(r+1)s}\binom{p}{i} \leq (es(r+1))^{p/(r+1)s} \leq (2^s)^{p/(r+1)s} = 2^{p/(r+1)}.$$

□

Let L be a positive integer. For each $y \in \{0,\dots,p-1\}$ we define a partition $\mathbf{R}_{y,L}$ of $\mathbf{Z}_p$ on the intervals of the form $J_i^y = \{(iL+1+y) \pmod p),\dots,((i+1)L+y) \pmod p)\}$, $0 \leq i \leq \lfloor p/L \rfloor - 1$. All intervals are J_i^y of $\mathbf{R}_{y,L}$ have length L, and the set $J_y = \mathbf{Z}_p \setminus \bigcup_i J_i^y$ has cardinality $p - L\lfloor p/L\rfloor < L$. The set J_y is called *remainder* partition $\mathbf{R}_{y,L}$. In what follows we fix $y \in \{0,\dots,p-1\}$ and consider the corresponding partition $\mathbf{R}_{y,L}$. For every $A \subseteq \mathbf{Z}_p$ and any integer d define $d \star A = \{da \pmod p) : a \in A\}$. The set $d \star A$ is called *dilation* of A. The set $A \subseteq \mathbf{Z}_p$ is called L-*granular* (see [1]), if some dilation of A is a union of some of the intervals J_i^y (other than remainder). We denote the family of L-granular subsets of $\mathbf{Z}_p$ by $\mathbf{G}_L(\mathbf{Z}_p)$.

Lemma 8 *We have*

$$|\boldsymbol{G}_L(\boldsymbol{Z}_p)| \leq p2^{p/L}. \tag{9}$$

Proof. Denote the number of subsets of intervals (other than remainder) of the partition $R_{y,L}$ of $\mathbf{Z}_p$ by $g(R_{y,L})$, and the number of different partitions $R_{y,L}$ of $\mathbf{Z}_p$ by $r(L)$. It is obvious that

$$|\mathbf{G}_L(\mathbf{Z}_p)| \leq g(R_{y,L})r(L). \tag{10}$$

Note that the number of intervals (other than remainder) of the partition $\mathbf{R}_{y,L}$ of $\mathbf{Z}_p$ is equal to $\lfloor p/L \rfloor$, and the number of different partitions $\mathbf{R}_{y,L}$ of $\mathbf{Z}_p$ is at most p. This and (10) imply the inequality (9). $\square$

Lemma 9 *Let $A \subseteq \boldsymbol{Z}_p$ have size αp, and let $\varepsilon_1, \varepsilon_2, \varepsilon_3$ be positive real numbers and $L > 0$, k, l be nonnegative integers satisfying $k + l \geq 2$. Suppose that*

$$p > (\sqrt{8(k+l)}L)^{4^{2(k+l)}\alpha^{2(k+l-1)}\varepsilon_1^{-2(k+l)}\varepsilon_2^{-2(k+l-1)}\varepsilon_3^{-1}}. \tag{11}$$

Then there exists a set $A' \subseteq \boldsymbol{Z}_p$ with the following properties:
(i) A' *is* L*-granular;*
(ii) $|A \setminus A'| \leq \varepsilon_1 p$;
(iii) *the set* $kA - lA$ *contains all* $x \in \boldsymbol{Z}_p$ *for which* $(\underbrace{\chi_{A'} * \cdots * \chi_{A'}}_{k} * \underbrace{\chi_{-A'} * \cdots * \chi_{-A'}}_{l})(x) \geq (\varepsilon_2 p)^{k+l-1}$, *with at most* $\varepsilon_3 p$ *exceptions.*

Proof. Let $h \in \{0, \ldots, p-1\}$, and $\mathbf{R}_{h,L}$ be partition of $\mathbf{Z}_p$.

(i) For given set $A \subset \mathbf{Z}_p$ we define $A' \subset \mathbf{Z}_p$ as the union of intervals J_i^h of the partition $\mathbf{R}_{h,L}$, such that $|A \cap J_i^h| \geq \varepsilon_1 L/2$. From the definition it follows that A' is L-granular. It is easy to see that $(-A)' = -(A')$.

(ii) Let $x \in A \setminus A'$. Then either $x \in J_h$ or $x \in A \cap J_i^h$, $(i = 0, \ldots, \lfloor p/L \rfloor - 1)$, and $|A \cap J_i^h| \leq \varepsilon_1 L/2$. In the first case we have $|J_h| < L$, and inequality (11) implies $L \leq \varepsilon_1 p/2$. Thus,

$$|A \setminus A'| \leq \frac{\varepsilon_1 L}{2} \cdot \frac{p}{L} + L \leq \varepsilon_1 p.$$

(iii) Let $\widehat{\chi_A}(x)$ be the Fourier transform of the characteristic function χ_A of A, so that

$$\widehat{\chi_A}(x) = \sum_{y \in \mathbf{Z}_p} \chi_A(y) e^{2\pi i \frac{yx}{p}} = \sum_{y \in A} e^{2\pi i \frac{yx}{p}}$$

for all $x \in \mathbf{Z}_p$. Take $\delta = 4^{-(k+l)}\varepsilon_1^{k+l}\varepsilon_2^{k+l-1}\varepsilon_3^{1/2}\alpha^{-(k+l)+3/2}$, where $\varepsilon_1, \varepsilon_2$ and ε_3 are from inequality (11). Set $\mathbf{D} = \{x \neq 0 : |\widehat{\chi_A}(x)| \geq \delta p\}$. We define the function $f(x)$ as follows:

$$f(x) = \frac{1}{2L-1}\sum_{j=-(L-1)}^{L-1} e^{2\pi i \frac{jqx}{p}}.$$

In the future we will show that there exists $q \in \mathbf{Z}_p \setminus \{0\}$ such that for all $x \in \mathbf{Z}_p$ it holds

$$|\widehat{\chi_A}(x)||1 - f^{k+l}(x)| \leq \delta p. \tag{12}$$

The inequality (12) obviously holds for the case $x = 0$, since $f(0) = 1$, as well as for the case $|\widehat{\chi_A}(x)| \leq \delta p$, since $f(x) \in [-1, 1]$. Thus, it remains to show the existence of q such that the inequality (12) holds for all $x \in \mathbf{D}$. First we estimate the value of $1 - f(x)$. Denote by $\langle x \rangle$ the distance from x to the nearest integer. We use the fact that $1 - \cos(2\pi x) \leq 2\pi^2 \langle x \rangle^2$. Then

$$1 - f(x) = \frac{2}{2L-1}\sum_{j=1}^{L-1}\left(1 - \cos\frac{2\pi jqx}{p}\right) \leq \frac{4\pi^2}{2L-1}\sum_{j=1}^{L-1}\left\langle \frac{jqx}{p}\right\rangle^2 \leq$$

$$\leq \frac{4\pi^2}{2L-1}\left\langle \frac{qx}{p}\right\rangle^2 \sum_{j=1}^{L-1} j^2 \leq \frac{2\pi^2 L^2}{3}\left\langle \frac{qx}{p}\right\rangle^2. \tag{13}$$

Recall that for $|x| \leq 1$

$$1 - x^m = (1-x)(1 + x + x^2 + \cdots + x^{m-1}) \leq m(1-x). \tag{14}$$

From (13) and (14) it follows

$$|\widehat{\chi_A}(x)||1 - f^{k+l}(x)| \leq (k+l)|\widehat{\chi_A}(x)||1 - f(x)| \leq 8(k+l)L^2\langle qx/p\rangle^2|\widehat{\chi_A}(x)|.$$

Note that if the inequality

$$\left\langle \frac{qx}{p}\right\rangle \leq \frac{1}{\sqrt{8(k+l)}L}\left(\frac{\delta p}{|\widehat{\chi_A}(x)|}\right)^{1/2} \tag{15}$$

holds for some $q \in \mathbf{Z}_p \setminus \{0\}$ and for all $x \in \mathbf{D}$ then the inequality (12) also holds. Now we will prove that such q exists. By definition, we have

$$\langle qx/p\rangle = \min\{(qx \pmod p)/p, (p - qx \pmod p)/p\}.$$

Set $|\mathbf{D}| = d$, $\mathbf{D} = \{r_1, \dots, r_d\}$. We denote $a_i = (1/\sqrt{8(k+l)}L)(\delta p/|\widehat{\chi_A}(r_i)|)^{1/2}$. Then the inequality (15) can be rewritten as

$$\min\{qr_i \pmod p, p - qr_i \pmod p\} \le pa_i, \quad \text{where} \quad i = 1, \dots, d. \tag{16}$$

Denote the set $\{(x_1, \dots, x_d) : x_1, \dots, x_d \in \mathbf{Z}_p\}$ by $\mathbf{Z}_p^d$. We split $\mathbf{Z}_p^d$ on disjoint subsets

$$\mathbf{Z}_p^d = \bigcup_{(i_1,\dots,i_d)} \mathbf{Q}_{i_1,\dots,i_d},$$

where

$$\mathbf{Q}_{i_1,\dots,i_d} = \{(x_1, \dots, x_d) : i_j pa_j < x_j \le (i_j + 1)pa_j, j = 1, \dots, d\}.$$

Let μ_d be number of different sets of $\mathbf{Q}_{i_1,\dots,i_d}$. Using the fact that $0 \le i_j \le 1/a_j - 1$, $j = 1, \dots, d$, we have

$$\mu_d \le \prod_{i=1}^{d} \frac{1}{a_i}.$$

Let us consider the following $p-1$ elements of $\mathbf{Z}_p^d$:

$$(qr_1 \pmod p), \dots, qr_d \pmod p)), \quad \text{where } r_1, \dots, r_d \in \mathbf{D}, \quad q = 1, \dots, p-1.$$

We show that if

$$p > \prod_{i=1}^{d} \frac{1}{a_i}, \tag{17}$$

then there exists q such that for all $r_i \in \mathbf{D}$, $i = 1, \dots, d$, the inequality (16) holds. We consider two cases:
(A) If $\mu_d = p-1$, then we take $q = q_0$, where $q_0 \in \mathbf{Z}_p \setminus \{0\}$ such that $(q_0 r_1 \pmod p), \dots, q_0 r_d \pmod p)) \in \mathbf{Q}_{0,\dots,0}$.
(B) If $\mu_d < p-1$, then by pigeonhole principle, there are q_1, $q_2 \in \mathbf{Z}_p \setminus \{0\}$ such that the vectors $(q_1 r_1 \pmod p), \dots, q_1 r_d \pmod p))$ and $(q_2 r_1 \pmod p), \dots, q_2 r_d \pmod p))$ belong to the same set of $\mathbf{Q}_{i_1,\dots,i_d}$. Obviously, when $q = q_1 - q_2$ the inequality (16) holds.

We now show that inequality (17) is a consequence of (11). Indeed, by the Parseval's identity, we have

$$p^{-1}\left(\sum_{x\in\mathbf{D}} |\widehat{\chi_A}(x)|^2 + \sum_{x\in\mathbf{Z}_p\setminus\mathbf{D}} |\widehat{\chi_A}(x)|^2\right) = \sum_{x\in\mathbf{Z}_p} |\chi_A(x)|^2 = \alpha p. \tag{18}$$

From (18) it follows
$$\sum_{x\in \mathbf{D}} |\widehat{\chi_A}(x)|^2 \le \alpha p^2. \tag{19}$$
From (19) and the arithmetic and geometric mean inequality, we get
$$\left(\prod_{x\in \mathbf{D}} |\widehat{\chi_A}(x)|^2\right)^{1/d} \le \frac{1}{d}\sum_{x\in \mathbf{D}} |\widehat{\chi_A}(x)|^2 \le \frac{\alpha p^2}{d}.$$
i.e.
$$\prod_{x\in \mathbf{D}} |\widehat{\chi_A}(x)| \le \left(\frac{\alpha p^2}{d}\right)^{d/2}. \tag{20}$$
From (20) we get
$$(\sqrt{8(k+l)}L)^d \left(\prod_{x\in \mathbf{D}} \frac{|\widehat{\chi_A}(x)|}{\delta p}\right)^{1/2} \le (\sqrt{8(k+l)}L\alpha^{1/4}\delta^{-1/2}d^{-1/4})^d. \tag{21}$$
It is easy to see that the right-hand side of (21) is an increasing function of d in the range $d < 64(k+l)^2L^4\alpha/\delta^2 e$.
On the other hand, from (19) we have $d\delta^2p^2 \le \alpha p^2$. Hence, $d \le \alpha/\delta^2$. Consequently
$$(\sqrt{8(k+l)}L\alpha^{1/4}\delta^{-1/2}d^{-1/4})^d \le (\sqrt{8(k+l)}L)^{\alpha/\delta^2}.$$
Recall that $\delta = 4^{-(k+l)}\varepsilon_1^{k+l}\varepsilon_2^{k+l-1}\varepsilon_3^{1/2}\alpha^{-(k+l)+3/2}$. From this it follows that there exists q such that the inequality (12) holds. Moreover, without loss of generality we can assume $q = 1$ (this can be achieved by selecting an appropriate dilation of the set A).

Define two functions $\chi_1(x)$ and $\chi_2(x)$ as follows:
$$\chi_1(x) = \frac{1}{|\mathcal{J}|}(\chi_A * \chi_{\mathcal{J}})(x),$$
$$\chi_2(x) = \frac{1}{|\mathcal{J}|}(\chi_{-A} * \chi_{\mathcal{J}})(x),$$
where $\mathcal{J} = \{-(L-1),\ldots,L-1\}$. From (2) it follows that
$$\chi_1(x) = \frac{1}{|\mathcal{J}|}|A \cap (\mathcal{J} + x)|, \tag{22}$$

$$\chi_2(x) = \frac{1}{|\mathcal{J}|}|(-A) \cap (\mathcal{J} + x)|, \tag{23}$$

and from (3) we have $\widehat{\chi_1}(x) = \widehat{\chi_A}(x)f(x)$ and $\widehat{\chi_2}(x) = \widehat{\chi_{-A}}(x)f(x)$. Hence, by Parseval's identity and from (3) we get

$$\sum_{x\in\mathbf{Z}_p}\left|(\underbrace{\chi_A*\cdots*\chi_A}_{k}*\underbrace{\chi_{-A}*\cdots*\chi_{-A}}_{l})(x)-(\underbrace{\chi_1*\cdots*\chi_1}_{k}*\underbrace{\chi_2*\cdots*\chi_2}_{l})(x)\right|^2=$$

$$=p^{-1}\sum_{x\in\mathbf{Z}_p}\left|(\underbrace{\chi_A*\cdots*\chi_A}_{k}\widehat{*\chi_{-A}}\underbrace{*\cdots*\chi_{-A}}_{l})(x)-(\underbrace{\chi_1*\cdots*\chi_1}_{k}\widehat{*\chi_2}\underbrace{*\cdots*\chi_2}_{l})(x)\right|^2$$

$$=p^{-1}\sum_{x\in\mathbf{Z}_p}\left|\widehat{\chi_A}^k(x)\widehat{\chi_{-A}}^l(x)-\widehat{\chi_1}^k(x)\widehat{\chi_2}^l(x)\right|^2=$$

$$=p^{-1}\sum_{x\in\mathbf{Z}_p}|\widehat{\chi_A}(x)|^{2k}|\widehat{\chi_{-A}}(x)|^{2l}\left|1-f^{k+l}(x)\right|^2\leq$$

$$\leq p^{-1}\left(\sup_{x\in\mathbf{Z}_p}|\widehat{\chi_A}(x)|^{k-1}|\widehat{\chi_{-A}}(x)|^{l}\left|1-f^{k+l}(x)\right|\right)^2\sum_{x\in\mathbf{Z}_p}|\widehat{\chi_A}(x)|^2. \tag{24}$$

We have

$$|\widehat{\chi_A}(x)|=\left|\sum_{y\in\mathbf{Z}_p}\chi_A(y)e^{2\pi i\frac{yx}{p}}\right|=\left|\sum_{y\in A}e^{2\pi i\frac{yx}{p}}\right|\leq\sum_{y\in A}\left|e^{2\pi i\frac{yx}{p}}\right|=\alpha p, \tag{25}$$

$$|\widehat{\chi_{-A}}(x)|=\left|\sum_{y\in\mathbf{Z}_p}\chi_{-A}(y)e^{2\pi i\frac{yx}{p}}\right|=\left|\sum_{y\in -A}e^{2\pi i\frac{yx}{p}}\right|\leq\sum_{y\in -A}\left|e^{2\pi i\frac{yx}{p}}\right|=\alpha p. \tag{26}$$

From (12), (18), (24), (25) and (26) it follows

$$\sum_{x\in\mathbf{Z}_p}\left|(\underbrace{\chi_A*\cdots*\chi_A}_{k}*\underbrace{\chi_{-A}*\cdots*\chi_{-A}}_{l})(x)-(\underbrace{\chi_1*\cdots*\chi_1}_{k}*\underbrace{\chi_2*\cdots*\chi_2}_{l})(x)\right|^2\leq$$

$$\leq\left(\sup_{x\in\mathbf{Z}_p}|\widehat{\chi_A}(x)|\left|1-f^{k+l}(x)\right|\right)^2\alpha^{2(k+l)-3}p^{2(k+l)-3}\leq$$

$$\leq \alpha^{2(k+l)-3}\delta^2 p^{2(k+l)-1}. \tag{27}$$

Suppose that $x \in A'$ ($x \in -A'$). Then there exists an interval $\mathcal{I}$ of length L such that $\mathcal{I} \subseteq \{x-(L-1), \dots, x+(L-1)\}$ and $x \in \mathcal{I}$. From definition of A' ($-A'$) it follows that $|\mathcal{I} \cap A| \geq \varepsilon_1 L/2$ ($|\mathcal{I} \cap (-A)| \geq \varepsilon_1 L/2$). From the definition of $\chi_1(x)$ ($\chi_2(x)$) it follows that $\chi_1(x) \geq \varepsilon_1/4$ ($\chi_2(x) \geq \varepsilon_1/4$). Observe, that $\chi_1(x) \geq \varepsilon_1 \chi_{A'}(x)/4$ and $\chi_2(x) \geq \varepsilon_1 \chi_{-A'}(x)/4$ hold for all $x \in \mathbf{Z}_p$. From this and (2) it follows that

$$(\underbrace{\chi_1 * \cdots * \chi_1}_{k} * \underbrace{\chi_2 * \cdots * \chi_2}_{l})(x) \geq$$

$$\geq \varepsilon_1^{k+l}(\underbrace{\chi_{A'} * \cdots * \chi_{A'}}_{k} * \underbrace{\chi_{-A'} * \cdots * \chi_{-A'}}_{l})(x)/4^{k+l} \tag{28}$$

for all $x \in \mathbf{Z}_p$. In the case

$$(\underbrace{\chi_{A'} * \cdots * \chi_{A'}}_{k} * \underbrace{\chi_{-A'} * \cdots * \chi_{-A'}}_{l})(x) \geq (\varepsilon_2 p)^{k+l-1}, \tag{29}$$

by (28) we have

$$(\underbrace{\chi_1 * \cdots * \chi_1}_{k} * \underbrace{\chi_2 * \cdots * \chi_2}_{l})(x) \geq {\varepsilon_1}^{k+l}(\varepsilon_2 p)^{k+l-1}/4^{k+l}. \tag{30}$$

Now we show that the number of elements $x \in \mathbf{Z}_p$ such that satisfying (29) and $(\underbrace{\chi_A * \cdots * \chi_A}_{k} * \underbrace{\chi_{-A} * \cdots * \chi_{-A}}_{l})(x) = 0$, does not exceed $\varepsilon_3 p$. Denote the set of such elements by $\mathbf{F}$. Observe, that for every $x \in \mathbf{F}$

$$|(\underbrace{\chi_A * \cdots * \chi_A}_{k} * \underbrace{\chi_{-A} * \cdots * \chi_{-A}}_{l})(x) - (\underbrace{\chi_1 * \cdots * \chi_1}_{k} * \underbrace{\chi_2 * \cdots * \chi_2}_{l})(x)|^2 \geq$$

$$\geq \frac{{\varepsilon_1}^{2(k+l)}{\varepsilon_2}^{2(k+l-1)}p^{2(k+l-1)}}{4^{2(k+l)}}. \tag{31}$$

By (27) and (31)

$$\alpha^{2(k+l)-3}\delta^2 p^{2(k+l)-1} \geq$$

$$\geq \sum_{x \in \mathbf{Z}_p} \left| (\underbrace{\chi_A * \cdots * \chi_A}_{k} * \underbrace{\chi_{-A} * \cdots * \chi_{-A}}_{l})(x) - (\underbrace{\chi_1 * \cdots * \chi_1}_{k} * \underbrace{\chi_2 * \cdots * \chi_2}_{l})(x) \right|^2 =$$

$$=\sum_{x\in\mathbf{F}}\left|(\underbrace{\chi_A*\cdots*\chi_A}_{k}*\underbrace{\chi_{-A}*\cdots*\chi_{-A}}_{l})(x)-(\underbrace{\chi_1*\cdots*\chi_1}_{k}*\underbrace{\chi_2*\cdots*\chi_2}_{l})(x)\right|^2+$$

$$+\sum_{x\in(\mathbf{Z}_p\setminus\mathbf{F})}\left|(\underbrace{\chi_A*\cdots*\chi_A}_{k}*\underbrace{\chi_{-A}*\cdots*\chi_{-A}}_{l})(x)-(\underbrace{\chi_1*\cdots*\chi_1}_{k}*\underbrace{\chi_2*\cdots*\chi_2}_{l})(x)\right|^2$$

$$\geq|\mathbf{F}|\frac{{\varepsilon_1}^{2(k+l)}{\varepsilon_2}^{2(k+l-1)}p^{2(k+l-1)}}{4^{2(k+l)}}+$$

$$+\sum_{x\in(\mathbf{Z}_p\setminus\mathbf{F})}\left|(\underbrace{\chi_A*\cdots*\chi_A}_{k}*\underbrace{\chi_{-A}*\cdots*\chi_{-A}}_{l})(x)-(\underbrace{\chi_1*\cdots*\chi_1}_{k}*\underbrace{\chi_2*\cdots*\chi_2}_{l})(x)\right|^2.$$

This implies

$$|\mathbf{F}|\leq\frac{4^{2(k+l)}\alpha^{2(k+l)-3}\delta^2}{{\varepsilon_1}^{2(k+l)}{\varepsilon_2}^{2(k+l-1)}}p\leq\varepsilon_3 p.$$

□

3 The proof of Theorem 1

3.1 The upper bound

Let k,l be nonnegative integers with $k+l\geq 2$. Suppose that s satisfies $es(k+l+1)\leq 2^s$. We divide a partition of $\mathbf{SS}_{k,l}(\mathbf{Z}_p)$ into two parts:

$$\mathbf{SS}_{k,l}(\mathbf{Z}_p)=\mathbf{SS}'_{k,l,s}(\mathbf{Z}_p)\cup\mathbf{SS}''_{k,l,s}(\mathbf{Z}_p), \tag{32}$$

where

$$\mathbf{SS}'_{k,l,s}(\mathbf{Z}_p)=\{B\in\mathbf{SS}_{k,l}(\mathbf{Z}_p):B=kA-lA\ \text{ and }\ |A|\leq p/(k+l+1)s\},$$

$$\mathbf{SS}''_{k,l,s}(\mathbf{Z}_p)=\{B\in\mathbf{SS}_{k,l}(\mathbf{Z}_p):B=kA-lA\ \text{ and }\ |A|>p/(k+l+1)s\}.$$

It is obvious that

$$|\mathbf{SS}_{k,l}(\mathbf{Z}_p)|\leq|\mathbf{SS}'_{k,l,s}(\mathbf{Z}_p)|+|\mathbf{SS}''_{k,l,s}(\mathbf{Z}_p)|. \tag{33}$$

Since every set $A\subseteq\mathbf{Z}_p$ generates one set of the form $kA-lA$ we obtain

$$\left|\mathbf{SS}'_{k,l,s}(\mathbf{Z}_p)\right|\leq|\mathbf{T}_{k+l,s}(\mathbf{Z}_p)|. \tag{34}$$

By (7) and (34) we have

$$\left|\mathbf{SS}'_{k,l,s}(\mathbf{Z}_p)\right| \leq 2^{p/(k+l+1)}. \tag{35}$$

Now we prove an upper bound for $|\mathbf{SS}''_{k,l,s}(\mathbf{Z}_p)|$. Suppose that the cardinality of $A \subseteq \mathbf{Z}_p$ is larger than $p/(k+l+1)s$. Let p be a prime number such that for some nonnegative integers $k, l, L > 0$ and positive real numbers $\varepsilon_1, \varepsilon_2$ and ε_3 the condition (11) is fulfilled. By Lemma 9 there exists a subset A' with properties $(i)-(iii)$. We estimate the number of (k,l)-sumsets $kA-lA$ by counting pairs $(A', kA-lA)$.

Now let $A' \in \mathbf{G}_L(\mathbf{Z}_p)$ be given. For any subset $C \subseteq \mathbf{Z}_p$ we denote by $\overline{C}$ the complement of the subset C in $\mathbf{Z}_p$.

If $|A'| \geq p/(k+l+1)$, then from (iii) of Lemma 9 we obtain that $\overline{kA-lA}$ is a subset of the union of the set $\overline{S_{(\varepsilon_2 p)^{k+l-1},k+l}(\underbrace{\chi_{A'},\ldots,\chi_{A'}}_{k},\underbrace{\chi_{-A'},\ldots,\chi_{-A'}}_{l})}$ and a set of cardinality not exceeding $\varepsilon_3 p$. By Lemma 6 we have

$$|S_{(\varepsilon_2 p)^{k+l-1},k+l}(\underbrace{\chi_{A'},\ldots,\chi_{A'}}_{k},\underbrace{\chi_{-A'},\ldots,\chi_{-A'}}_{l})| \geq$$

$$\geq \min(p,(k+l)|A'|-(k+l)+2) - 2((\varepsilon_2 p)^{k+l-1}p)^{1/2}.$$

If $|A'| \geq p/(k+l+1)$, we obtain

$$|\overline{S_{(\varepsilon_2 p)^{k+l-1},k+l}(\underbrace{\chi_{A'},\ldots,\chi_{A'}}_{k},\underbrace{\chi_{-A'},\ldots,\chi_{-A'}}_{l})}| =$$

$$= p - |S_{(\varepsilon_2 p)^{k+l-1},k+l}(\underbrace{\chi_{A'},\ldots,\chi_{A'}}_{k},\underbrace{\chi_{-A'},\ldots,\chi_{-A'}}_{l})| \leq$$

$$\leq p/(k+l+1) + 2{\varepsilon_2}^{(k+l-1)/2}p^{(k+l)/2} + (k+l-2).$$

It is obvious that for any subset $B \subseteq \mathbf{Z}_p$ the set $kB-lB$ uniquely determines the set $\overline{kB-lB}$. From above it follows that the number of choices $kA-lA$ for given A' of cardinality exceeding $p/(k+l+1)$, is at most

$$2^{p/(k+l+1)+(k+l-2)+(2{\varepsilon_2}^{(k+l-1)/2}p^{(k+l-2)/2}+\varepsilon_3)p}. \tag{36}$$

If $|A'| < p/(k+l+1)$, then by (i) of Lemma 9 we have $|A \setminus A'| \leq \varepsilon_1 p$. This implies that $|A| \leq |A'| + \varepsilon_1 p$. Since every set $A \subseteq \mathbf{Z}_p$ generates exactly one set of form $kA-lA$, we obtain that the number of choices $kA-lA$ for given A' of cardinality not exceeding $p/(k+l+1)$, is at most

$$2^{p/(k+l+1)+\varepsilon_1 p}. \tag{37}$$

From (36), (37), Lemma 8 by applying Lemma 9 with parameters $\varepsilon_1 = \varepsilon_3 = \varepsilon$, $L = 1 + \lfloor 1/\varepsilon \rfloor$ and $\varepsilon_2 = \varepsilon^{2/(k+l-1)} p^{(2-k-l)/(k+l-1)}$, we obtain

$$|\mathbf{SS}''_{k,l,s}(\mathbf{Z}_p)| \le 2^{(p/(k+l+1))+(k+l-2)+o(p)}. \tag{38}$$

From (33), (35) and (38) it follows that

$$|\mathbf{SS}_{k,l}(\mathbf{Z}_p)| \le 2^{p/(k+l+1)} + 2^{(p/(k+l+1))+(k+l-2)+o(p)} = 2^{(p/(k+l+1))+(k+l-2)+o(p)}.$$

3.2 The lower bound

Set $\mathbf{SS}_{k,l}(\mathbf{Z}_p, \mathbb{P}) = \{A : \mathbb{P} \subseteq A,\ A \in \mathbf{SS}_{k,l}(\mathbf{Z}_p)\}$ and $L = \lfloor p/(2(k+l)-1) \rfloor - 1$.

Lemma 10 *Let* k, l *be nonnegative integers with* $k + l \ge 2$, *and let* $\mathbb{P} \subseteq \boldsymbol{Z}_p$ *be arbitrary arithmetic progression of length* $(k+l)(L-1)+1$. *Then there exists a positive constant* $\boldsymbol{C}_{k,l}$ *such that*

$$|\boldsymbol{SS}_{k,l}(\boldsymbol{Z}_p, \mathbb{P})| \ge \boldsymbol{C}_{k,l} 2^{p/(2(k+l)-1)}.$$

Proof. Without loss of generality we assume $\mathbb{P} = \{k - lL, \dots, kL - l\}$. All of our sets will be of the form

$$A = A(B) = k(B \cup \{-(2L+1), 2L+1\}) - l(B \cup \{-(2L+1), 2L+1\}),$$

where $B \subseteq \{-L, -L+1, \dots, L\}$ and $-B = B$. It is easy to see that different sets $B \subseteq \{-L, -L+1, \dots, L\}$ generate different sets $A(B)$.

Set $N_{k,l} = \lceil \log(8(k+l)^2)/\log(4/3) \rceil$ and

$$X = \{0, 1, \dots, N_{k,l}\} \cup \bigcup_{i=1}^{k+l-1} (\lfloor (i+1)L/(k+l) \rfloor - N_{k,l}, \dots, \lceil (i+1)L/(k+l) \rceil).$$

We define the set $B \subseteq \{-L, -L+1, \dots, L\}$ as follows:

$$B = B(C) = -C \cup C \cup X \cup -X,$$

where elements of the set C are picked from the set $\{1, \dots, L\} \setminus X$ randomly, independently, with probability $1/2$. Set

$$Y = \{0\} \cup \{k+l, \dots, (k+l)N_{k,l}\} \cup \bigcup_{i=1}^{k+l-1} \{(i+1)L - (k+l)N_{k,l}, \dots, (i+1)L\}.$$

It is obvious that $-Y \cup Y \subseteq kB - lB$. If $x \notin kB - lB$, then in the representation x in the form $x = x_1 + \cdots + x_k - x_{k+1} - \cdots - x_{k+l}$, there exists at least one x_i ($i \in \{1, \ldots, k+l\}$) such that $x_i \notin B$. Set

$$\mathcal{Q}(x) = \{(x_1, \ldots, x_{k+l}) : x = \sum_{i=1}^{k} x_i - \sum_{j=k+1}^{k+l} x_j, x_1, \ldots, x_{k+l} \in \{-L, \ldots, L\}\},$$

and suppose that $|\mathcal{Q}(x)| = q$.

We say that the vectors $(x_1, \ldots, x_{k+l})$ and $(y_1, \ldots, y_{k+l})$ do not intersect, if $\{x_1, \ldots, x_{k+l}\} \cap \{y_1, \ldots, y_{k+l}\} = \emptyset$.

Set $\mathcal{R}_0 = \{(k+l)N_{k,l} + 1, \ldots, L\}$. We show that for every $x \in -\mathcal{R}_0 \cup \mathcal{R}_0$ the following inequality

$$\mathbf{Pr}(x \notin kB - lB) \leq \left(\frac{3}{4}\right)^{\left\lfloor \frac{|x|}{k+l} \right\rfloor} \tag{39}$$

holds. We have

$$\begin{gathered}
\mathbf{Pr}(x \notin kB - lB) = \\
= \mathbf{Pr}((x_1^1 + \cdots + x_k^1 - x_{k+1}^1 - \cdots - x_{k+l}^1 \notin kB - lB) \& \ldots \\
\ldots \& (x_1^q + \cdots + x_k^q - x_{k+1}^q - \cdots - x_{k+l}^q \notin kB - lB)) \leq \\
\leq \mathbf{Pr}((x_1^{11} + \cdots + x_k^{11} - x_{k+1}^{11} - \cdots - x_{k+l}^{11} \notin kB - lB) \& \ldots \\
\ldots \& (x_1^{1n} + \cdots + x_k^{1n} - x_{k+1}^{1n} - \cdots - x_{k+l}^{1n} \notin kB - lB)) = \\
= \mathbf{Pr}\left((x_1^{11} \notin B \vee \cdots \vee x_{k+l}^{11} \notin B) \& \ldots \& (x_1^{1n} \notin B \vee \cdots \vee x_{k+l}^{1n} \notin B)\right) = \\
= \mathbf{Pr}\left((x_1^{11} \notin B) \vee \cdots \vee (x_{k+l}^{11} \notin B)\right) \cdot \ldots \cdot \mathbf{Pr}\left((x_1^{1n} \notin B) \vee \cdots \vee (x_{k+l}^{1n} \notin B)\right) = \\
= \mathbf{Pr}\left(\overline{(x_1^{11} \in B) \& \ldots \& (x_{k+l}^{11} \in B)}\right) \times \ldots \\
\cdots \times \mathbf{Pr}\left(\overline{(x_1^{1n} \in B) \& \ldots \& (x_{k+l}^{1n} \in B)}\right) = \\
= \left(1 - \mathbf{Pr}\left((x_1^{11} \in B) \& \ldots \& (x_{k+l}^{11} \in B)\right)\right) \times \ldots \\
\cdots \times \left(1 - \mathbf{Pr}\left((x_1^{1n} \in B) \& \ldots \& (x_{k+l}^{1n} \in B)\right)\right),
\end{gathered} \tag{40}$$

where the vectors $(x_1^i, \ldots, x_{k+l}^i) \in \mathcal{Q}(x), i = 1, \ldots, q$, and the vectors $(x_1^{1j}, \ldots, x_{k+l}^{1j})$, $j = 1, \ldots, n \leq q$, are pairwise disjoint.

Note that the vectors $(x-i(k+l-1),\underbrace{i,\dots,i}_{k-1},\underbrace{-i,\dots,-i}_{l})$ are pairwise disjoint for every $x\in -\mathcal{R}_0$, where $-\lfloor |x|/(k+l)\rfloor \le i\le -1$, and $x\in\mathcal{R}_0$, where $1\le i\le \lfloor x/(k+l)\rfloor$. From this and (40) we obtain the inequality (39).

Set $\mathcal{L}_j=\{jL+1,\dots,(j+1)L-(k+l)N_{k,l}-1\}$, $j=1,\dots,k+l-1$. Similarly to the inequality (39) we have

$$\mathbf{Pr}(x\notin kB-lB)\le\left(\frac{3}{4}\right)^{\left\lfloor\frac{(j+1)L-|x|}{k+l}\right\rfloor},\tag{41}$$

where $x\in-\mathcal{L}_j\cup\mathcal{L}_j$, $j=1,\dots,k+l-1$.

From (39) and (41) it is easy to see that

$$\mathbf{Pr}\left(\mathbb{P}\nsubseteq kB-lB\right)\le(k+l)\sum_{x\ge(k+l)N_{k,l}+1}\left(\frac{3}{4}\right)^{\left\lfloor\frac{x}{k+l}\right\rfloor}.\tag{42}$$

Note that if $N_{k,l}\ge\log\left(8(k+l)^2\right)/\log(4/3)$, the right-hand side of (42) does not exceed $1/2$. This leads that there exists at least $2^{L-(k+l)N_{k,l}-1}$ subsets $B\subseteq\{-L,-L+1,\dots,L\}$ such that $\mathbb{P}\subseteq kB-lB$. $\square$

Let k,l be nonnegative integers with $k+l\ge2$, and let $\mathbb{P}\subseteq\mathbf{Z}_p$ be arbitrary arithmetic progression of length $(k+l)(L-1)+1$. By Lemma 10 we have

$$|SS_{k,l}(\mathbf{Z}_p)|\ge|\mathbf{SS}_{k,l}(\mathbf{Z}_p,\mathbb{P})|\ge\mathbf{C}_{k,l}2^{p/(2(k+l)-1)}.$$

Acknowledgements

The author expresses his gratitude to his supervisor Prof. A. A. Sapozhenko for the statement of the problem and attention to this work. The research was supported by RFBR project Nr 10-01-00768-a.

References

[1] B. Green, I. Ruzsa, Counting sumsets and sum-free sets modulo a prime, *Studia Sci. Math. Hungarica* **41,** 3 (2004) 285–293. ⇒33, 36

[2] M. B. Nathanson, *Additive Number Theory: Inverse Problems and the Geometry of Sumsets.* Vol. 165. Graduate Texts in Mathematics, Springer-Verlag, New York, 1996. ⇒35

[3] J. M. Pollard, A generalization of the theorem of Cauchy and Davenport, *J. London Math. Soc.* **8,** 2 (1974) 460–462. ⇒35

[4] A. A. Sapozhenko, *Dedekind Problem and the Method of Boundary Functionals*, Fizmatlit, Moscow, 2009. ⇒36

Received: March 8, 2012 • Revised: April 16, 2012

Acta Univ. Sapientiae, Informatica, **4,** 1 (2012) 48–66

A novel Hopfield neural network approach for minimizing total weighted tardiness of jobs scheduled on identical machines

Norbert FOGARASI
Department of Telecommunications;
Budapest University of Technology and Economics Budapest, Hungary;
email: `fogarasi@hit.bme.hu`

Kálmán TORNAI
Faculty of Information Technology;
Pázmány Péter Catholic University
Budapest, Hungary;
email: `tornai.kalman@itk.ppke.hu`

János LEVENDOVSZKY
Department of Telecommunications; Budapest
University of Technology and Economics Budapest,
Hungary;
email: `levendov@hit.bme.hu`

Abstract. This paper explores fast, polynomial time heuristic approximate solutions to the NP-hard problem of scheduling jobs on N identical machines. The jobs are independent and are allowed to be stopped and restarted on another machine at a later time. They have well-defined deadlines, and relative priorities quantified by non-negative real weights. The objective is to find schedules which minimize the total weighted tardiness (TWT) of all jobs. We show how this problem can be mapped into quadratic form and present a polynomial time heuristic solution based on the Hopfield Neural Network (HNN) approach. It is demonstrated, through the results of extensive numerical simulations, that this solution outperforms other popular heuristic methods. The proposed heuristic is both theoretically and empirically shown to be scalable to large problem sizes (over 100 jobs to be scheduled), which makes it applicable to grid computing scheduling, arising in fields such as computational biology, chemistry and finance.

Computing Classification System 1998: G.1.6
Mathematics Subject Classification 2010: 90C27
Key words and phrases: scheduling theory, artificial neural networks, Hopfield neural network, total weighted tardiness problem, quadratic optimization

1 Introduction

With the advent of grid computing, the classical science of scheduling theory has gained a new sphere of applications. Methods which were originally developed for decision making around limited resources in manufacturing and service industries have been adopted in the areas of computer science, telecommunication and other computationally intensive disciplines such as computational biology, chemistry and finance [5, 22]. Specifically, in the area of computational finance, where this paper sources its motivation, the problems of portfolio selection, pricing and hedging of complex financial instruments requires an enormous amount of computational resources whose optimal usage is of utmost importance to investment banks. The prices and risk sensitivity measures of complex portfolios need to be reevaluated daily, for which an overnight batch of calculations is scheduled and performed for millions of financial transactions, utilizing thousands of computing nodes. Each job has a well-defined priority and required completion time for availability of the resulting figures to the trading desk, risk managers and regulators. The jobs can generally be stopped and resumed at a later point on a different machine which is referred to as *preemption* in scheduling theory. For simplicity of modeling the problem, machines are generally assumed to be *identical* and there is a known, constant number of machines available.

The problem of finding optimal schedules for jobs running on identical machines has been extensively studied over the last three decades. Sahni [25] presents an $O(n \log mn)$ algorithm to construct a *feasible* schedule, one that meets all deadlines, if one exists, for n jobs and m machines. The basic idea of the algorithm is to schedule jobs with earliest due dates first, but fill up machines with smaller jobs if possible. Note that this method allows the development of an algorithm to compute the minimal amount of unit capacity for which a feasible schedule exists. This result has been extended to machines with identical functionality but different processing speed, termed *uniform machines*, and jobs with both starting times and deadlines [19]. However, the scheduling task becomes more difficult when a feasible schedule does not exist and the goal is to minimize some measure of delinquency, often termed *tardiness.* Tardiness of an individual job under a given schedule is defined as the amount of time by which the job finishes after its prescribed deadline, and is considered to be zero if the job finishes on or before the deadline.

In case of minimizing the maximum tardiness across all jobs, Lawler [14] shows that the problem is solvable in polynomial time, even with some precedence constraints. Martel [19] also used his construction to create a polynomial time algorithm to find the schedule which minimizes maximum lateness. However, if our measure concerns the total tardiness instead of the maximal one, then even the single machine, total tardiness problem (without weights) was proven to be NP-hard by Du et al. [7]. A pseudopolynomial algorithm has been developed by Lawler [13] for this problem, using dynamic programming, but this is for the 1-machine problem and does not have good practical runtime characteristics.

In practical applications, jobs often have relative priorities associated with them, represented by positive real *weights* and the objective becomes minimizing the total weighted tardiness (TWT). Once the NP-hardness of the TWT problem was established, most of the research work on the problem concerned the development of fast, heuristic algorithms. Dogramaci et al. [6] propose a simple heuristic for the total (non-weighted) tardiness problem without preemption. Rachamadugu et al. [23] then studied the identical machine, total weighted tardiness problem without preemption. They proposed a myopic heuristic and compared this to earliest due date (EDD), weighted shortest processing time (WSPT) and Montagnes rule on small problem sizes (2 or 5 jobs in total). Azizoglu et al. [3] worked on an algorithm to find optimal schedule for the unweighted total tardiness problem without preemption, but their branch and bound exponential algorithm is too slow, in practice, for problems with more than 15 jobs. Armentano et al. [2] examined the non-weighted problem without preemption, and starting from the KPM heuristic of Koulamas [11] improved upon it, using tabu search. Guinet [8] applies simulated annealing to solve the problem with uniform and identical machines and a lower bound is presented in order to evaluate the performance of the proposed method. More recently, Sen et al. [26] surveyed the existing heuristic algorithms for the single-machine total tardiness and total weighted tardiness problems while Biskup et al. [4] did this for the identical machines total tardiness problem and also proposed a new heuristic. Akyol et al. [1] provide an excellent recent review of artificial neural network based approaches to scheduling problems and proposes a coupled gradient network to solve the weighted earliness plus tardiness problem on multiple machines. The feasibility of the method is illustrated on a single 8-job scheduling problem.

We suggest a novel heuristic for the TWT problem, based on the Hopfield Neural Network approach which is shown to perform better than existing simple heuristics and has desirable scaling characteristics. Maheswaran et al. [16] applied a similar approach to the single machine TWT problem and their results were encouraging for a specific 10-job problem.

In this paper, we first map the problem into quadratic optimization and then the Hopfield net is used to provide fast polynomial time, heuristic solution. The topics are organized as follows: in Section 2, we present the problem formulation and the model used, in Section 3 existing heuristics are defined and explained, in Section 4 our novel HNN approach is introduced, in Section 5 numerical results are presented and finally in Section 6 some conclusions are drawn and directions for future research are outlined.

2 Problem formulation

In this section, we give a formal presentation for the problem of optimally scheduling jobs on finite number of identical processors under constraints on the completion times. The basic formalism is the following:

- Given N jobs with sizes $\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_N\} \in \mathbb{N}^N$. The processing of the jobs can be stopped and resumed at any time, so the processing time units of each job need not be contiguous. In the literature this condition is known as preemption and also assumes a task started on one machine can continue on another [5].
- For each job a cutoff time is prescribed by $\mathbf{K} = \{K_1, K_2, K_3, \ldots, K_N\} \in \mathbb{N}^N$. This constraint defines the time within which the job is to be completed.
- The constant number of processors, the capacity of the system is denoted by $V \in \mathbb{N}$.
- We are also given a vector $\mathbf{w} = \{w_1, w_2, w_3, \ldots, w_N\} \in \mathbb{R}^N$,$w_i \geq 0, \forall i = 1, \ldots, N$ denoting the relative priority (or weight) of each job, which can be used in the definition of the objective function.

A schedule is represented by a binary matrix $\mathbf{C} \in \{0, 1\}^{N \times L}$ where $C_{i,j} = 1$ if job i is being processed at time slot j, and L denotes the length of the schedule. An example is given in (1) where the parameters are the following: $V = 2$, $N = 3$, $\mathbf{x} = \{2, 3, 1\}$, $\mathbf{K} = \{3, 3, 3\}$.

$$\mathbf{C} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{1}$$

The first row in (1) denotes the fact that under this schedule $\mathbf{C}$, the first job is processed in time steps 1 and 3 (note that preemption is used as the processing of this job is not continuous) and therefore the prescribed size 2 of this job completes within the prescribed cutoff time of time step 3. Similarly, the 3 units of the second job complete within the cutoff time of 3 and the third job is completed ahead of the cutoff time on time step 2. Summing the columns of matrix $\mathbf{C}$, we see that the maximal capacity of $V = 2$ is fully utilized on each time step.

In order to evaluate the effectiveness of a given schedule $\mathbf{C}$, we define tardiness of a job as follows:

$$T_i = \max\left(0, F_i - K_i\right), \tag{2}$$

where F_i is the actual finish time of job i under schedule $\mathbf{C}$: $F_i = \arg\max_j \{C_{i,j} = 1\}$ (The position of the last 1 in the ith row in scheduling matrix $\mathbf{C}$.)

The problem can now be stated formally as follows:

$$\mathbf{C}_{\text{opt}} := \arg\min_{\mathbf{C}} \sum_{i=1}^{N} w_i T_i. \tag{3}$$

Under the following constraints:

- The sizes of the scheduled jobs in the scheduling matrix are equal to the predefined amounts:

$$\sum_{j=1}^{L} C_{i,j} = x_i, \forall i = 1, \dots, N. \tag{4}$$

- The number of scheduled jobs at any given time instant does not exceed the capacity of the system:

$$\sum_{i=1}^{N} C_{i,j} \leq V, \forall j = 1, \dots, L. \tag{5}$$

We revisit our previous example with a minor change: $V = 2$, $N = 3$, $\mathbf{x} = \{2, 3, 2\}$, $\mathbf{K} = \{3, 3, 3\}$ and a weight vector $\mathbf{w} = \{3, 2, 1\}$. It can be observed that there is no solution, in which all jobs are completed before their cutoff times. A minimal weighted tardiness solution is the following:

$$\mathbf{C} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

3 Existing heuristic methods

Given the NP-hardness of the scheduling task as shown by Du et al. [7], and therefore the amount of time it would take to find the exact, optimal solution, in most real-world settings the pragmatic approach of finding a fast, sub-optimal, but good solution is followed. As outlined in the introduction, this has been a very active field of research, and most studies use the Earliest Due Date (EDD) and Weighted Shortest Process Time (WSPT) heuristics as benchmarks for evaluating the proposed algorithms. In addition to these, we also outline the recently developed Load Balancing Scheduling (LBS) heuristic and a newly proposed Largest Weighted Process First (LWPF) heuristic. Furthermore, we also outline a random processing method which is used as a low benchmark for our testing results.

3.1 Earliest due date (EDD) heuristic

The EDD heuristic orders the sequence of jobs to be executed from the job with the earliest due date to the job with the latest due date. Using the notation of Section 2, we relabel the job indices so that the following inequality holds:

$$K_1 \leq K_2 \leq \ldots \leq K_N.$$

Once this ordering is determined, the jobs are allocated to the machines in this order, always utilizing the maximum available capacity. Once a job finishes and capacity is freed up, the next job using the above ordering is scheduled on the freed up machine. It has been shown in [22] that EDD finds the optimal schedule when one wants to minimize the maximum tardiness on a single machine. However, we note that when the objective function includes the relative priorities of jobs, EDD is at a severe disadvantage, as it does not include consideration of the weights.

3.2 Weighted shortest processing time (WSPT) heuristic

The WSPT method is analogous to the EDD method in that it orders the jobs according to well-defined criteria and then schedules the jobs according to this ordering, utilizing the maximum available capacity available. Once a job finishes and capacity is freed up, the next job using the ordering is scheduled. Using the notation of Section 2, the jobs are ordered such that the below holds:

$$\frac{x_1}{w_1} \leq \frac{x_2}{w_2} \leq \ldots \leq \frac{x_N}{w_N}.$$

3.3 Largest weighted process first (LWPF) heuristic

This heuristic is analogous to the EDD and the WSPT heuristics with the jobs ordered according to the following inequality:

$$w_1 \geq w_2 \geq \ldots \geq w_N.$$

This is a simple heuristic which allows us get a sense of the importance of considering the weights versus the cutoff times. We have not seen this simple heuristic mentioned anywhere in the literature, but it turns out to have quite good empirical characteristics which is one of the many contributions of this paper.

3.4 Load Balancing Scheduling (LBS) algorithm

Laszlo et al. [12] suggest a novel deterministic, polynomial time algorithm for scheduling jobs with cutoff times, in the context of load balancing for wireless sensor networks. The basic idea of the algorithm is to start scheduling the jobs backwards from the maximal cutoff time, in order of decreasing cutoff times, always ensuring full capacity is utilized. For a more formal definition of the algorithm see [12]. We note that whilst the algorithm has proven to be extremely successful for load balancing, it does not take into account the weights of the jobs and therefore will suffer from the same shortcomings in our setting as EDD.

3.5 Random method

In this method, the jobs are ordered randomly and are scheduled in such a way as to always fully utilize the maximum available capacity. This unsophisticated heuristic is useful as a low benchmark by repeating it a number of times and taking the best solution over the multiple repeats.

4 Novel Hopfield neural network (HNN) based approach

Since the number of possible binary matrices grows exponentially with the number of nodes and the length of the schedule, exhaustive search with complexity $O\left(2^{N \cdot L}\right)$ is generally computationally infeasible to solve the optimization problem at hand. Thus, our goal is to develop a polynomial time approximate solution by mapping the problem to an analogous quadratic optimization problem, similar to the approach of Levendovszky et al. [15] and Treplan et al. [27].

We first review the methods available for optimizing equations in quadratic form. Let us assume that matrix $\mathbf{W}$ is a symmetric matrix of size $n \times n$ and vector $\mathbf{b}$ is of length n. We seek the optimal n dimensional vector $\mathbf{y}$ which minimizes the following quadratic function [21]:

$$f(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^T\mathbf{W}\mathbf{y} + \mathbf{b}^T\mathbf{y}, \tag{6}$$

subject to one or more constraints of the form of

$$\begin{aligned}\mathbf{A}\mathbf{y} &\leq \mathbf{v},\\ \mathbf{B}\mathbf{y} &= \mathbf{u}.\end{aligned}$$

If the problem at hand contains only linear constraints then it can be solved as presented by Murty et al. [20]. In other cases, if the matrix $\mathbf{W}$ is positive definite, then the function $f(\mathbf{y})$ is convex and the problem can be solved with the ellipsoid method presented by Zhi-Quan et al. [28]. When $\mathbf{W}$ is indefinite, the problem is NP-hard (for details see [24]).

A frequently used powerful, heuristic algorithm to solve quadratic optimization problems is the Hopfield Neural Network (HNN). This neural network is described by the following state transition rule:

$$\mathbf{y}_i(k+1) = \operatorname{sgn}\left(\sum_{j=1}^{N}\hat{W}_{ij}y_j(k) - \hat{b}_i\right), i = \mathrm{mod}_N k,$$

where

$$\begin{aligned}\mathbf{d} &= -\operatorname{diag}(\mathbf{W}),\\ \widehat{\mathbf{W}} &= -\mathbf{W} - \operatorname{diag}(\mathbf{d}),\\ \widehat{\mathbf{b}} &= \mathbf{b} - \tfrac{1}{2}\mathbf{d}.\end{aligned}$$

Using the Lyapunov method, Hopfield [10] proved that the HNN converges to its fixed-point, as a consequence the HNN minimizes a quadratic Lyapunov function:

$$\mathcal{L}(\mathbf{y}) := -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\hat{W}_{ij}y_iy_j + \sum_{i=1}^{N}y_i\hat{b}_i = -\frac{1}{2}\mathbf{y}^T\hat{\mathbf{W}}\mathbf{y} + \hat{\mathbf{b}}^T\mathbf{y}.$$

Thus, the HNN is able to solve combinatorial optimization problems in polynomial time under special conditions as it has been demonstrated by Mandziuk

[18, 17]. Using this method in practice we have to handle the following problem: in some cases the HNN method may get stuck in local minimal point of its Lyapunov function.

These methods motivate us to map the existing optimization problem into quadratic form. First the binary scheduling matrix $\mathbf{C}$ is mapped into a binary column vector $\mathbf{y}$ as follows:

$$\mathbf{C} = \begin{pmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,L} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,L} \\ \vdots & \vdots & \ddots & \vdots \\ C_{N,1} & C_{N,2} & \cdots & C_{N,L} \end{pmatrix} \rightarrow$$

$$\mathbf{y} = (C_{1,1}, C_{1,2}, \cdots, C_{1,L}, C_{2,1}, \cdots, C_{2,L}, C_{N,1}, \cdots, C_{N,L})^T.$$

The original objective function (3) is elaborated as follows:

$$\min_{\mathbf{C}} \sum_{i=1}^{N} w_i \left(\max \left(0, \arg\max_j \{C_{i,j} = 1\} - K_i \right) \right).$$

This objective is equivalent to:

$$\min_{\mathbf{C}} \sum_{i=1}^{N} w_i \left(\sum_{j=K_i+1}^{L} C_{i,j} \right).$$

The minimization problem is thus equivalent to:

$$\mathbf{C}_{\text{opt}} := \arg\min_{\mathbf{C}} \sum_{i=1}^{N} \sum_{j=K_i+1}^{L} w_i C_{i,j}^2.$$

Therefore the mapping required is:

$$-\frac{1}{2}\mathbf{y}^T \mathbf{W}_A \mathbf{y} + \mathbf{b}_A{}^T \mathbf{y} = \sum_{i=1}^{N} \sum_{j=K_i+1}^{L} w_i C_{i,j}^2.$$

The solution is the following:

$$\mathbf{b}_A = \mathbf{0}_{NL\times 1},$$

$$\mathbf{W}_A = -2 \begin{pmatrix} \mathbf{D}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_2 & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{D}_N \end{pmatrix},$$

where

$$\mathbf{D}_j = \begin{pmatrix} \mathbf{0}_{K_j \times K_j} & \mathbf{0}_{K_j \times (L-K_j)} \\ \mathbf{0}_{(L-K_j)\times K_j} & w_j \mathbf{I}_{(L-K_j)\times(L-K_j)} \end{pmatrix} \in \mathrm{R}^{L\times L}.$$

Having transformed the objective function to a quadratic form, we now turn our attention to doing the same with the two constraints outlined in (4) and (5).

The constraints in (4) can be rewritten as follows:

$$\forall i : \sum_{j=1}^{L} C_{i,j} = x_i \rightarrow \min_{\mathbf{C}} \sum_{i=1}^{N} \left(\left(\sum_{j=1}^{L} C_{i,j} \right) - x_i \right)^2 .$$

This will lead to the following equation:

$$-\frac{1}{2}\mathbf{y}^T \mathbf{W}_B \mathbf{y} + \mathbf{b}_B{}^T \mathbf{y} = \sum_{i=1}^{N} \left(\left(\sum_{j=1}^{L} C_{i,j} \right) - x_i \right)^2 . \tag{7}$$

The solution of equation (7) is:

$$\mathbf{b}_B = 2 \begin{pmatrix} x_{1_{1\times L}} & x_{2_{1\times L}} & \cdots & x_{J_{1\times L}} \end{pmatrix},$$

$$\mathbf{W}_B = -2 \begin{pmatrix} \mathbf{1}_{L\times L} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{1}_{L\times L} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{1}_{L\times L} \end{pmatrix}.$$

Another constraint ensures that the scheduling does not overload the processing units, described in (5). The required transformation is as follows:

$$\forall j : \sum_{i=1}^{N} C_{i.j} = V \rightarrow \min_{\mathbf{C}} \sum_{i=1}^{L} \left(\left(\sum_{j=1}^{N} C_{i,j} \right) - V \right)^2 . \tag{8}$$

Please note the technicality that this constraint may not be relevant for the last few columns where only the remaining jobs have to be scheduled and schedule matrices not exhausting the full capacity should not be penalized. The total length of scheduling can be written as follows:

$$\tilde{L} = \left\lceil \frac{\sum_{i=1}^{N} x_i}{V} \right\rceil,$$

$$L = \max\left(\tilde{L}, \max_i x_i, \max_i K_i\right).$$

The number of columns where capacity V needs to be fully utilized is the following:

$$M = \max\left(1, \max_i x_i - \hat{L} + 1\right). \tag{9}$$

Taking into consideration (9) the mapping of (8) can be described by the following equation:

$$-\frac{1}{2}\mathbf{y}^T\mathbf{W}_B\mathbf{y} + \mathbf{b}_B{}^T\mathbf{y} = \sum_{i=1}^{M}\left(\left(\sum_{j=1}^{N} C_{i,j}\right) - V\right)^2.$$

The solution of this equation is the following:

$$\mathbf{b}_C = \left[\mathbf{V}_{M\times 1}, \mathbf{0}_{(L-M)\times 1}, \mathbf{V}_{M\times 1}, \mathbf{0}_{(L-M)\times 1}, \ldots, \mathbf{V}_{M\times 1}, \mathbf{0}_{(L-M)\times 1}\right],$$

$$\mathbf{W}_C = -2\begin{pmatrix} \mathbf{D} & \mathbf{D} & \cdots & \mathbf{D} \\ \mathbf{D} & \mathbf{D} & \cdots & \mathbf{D} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{D} & \mathbf{D} & \cdots & \mathbf{D} \end{pmatrix},$$

where

$$\mathbf{D} = \begin{pmatrix} \mathbf{I}_{M\times M} & \mathbf{0}_{M\times(L-M)} \\ \mathbf{0}_{(L-M)\times M} & \mathbf{0}_{(L-M)\times(L-M)} \end{pmatrix}.$$

We can combine these mappings into the form of equation (6) as follows:

$$\mathbf{W} = \alpha\mathbf{W}_A + \beta\mathbf{W}_B + \gamma\mathbf{W}_C \in \mathrm{R}^{NL\times NL},$$

and

$$\mathbf{b} = \alpha\mathbf{b}_A + \beta\mathbf{b}_B + \gamma\mathbf{b}_C \in \mathrm{R}^{NL\times 1}.$$

Note that the objectives can be controlled with heuristic constants α, β and γ in order to strike an appropriate balance between the weights of different requirements. Having this quadratic form at hand, we can apply the HNN to provide an approximate solution to the optimization problem in polynomial time.

5 Numerical results

In this section, the performance of the HNN approach is investigated and is compared to the performance of other heuristics.

5.1 Simulation method

Each method outlined in Section 3 and the HNN method outlined in Section 4 has been tested by simulation on a large and diverse set of input parameters with the aim to characterize the algorithms empirically on scheduling problems of different size of jobs. The algorithms were implemented in MATLAB and tests were run in this simulation environment with randomly generated parameters such as size of jobs, cutoff times, and weights. The size of each job and its cutoff time and corresponding weight are generated as follows:

$$\mathbf{x}_i = \text{random}\left(\left[1, c_1\right]\right), \tag{10}$$

$$\mathbf{K}_i = \mathbf{x}_i + \text{random}\left(\left[c_1, 1.5 \cdot c_1\right]\right), \tag{11}$$

$$\mathbf{w}_i = \text{random}\left(\left[1, c_2\right]\right), \tag{12}$$

where random (Θ) produces a uniformly distributed random integer value in range Θ. In our simulations the constant c_1 equals 10 and c_2 equals 5. The number of processors (V) is determined as follows:

$$V = 0.25 \cdot J. \tag{13}$$

The problem is expected to be solved without tardiness when the capacity is $V = J$. Therefore (13) ensures that there is a high likelihood of tardiness associated with the generated problem.

For each problem size, 100 different problems were generated randomly, using (10)-(13) and the results of the methods were compared in each case.

The random method was repeated 1000 times for each problem and the best solution was used. Furthermore, the HNN method was repeated 1000 times for each problem with different random starting points and the best solution was used. In addition, the heuristic parameters α, β, and γ were adjusted between the simulations in order to provide a good balance between optimizing the objective function, but also meeting the required constraints to produce a valid scheduling matrix. To adjust the heuristical parameters we used Algorithm 1.

Even so, in some cases the HNN is unable to provide a valid solution, because the prescribed constraints on job sizes and capacity are violated. Therefore, we introduced an error correction function (see Algorithm 2) in order to cut and replace the unnecessary 1's in the scheduling matrix. In our simulations parameter e equals 5. All of the results presented in the following sections concern valid schedules meeting the required constraints.

Algorithm 1 Algorithm for adjusting the heuristical parameters

```
Require: $\mathbf{x}, \mathbf{K}, V, e$
  $\alpha \leftarrow 0.1, \beta \leftarrow 5, \gamma \leftarrow 5$
  $i \leftarrow 0$
  repeat
    $i \leftarrow i + 1$
    $\mathbf{C}_i \leftarrow \mathrm{HNN}(\mathbf{x}, \mathbf{K}, V, \alpha, \beta, \gamma)$
    $\alpha \leftarrow \alpha + 0.01$
  until $\mathrm{errors}(\mathbf{C}_i) \leq e$
  for $k = 1 \rightarrow i$ do
    $\mathbf{C}_k \leftarrow \mathrm{correct}(\mathbf{C}_k)$
    $\mathbf{T}_k \leftarrow \mathrm{calculateTWT}(\mathbf{C}_k)$
  end for
  return $\min(\mathbf{T})$
```

Algorithm 2 Correction algorithm for the scheduling matrix produced by the HNN

```
Require: $\mathbf{x}, \mathbf{w}, \mathbf{K}, V, \mathbf{C}$
  for $k = 1 \rightarrow L$ do
    while $\sum_{i=1}^{N} \mathbf{C}_{i,k} > V$ do
      Remove $1$ from row $j$, where $j$ is the row in column $k$ with minimal
      weight that has $C_{j,k} = 1$
    end while
  end for
  for $k = 1 \rightarrow N$ do
    while $\sum_{i=1}^{L} \mathbf{C}_{k,i} > \mathbf{x}_k$ do
      Remove $1$ from row $k$ from the column $l$, where $l$ is the righternmost
      column in row $k$ such that $C_{k,l} = 1$
    end while
    while $\sum_{i=1}^{L} \mathbf{C}_{k,i} < \mathbf{x}_k$ do
      Add $1$ to row $k$ in column $l$ where $l$ is the lefternmost column where $1$
      can be added without violating the capacity constraint $V$.
    end while
  end for
  return $\mathbf{C}$
```

5.2 Average total weighted tardiness of the different methods

The first simulation compares the average total weighted tardiness provided by the algorithms for different problem sizes.

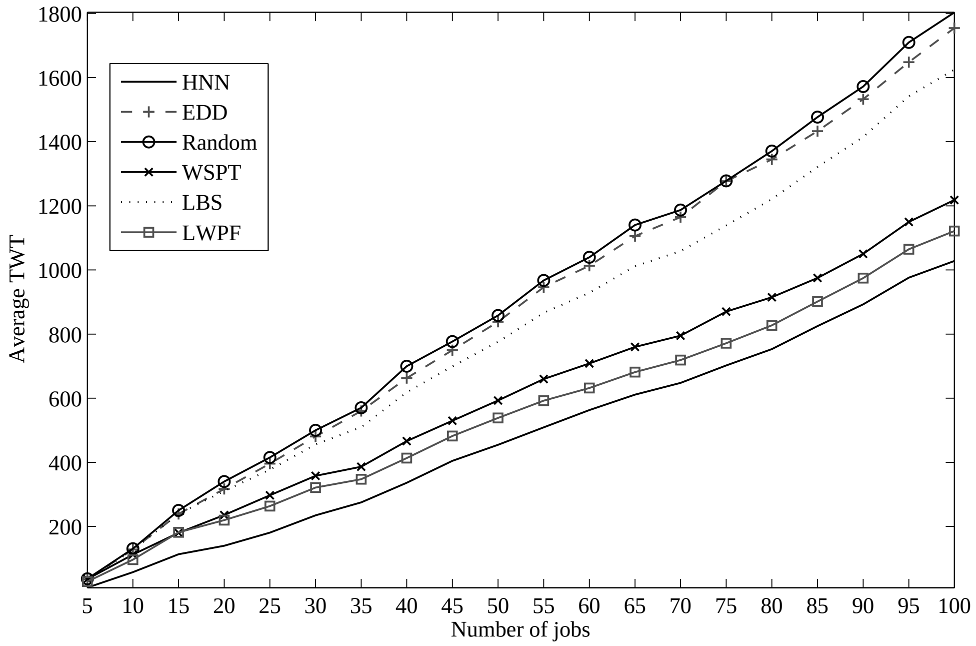

Figure 1: Average TWT produced by each heuristic over randomly generated problems depicted as a function of the number of jobs in the problem.

Figure 1 demonstrates that the best solution achieved by the HNN produces better average TWT for all problem sizes than any of the other heuristics. The performances of the EDD, LBS and random methods are, on average, worse than the WSPT and LWPF heuristics and the HNN for all problem sizes. This shows that consideration of the weights in the optimization problem is critical to reach near-optimal TWT schedules. It is also clearly visible that the HNN consistently outperforms all other methods for all problem sizes. Note also that the simple LWPF heuristic, proposed by us, outperforms the WSPT heuristic widely used as a benchmark in the literature.

5.3 Detailed comparison of the HNN performance versus other heuristics

In this section, we quantify how much better the solution provided by the HNN is, in comparison to the EDD, WSPT and LWPF algorithms. Figure 2 depicts the ratio of average TWT produced by the HNN method versus the other heuristics. We conclude that, on average, the best solution of the HNN heavily outperforms the traditional solutions: the total weighted tardiness of the best HNN solution is $25\% - 56\%$ of the EDD, $25\% - 84\%$ of the WSPT, and $34\% - 91\%$ of the LWPF. As the problem size increases, the WSPT and LWPF heuristics produce solutions which are closer to the HNN.

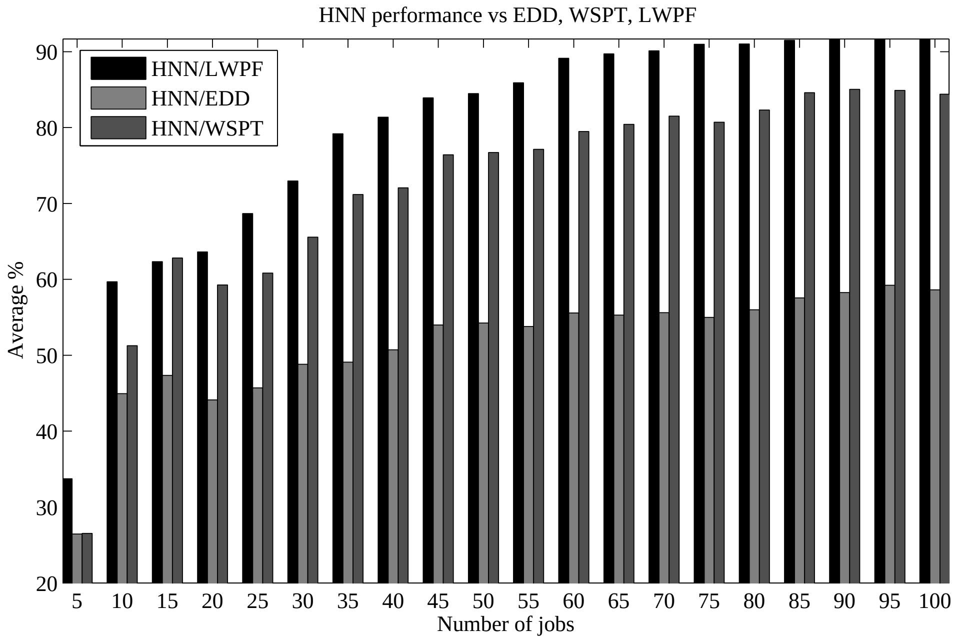

Figure 2: Ratio of average TWT produced by the HNN versus EDD, WSPT and LWPF as a function of the problem size

In order to verify that the HNN consistently outperforms the other heuristics on a wide spectrum of problems, not just on a few selected problems for each given problem size, we ran a more detailed experiment. In this case, we investigated 500 randomly generated problems for each problem size and computed the percentage of times the HNN produced a better solution than

LWPF, the next best method. Table 1 shows the result which quite convincingly proves the general applicability of the HNN to different problem types. Across the spectrum of problem domain, the ratio is higher than 98.5% for all investigated job sizes.

Table 1: Percentage of problems in which the HNN provides an improved solution over the next best heuristic, LWPF

Job size	**5**	**10**	**20**	**25**	**50**	**75**	**100**
	99.9%	100%	99.5%	99.2%	99.3%	98.6%	98.8%

5.4 Runtime characteristics of the investigated algorithms

In this section, we summarize the runtime characteristics of the introduced algorithms. Table 2 contains the theoretical order of convergence of the algorithms as a function of the length of the input parameters.

Table 2: Theoretical order of convergence of the investigated algorithms

Algorithm	**Order of convergence**	**Reference**
Random strategy	$O(\mathsf{L}\cdot\mathsf{N})$	[12]
EDD, WSPT, LWPF	$O(\mathsf{L}\cdot\mathsf{N})$	[16]
LBS	$O\left(\mathsf{L}\cdot\mathsf{N}^2\right)$	[12]
HNN	$O\left(\mathsf{L}^2\cdot\mathsf{N}^2\right)$	[10], [9]
Exhaustive search	$O\left(2^{\mathsf{L}\cdot\mathsf{N}}\right)$	

Note that this table shows the theoretical runtime for a single run of the specified algorithm, whilst we need to run a constant number of the HNN and random iterations. On the other hand, these independent runs can be executed in parallel on several computers, multicore machines or even GPU-based architectures, so the overall runtime should not be significantly different due to the need to run multiple iterations.

Figure 3 depicts the empirical runtime comparison of the different heuristics which confirm the theoretical limits. Although the HNN is the slowest method of the investigated heuristics, in practice we see that near optimal solution of a 100 job problem takes only around 6 seconds / iteration (including the iterative adjustment of the heuristic parameters as per Algorithm 1) on Core i7@3GHz processor, running non-optimized MATLAB code, which is very promising as far as its practical applicability in real life applications is concerned. With further optimization and parallelization on multicore machines or GPGPU

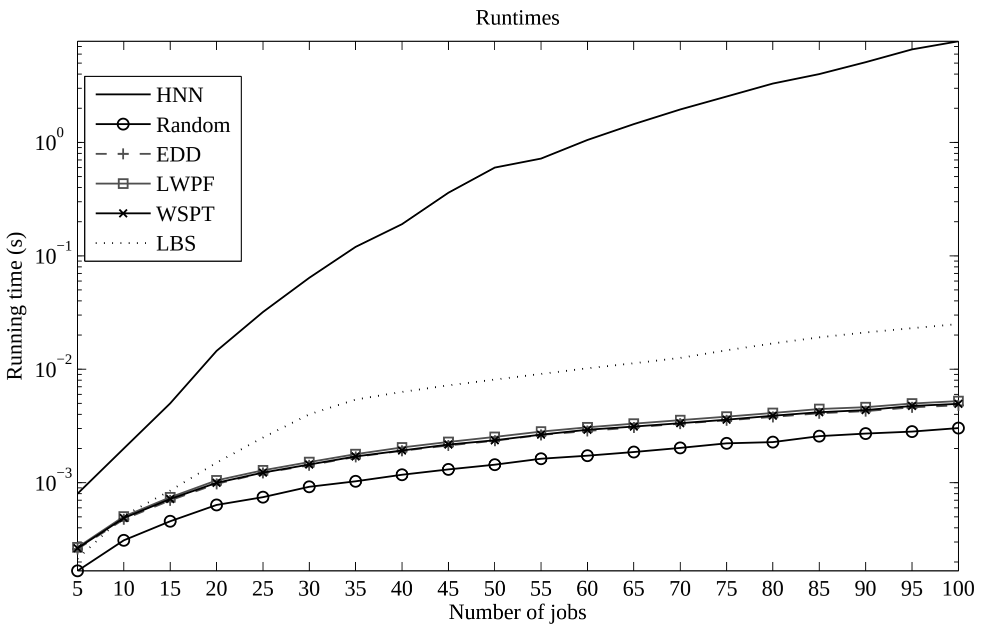

Figure 3: The average runtime of each algorithm per iteration (including the iterative adjustment of the heuristic parameters as per Algorithm 1 for HNN), depicted as a function of the number of jobs. Note that the vertical axis is using a logarithmic scale

technology, the HNN method provides better quality solutions than the other heuristics within acceptable runtimes.

6 Conclusions and directions of future research

In this paper, we studied the NP-hard problem of scheduling jobs with given relative priorities (weights) and deadlines on identical machines, minimizing the TWT measure. We developed a novel heuristic approach, utilizing quadratic programming and the Hopfield neural network and we showed that, in general, it outperforms the EDD and WSPT methods. Furthermore, we have shown that our approach can be applied in real-time settings for small problem sizes and possesses scalability features which are acceptable for real world applications.

More formal methods for the selection of alpha, beta and gamma parameters could be investigated in the future to further improve performance. Another idea to explore is whether a more intelligent selection of the initial point for the HNN algorithm (eg. the result of the LWPF or WSPT heuristics) would improve the algorithms performance over the random starting point which was used in our tests. Finally, the performance of the HNN algorithm needs to be compared to more complex heuristic methods.

References

[1] D. E. Akyol, G. M. Bayhan, Multi-machine earliness and tardiness scheduling problem: an interconnected neural network approach. *Int. J. of Adv. Manuf. Technol.*, **37** (2008) 576–588. ⇒50

[2] V. A. Armentano, D. S. Yamashita, Taboo search for scheduling on identical parallel machines to minimize mean tardiness. *J. of Intell. Manuf.*, **11** (2000) 453–460. ⇒50

[3] M. Azizoglu, O. Kirca, Tardiness minimization on parallel machines. *Int. J. of Production Econ.*, **55** (1998) 163–168. ⇒50

[4] D. Biskup, J. Herrman, J. N. D. Gupta, Scheduling identical parallel machines to minimize total tardiness. *Int. J. of Production Econ.*, **115** (2008) 134–142. ⇒50

[5] P. Brucker, *Scheduling Algorithms*, 5th edn. New York, Springer, 2007. ⇒49, 51

[6] A. Dogramaci, J. Surkis, Evaluation of a heuristic for scheduling independent jobs on parallel identical processors. *Manag. Sci.*, **25**, 12 (1979) 1208–1216. ⇒50

[7] J. Du, J. Y. T. Leung, Minimizing total tardiness on one machine is np-hard. *Math. Oper. Res.*, **15**, 3 (1990) 483–495. ⇒50, 53

[8] A. Guinet, Scheduling independent jobs on uniform parallel machines to minimize tardiness criteria. *J. of Intell. Manuf.*, **6** (1995) 95–103. ⇒50

[9] S. Haykin, *Neural Networks* and Learning Machines, 3rd edn. Prentice Hall, 2008. ⇒63

[10] J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities. In: *Proc. Natl. Acad. Sci. USA*, **79** (1982) 2554–2558. ⇒55, 63

[11] C. P. Koulamas, The total tardiness problem: review and extensions. *Oper. Res.*, **42** (1994) 1025–1041. ⇒50

[12] E. Laszlo, K. Tornai, G. Treplan, J. Levendovszky, Novel load balancing scheduling algorithms for wireless sensor networks. In: *The Fourth Int. Conf. on Communication Theory, Reliability, and Quality of Service*, Budapest, 2011, pp. 54–49. ⇒54, 63

[13] E. L. Lawler, A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Ann. of Discrete Math.*, **1** (1977) 331–342. ⇒50

[14] E. L. Lawler, Preemptive scheduling of precedence-constrained jobs on parallel machines, deterministic and stochastic scheduling. *Proc. NATO Advanced Study and Research Institute on Theoretical Approaches to Scheduling Problems*, 1981, pp. 101–123. ⇒49

[15] J. Levendovszky, A. Olah, K. Tornai, G. Treplan, Novel load balancing algorithms ensuring uniform packet loss probabilities for WSN, *IEEE 73rd Vehicular Technology Conf., Budapest*, 2011. ⇒55

[16] R. Maheswaran, S. G. Ponnambalam, S. D. Nithin, A. S. Ramkumar, Hopfield neural network approach for single machine scheduling problem. In: *Proc. 2004 IEEE Conf. on Cybernetics and Intelligent Systems*, 2004, pp. 850–854. ⇒50, 63

[17] J. Mandziuk, Solving the travelling salesman problem with a Hopfield – type neural network. *Demonstratio Math.*, **29**, 1 (1996) 219–231. ⇒56

[18] J. Mandziuk, B. Macukow, A neural network designed to solve the n-queens problem. *Biol. Cybernet.*, **66**, 4 (1992) 375–379. ⇒56

[19] C. Martel, Preemptive Scheduling with Release Times, Deadlines, and Due Times *J. of the ACM*, **29**, 3 (1982) 812–829. ⇒49

[20] K. G. Murty, Linear complementarity, *linear and nonlinear programming*. Heldermann Springer-Verlag, 1988. ⇒55

[21] J. Nocedal, S. J. Wright, *Numerical Optimization*. Springer-Verlag, 1988. ⇒55

[22] M. Pinedo, *Scheduling – Theory, Algorithms and Systems*, 3rd edn. New York, Springer, 2008. ⇒49, 53

[23] R. M. V. Rachamadugu, T. E. Morton, *Myopic heuristics for the weighted tardiness problem on identical parallel machines*. Tech. rep., (CMU-RI-TR-83-17) Carnegie-Melon University, the Robotics Institute, 1983. ⇒50

[24] S. Sahni, Computationally related problems. *SIAM J. on Comp.* **3** (1974) 262–279. ⇒55

[25] S. Sahni, Preemptive scheduling with due dates. *Oper. Res.*, **27**, 5 (1979) 925–934. ⇒49

[26] T. Sen, J. M. Sulek, P. Dileepan, Static scheduling research to minimize weighted and unweighted tardiness: A state-of-the-art survey. *Int. J. of Production Econ.*, **83** (2003) 1–12. ⇒50

[27] G. Treplan, K. Tornai, J. Levendovszky, Quadratic programming for TDMA scheduling in wireless sensor networks, *Int. J. of Distrib. Sen. Netws.*, 2011. 17 p. ⇒55

[28] L. Zhi-Quan, M. Wing-Kin, M. C. S. Anthony, Y. Yinyu, Z. Shuzhong, Semidefinite relaxation of quadratic optimization problems. *Signal Process. Mag.*, IEEE **27**, 3 (2010) 20–34. ⇒55

Received: March 5, 2012 • Revised: April 16, 2012

Efficient implementations of minimum-cost flow algorithms

Zoltán KIRÁLY
Dept. of Computer Science and
MTA-ELTE Egerváry Research Group
Eötvös Loránd University
Budapest, Hungary
email: kiraly@cs.elte.hu

Péter KOVÁCS
ChemAxon Ltd. and
Dept. of Algorithms and Applications
Eötvös Loránd University
Budapest, Hungary
email: kpeter@inf.elte.hu

Abstract. This paper presents efficient implementations of several algorithms for solving the minimum-cost network flow problem. Various practical heuristics and other important implementation aspects are also discussed. A novel result of this work is the application of Goldberg's recent partial augment-relabel method in the cost-scaling algorithm. The presented implementations are available as part of the LEMON open source C++ optimization library (http://lemon.cs.elte.hu/). The performance of these codes is compared to well-known and efficient minimum-cost flow solvers, namely CS2, RelaxIV, MCF, and the corresponding method of the LEDA library. According to thorough experimental analysis, the presented cost-scaling and network simplex implementations turned out to be more efficient than LEDA and MCF. Furthermore, the cost-scaling implementation is competitive with CS2. The RelaxIV algorithm is often much slower than the other codes, although it is quite efficient on particular problem instances.

1 Introduction

Network flow theory comprises a wide range of optimization models, which have countless applications in various fields. One of the most fundamental

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 05C21
Key words and phrases: network, flow, algorithm, minimum-cost flow, efficient implementation

network flow problems is the *minimum-cost flow* (MCF) problem. It seeks a minimum-cost transportation of a specified amount of a commodity from a set of supply nodes to a set of demand nodes in a directed network with capacity constraints and linear cost functions defined on the arcs. This problem directly arises in various real-world applications in the fields of transportation, logistics, telecommunication, network design, resource planning, scheduling, and many other industries. Moreover, it also arises as a subproblem in more complex optimization tasks, such as multicommodity flow problems. For a comprehensive study of the theory, algorithms, and applications of network flows, see the book of Ahuja, Magnanti, and Orlin [3].

The MCF problem and its solution methods have been the object of intensive research for decades and they have enormous literature. Numerous algorithms have been developed and studied both from theoretical and practical aspects (see the books [26, 13, 48, 3, 64, 52]). Efficient implementation and profound experimental analysis of these algorithms are also of high interest to the operations research community (for example, see [10, 41, 8, 57, 33, 11, 28]). Nowadays, several commercial and non-commercial MCF solvers are available under different license terms.

The primary goal of our research is to provide highly efficient and robust open source implementations of different MCF algorithms and to compare their performance in practice. Preliminary work was published in [49]. This paper presents a more detailed discussion of our implementations along with extensive benchmark testing on a wide range of problem instances.

In order to achieve a comprehensive study, the following algorithms were implemented: SCC: a *simple cycle-canceling* algorithm; MMCC: *minimum-mean cycle-canceling* algorithm; CAT: *cancel-and-tighten* algorithm; SSP: *successive shortest path* algorithm; CAS: *capacity-scaling* algorithm; COS: *cost-scaling* algorithm in three different variants; and NS: primal *network simplex* method with five different pivot strategies. All of these methods are generally known and well-studied algorithms, our contribution is their efficient implementation with some new heuristics and practical considerations.

According to the authors knowledge, our implementation of the *cost-scaling* algorithm is the first to apply Goldberg's recent *partial augment-relabel* method, which was originally developed to solve the maximum flow problem efficiently [34]. Its utilization in MCF algorithms was also suggested, but was not investigated by Goldberg. According to our tests, this new idea turned out to be a considerable improvement in the cost-scaling MCF algorithm similarly to Goldberg's results on the maximum flow algorithm.

This paper also presents an empirical evaluation of our implementations and their variants. Numerous benchmark tests were performed on many kinds of large-scale networks containing up to millions of nodes and arcs. These problem instances were created either using well-known random generators, namely NETGEN and GOTO, or based on networks arising in real-life problems. The presented results demonstrate the relative performance of the solution methods and give some guidelines for selecting an MCF algorithm that is suitable for a desired application domain.

Our fastest implementations were also compared to four highly regarded minimum-cost flow solvers: CS2 code of Goldberg and Cherkassky [33, 16], an efficient authoritative implementation of the cost-scaling push-relabel algorithm that has served as a benchmark for a long time; the LEDA library [54], which also implements the cost-scaling algorithm; Löbel's MCF code [57, 58], which implements the network simplex algorithm; and RelaxIV [27], a C++ translation of the authoritative FORTRAN implementation of the relaxation algorithm due to Bertsekas and Tseng [8, 7]. We henceforth refer to the MCF code as MCFZIB in order to differentiate it from the problem itself. The experiments we conducted show that our cost-scaling implementation is more efficient than LEDA and performs similarly to or slightly slower than CS2. Our network simplex code clearly outperforms MCFZIB and it is the fastest implementation for solving relatively small problem instances (up to a few thousands of nodes). For large networks, however, the cost-scaling codes are usually more efficient than the network simplex algorithms. The performance of RelaxIV turned out to be fluctuating: it is one of the fastest implementations for solving certain kinds of problem instances, but it is very slow for other instances. The detailed experimental results can be found in Section 4.

The implementations presented in this paper are available with full source codes as part of the LEMON optimization library [55]. LEMON is an abbreviation of *Library for Efficient Modeling and Optimization in Networks.* It is an open source C++ template library with focus on combinatorial optimization tasks related mainly to graphs and networks. It provides easy-to-use and highly efficient implementations of graph algorithms and related data structures, which help solving complex real-life optimization problems. The LEMON project is maintained by the MTA-ELTE Egerváry Research Group on Combinatorial Optimization (EGRES) [25] at the Department of Operations Research, Eötvös Loránd University, Budapest, Hungary. The library is also a member of the COIN-OR initiative [14], a collection of open source projects related to operations research. LEMON applies a very permissive licensing scheme that makes it favorable for commercial and non-commercial software development as well as for research activities. For more information about LEMON, the readers are referred to the introductory paper [21] and to the web site of the library: `http://lemon.cs.elte.hu/`.

The rest of this paper is organized as follows. Section 2 briefly introduces the MCF problem along with the used notations and theorems. Section 3 describes the implemented algorithms and their variants. Section 4 presents the main experimental results. Finally, the conclusions are drawn in Section 5.

2 The minimum-cost flow problem

2.1 Definitions and notations

The *minimum-cost flow* (MCF) problem is defined as follows. Let $G = (V, A)$ be a weakly connected directed graph consisting of $n = |V|$ nodes and $m = |A|$ arcs. We associate with each arc $(i, j) \in A$ a *capacity* (upper bound) $u_{ij} \geq 0$ and a *cost* c_{ij}, which denotes the cost per unit flow on the arc. Each node $i \in V$ has a signed *supply* value b_i. If $b_i > 0$, then node i is called a *supply node* with a supply of b_i; if $b_i < 0$, then node i is called a *demand node* with a demand of $-b_i$; and if $b_i = 0$, then node i is referred to as a *transshipment node.* We assume that all data are integer and we wish to find an integer-valued flow of minimum total cost satisfying the supply-demand constraints at all nodes and the capacity constraints on all arcs. The solution of the problem is represented by flow values x_{ij} assigned to the arcs. Therefore, the MCF problem can be stated as

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1a}$$

subject to

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b_i \qquad \forall i \in V, \tag{1b}$$

$$0 \leq x_{ij} \leq u_{ij} \qquad \forall (i,j) \in A. \tag{1c}$$

We refer to (1b) as *flow conservation constraints* and (1c) as *capacity constraints.* A solution vector x is called *feasible* if it satisfies all constraints defined in (1b) and (1c), and it is called *optimal* if it also minimizes the total flow cost (1a) over the feasible solutions.

The flow conservation constraints (1b) imply that the sum of the node supply values is required to be zero, that is, $\sum_{i\in V} b_i = 0$, in order to have a feasible solution to the MCF problem:

$$\sum_{i\in V} b_i = \sum_{i\in V}\Big(\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji}\Big) = \sum_{i\in V}\sum_{j:(i,j)\in A} x_{ij} - \sum_{i\in V}\sum_{j:(j,i)\in A} x_{ji} = 0.$$

Without loss of generality, we may further assume that all arc capacities are finite, all arc costs are nonnegative, and the problem has a feasible solution [3].

There are several other problem formulations that are equivalent to the above definition, for instance, the minimum-cost circulation problem, the uncapacitated minimum-cost flow problem, and the transportation problem. However, this definition is quite common in the literature.

Flow algorithms and related theorems usually rely on the concept of *residual networks* [26, 13, 3, 64, 52]. For a given feasible flow x, the corresponding residual network G_x is defined as follows. Let $G_x = (V, A_x)$ be a directed graph that contains *forward* and *backward arcs* on the original node set V. A forward arc $(i, j) \in A_x$ corresponds to each original arc $(i, j) \in A$ for which the *residual capacity* $r_{ij} = u_{ij} - x_{ij}$ is positive. A backward arc $(j, i) \in A_x$ corresponds to each original arc $(i, j) \in A$ for which the *residual capacity* $r_{ji} = x_{ij}$ is positive. The cost of a forward arc (i, j) is defined as c_{ij}, while the cost of a backward arc (j, i) is $-c_{ij}$.

The concept of *pseudoflows* is also important for several flow algorithms. A pseudoflow is a function x defined on the arcs that satisfies only the nonnegativity and capacity constraints (1c) but might violate the flow conservation constraints (1b). A feasible flow is also a pseudoflow. In case of a pseudoflow x, a node might have a certain amount of undelivered supply or unfulfilled demand, which is called the *excess* or *deficit* of the node, respectively. Formally, the signed *excess* value of a node i with respect to a pseudoflow x is defined as

$$e_i = b_i + \sum_{j:(j,i)\in A} x_{ji} - \sum_{j:(i,j)\in A} x_{ij}. \tag{2}$$

If $e_i > 0$, node i is referred to as an *excess node* with an excess of e_i; and if $e_i < 0$, node i is called a *deficit node* with a deficit of $-e_i$. Note that $\sum_{i\in V} e_i = \sum_{i\in V} b_i = 0$, that is, the total excess of the nodes equals to the total deficit. The residual network corresponding to a pseudoflow is defined in the same way as in case of a feasible flow.

The running time of an MCF algorithm is measured as a function of the size of the network and the magnitudes of the input data. Let U henceforth denote the largest node supply or arc capacity:

$$U = \max\{\max\{|b_i| : i \in V\}, \max\{u_{ij} : (i, j) \in A\}\} \tag{3}$$

and let C denote the largest arc cost:

$$C = \max\{c_{ij} : (i, j) \in A\}. \tag{4}$$

An algorithm is referred to as *pseudo-polynomial* if its running time is bounded by a polynomial function in the dimensions of the problem and the magnitudes of the numerical data, namely, n, m, U, and C. These algorithms technically run in exponential time with respect to the size of the input and they are therefore not considered polynomial. A *(weakly) polynomial* algorithm is one that runs in time polynomial in the input size, namely, n, m, $\log U$, and $\log C$. Furthermore, an algorithm is called *strongly polynomial* if its running time depends upon only on the inherent dimensions of the problem, that is, it runs in time polynomial in n and m regardless of the numerical input data.

2.2 Optimality conditions

In the followings, we formulate optimality conditions for the MCF problem in terms of the residual network as well as the original network. These fundamental theorems are useful in several aspects. Not only do they provide simple methods for verifying the optimality of a certain solution, but they also suggest algorithms for solving the problem. These results are discussed in [26, 13, 3, 64, 52].

Theorem 1 (Negative cycle optimality conditions) *A feasible solution x of the MCF problem is optimal if and only if the residual network G_x contains no directed cycle of negative total cost.*

This theorem is a consequence of the observation that any feasible flow can be decomposed into a finite set of augmenting paths and cycles.

We also introduce two equivalent formulations of optimality conditions that rely on the notions of *node potentials* and *reduced costs.* We associate with each node $i \in V$ a signed value π_i, which is referred to as the potential of node i. Actually, π_i can be viewed as the linear programming dual variable corresponding to the flow conservation constraint of node i (see [3]). With respect to a given potential function π, the reduced cost of an arc (i,j) is defined as

$$c^{\pi}_{ij} = c_{ij} + \pi_i - \pi_j. \qquad (5)$$

Note that c^{π}_{ij} measures the relative cost of the arc (i,j) with respect to the potentials of its end-nodes.

This concept allows us to formulate the following optimality conditions.

Theorem 2 (Reduced cost optimality conditions) *A feasible solution x of the MCF problem is optimal if and only if for some node potential func-*

tion π*, the reduced cost of each arc in the residual network* G_x *is nonnegative:*

$$c^{\pi}_{ij} \geq 0 \qquad \forall (i,j) \in A_x. \tag{6}$$

Note that the total reduced cost of a directed cycle with respect to any potential function equals to the original cost of the cycle. Therefore, the conditions of Theorem 2 obviously imply the negative cycle optimality conditions defined in Theorem 1. Furthermore, a constructive proof exists for the converse result. For an optimal flow x, corresponding optimal node potentials π can be obtained by solving a shortest path problem in the residual network.

Theorem 2 can be restated in terms of the original network as follows.

Theorem 3 (Complementary slackness optimality conditions) *A feasible solution* x *of the MCF problem is optimal if and only if for some node potential function* π*, the following complementary slackness conditions hold for each arc* $(i,j) \in A$ *of the original network:*

$$\text{if } c^{\pi}_{ij} > 0, \text{ then } x_{ij} = 0; \tag{7a}$$

$$\text{if } 0 < x_{ij} < u_{ij}, \text{ then } c^{\pi}_{ij} = 0; \tag{7b}$$

$$\text{if } c^{\pi}_{ij} < 0, \text{ then } x_{ij} = u_{ij}. \tag{7c}$$

In addition to these exact optimality conditions, the characterization of *approximate optimality* is also of particular importance. Several algorithms rely on the concept of ϵ-optimality. For a given $\epsilon \geq 0$, a feasible flow or a pseudoflow x is called ϵ*-optimal* if for some node potential function π, the reduced cost of each arc in the residual network G_x is at least $-\epsilon$, that is,

$$c^{\pi}_{ij} \geq -\epsilon \qquad \forall (i,j) \in A_x. \tag{8}$$

These conditions are the relaxations of the reduced cost optimality conditions defined in Theorem 2 and are equivalent to them when $\epsilon = 0$. The ϵ-optimality conditions can also be restated in terms of the original network to obtain the relaxations of the complementary slackness optimality conditions defined in Theorem 3.

The following lemma formulates two simple observations that are related to ϵ-optimality.

Lemma 4 *Any feasible solution* x *of the MCF problem is* ϵ*-optimal if* $\epsilon \geq C$*. Moreover, if the arc cost are integer and* $\epsilon < 1/n$*, then an* ϵ*-optimal feasible flow is an optimal solution.*

Note that an ϵ-optimal flow is also ϵ'-optimal for all $\epsilon' > \epsilon$ and hence the approximate optimality of x is best indicated by the smallest value $\epsilon \geq 0$ for which x is ϵ-optimal. This minimum value is referred to as $\epsilon(x)$. The following theorem reveals an inherent connection between $\epsilon(x)$ and the *minimum-mean cycles* of the residual network. The *mean cost* of a directed cycle is defined as its total cost divided by the number of arcs in the cycle.

Theorem 5 *For a non-optimal feasible solution x of the MCF problem, $\epsilon(x)$ equals to the negative of the minimum-mean cost of a directed cycle in the residual network G_x. For an optimal solution x, $\epsilon(x) = 0$.*

Therefore, $\epsilon(x)$ can be computed by finding a directed cycle of minimum-mean cost, which can be carried out in $O(nm)$ time [46]. Another related problem is to find an appropriate potential function π for an ϵ-optimal flow or pseudoflow x so that they satisfy the ϵ-optimality conditions. Similarly to the problem of finding optimal node potentials, this problem can also be solved by performing a shortest path computation in G_x but with a modified cost function c' for which $c'_{ij} = c_{ij} + \epsilon$ for each arc (i, j) in G_x. See, for example, [3, 29] for the proof of all these results related to ϵ-optimality.

2.3 Solution methods

The MCF problem and its solution methods have a rich history spanning more than fifty years. Researchers first studied a classical special case of the MCF problem, the so-called *transportation problem*, in which the network consists only of supply and demand nodes. Dantzig was the first to solve the transportation problem by specializing his famous linear programming method, the simplex algorithm. Later, he also applied this approach to the MCF problem and developed a solution method that is known as the *network simplex* algorithm. These results are discussed in Dantzig's book [18].

Ford and Fulkerson developed the first combinatorial algorithms for the uncapacitated and capacitated transportation problems by generalizing Kuhn's remarkable Hungarian Method [53]. Ford and Fulkerson later proposed a similar *primal–dual* algorithm for the MCF problem, as well. Their results are presented in the book [26].

In the next few years, other algorithmic approaches were also suggested, namely, the *successive shortest path* algorithm, the *out-of-kilter* algorithm, and the *cycle-canceling* algorithm. These methods, however, do not run in polynomial time. Therefore, both theoretical and practical expectations motivated further research on developing more efficient algorithms. Edmonds and

Karp [24] introduced the *scaling technique* and developed the first weakly polynomial-time algorithm for solving the MCF problem. Later, other researchers also recognized the significant value of this approach and proposed various scaling algorithms. The problem of finding a strongly polynomial-time MCF algorithm, however, remained a challenging open question for several years. Tardos [67] developed the first such algorithm, which was followed by many other methods providing improved running time bounds.

Besides theoretical aspects, efficient implementation and computational evaluation of MCF algorithms have also been an object of intensive research. The *network simplex* algorithm became quite popular in practice when efficient spanning tree labeling techniques were developed to improve its performance. Later, other algorithms also turned out to be quite efficient. Implementations of *relaxation* and *cost-scaling* algorithms were reported to be competitive with the fastest network simplex codes.

Detailed discussion and complexity survey of MCF algorithms can be found in, for example, [3, 64, 52]. Table 1 provides a brief summary of the MCF algorithms having best theoretical running time. Recall from Section 2.1 that n and m denote the number of nodes and arcs in the network, respectively; U denotes the maximum of supply values and arc capacities; and C denotes the largest arc cost. Furthermore, let $\mathrm{SP}_+(n, m)$ denote the running time of any algorithm solving the single-source shortest path problem in a directed graph with n nodes, m arcs, and a nonnegative length function. Dijkstra's algorithm with Fibonacci heaps provides an $O(m + n \log n)$ bound for $\mathrm{SP}_+(n, m)$ [64, 15, 52].

$O(nU \cdot \mathrm{SP}_+(n, m))$	Edmonds and Karp [24]; Tomizawa [70] *successive shortest path*
$O(m \log U \cdot \mathrm{SP}_+(n, m))$	Edmonds and Karp [24] *capacity-scaling*
$O(m \log n \cdot \mathrm{SP}_+(n, m))$	Orlin [60] *enhanced capacity-scaling*
$O(nm \log(n^2/m) \log(nC))$	Goldberg and Tarjan [38] *generalized cost-scaling*
$O(nm \log \log U \log(nC))$	Ahuja, Goldberg, Orlin, and Tarjan [1] *double scaling*
$O((m^{3/2}U^{1/2} + mU \log(mU)) \log(nC))$	Gabow and Tarjan [30]
$O((nm + mU \log(mU)) \log(nC))$	Gabow and Tarjan [30]

Table 1: Best theoretical running time bounds for the MCF problem

3 Implemented algorithms

This section discusses the algorithms we implemented as well as the most important heuristics and other practical improvements. All of these methods are well-studied in the literature. Their profound theoretical analysis with the proof of correctness and running time can be found in [3] and in other papers and books cited later in this section. The contribution of this paper is the efficient implementation and empirical analysis of several variants of these algorithms. For further details of our implementations, the readers are referred to the documentation and the source code of the LEMON library [55].

Table 2 provides an overview of the implemented algorithms and their worst-case running time. The same notations are used as in the previous section. Two of these algorithms perform shortest path computations with nonnegative length functions. Our implementations use Dijkstra's algorithm with binary heaps by default, hence $\mathrm{SP}_+(n, m) = O((n+m)\log n)$. Note that some of these algorithms have other variants with better theoretical running time, but our research especially focused on the practical performance of them. The given running time bounds correspond to the actual implementations.

Alg.	Name	Running time
SCC	*simple cycle-canceling*	$O(nm^2CU)$
MMCC	*minimum-mean cycle-canceling*	$O(n^2m^2 \min\{\log(nC), m\log n\})$
CAT	*cancel-and-tighten*	$O(n^2m \min\{\log(nC), m\log n\})$
SSP	*successive shortest path*	$O(nU \cdot \mathrm{SP}_+(n, m))$
CAS	*capacity-scaling*	$O(m \log U \cdot \mathrm{SP}_+(n, m))$
COS	*cost-scaling*	$O(n^2m \log(nC))$
NS	*network simplex*	$O(nm^2CU)$

Table 2: Implemented algorithms and their worst-case running time

3.1 Cycle-canceling algorithms

Cycle-canceling is one of the simplest methods for solving the MCF problem. This algorithm applies a primal approach based on Theorem 1. A feasible flow x is first established, which can be carried out by solving a maximum flow problem. After that, the algorithm throughout maintains feasibility of the solution x and gradually decreases its total cost. At each iteration, a directed cycle of negative cost is identified in the residual network G_x and this cycle is canceled by pushing the maximum possible amount of flow along it. When the residual network contains no negative-cost directed cycle, the algorithm terminates and Theorem 1 implies that the solution is optimal.

The cycle-canceling algorithm was proposed by Klein [50]. Its generic version does not specify the order of selecting negative cycles to be canceled, but it runs in pseudo-polynomial time for the MCF problem with integer data. Since the total flow cost is decreased at each iteration and mCU is clearly an upper bound of the flow cost, the algorithm performs $O(mCU)$ iterations if all data are integer. Klein used a label-correcting shortest path algorithm that identifies a negative cycle in $O(nm)$ time, thus his algorithm runs in $O(nm^2CU)$ time. Later, numerous other variants of the cycle-canceling method were also developed by applying different rules for cycle selection, for example [4, 37, 66]. These algorithms have quite different theoretical and practical behavior. Some of them run in polynomial or even strongly polynomial time.

We implemented three cycle-canceling algorithms, which are discussed in the followings.

Simple cycle-canceling algorithm. This is a simple version of the cycle-canceling method using the Bellman–Ford algorithm for identifying negative cycles. We henceforth denote this implementation as SCC.

It is well-known that the Bellman–Ford algorithm is capable of detecting a negative-cost directed cycle after performing n iterations or detecting that such a cycle does not exist [15]. However, it is not required to perform n iterations in most cases. If negative cycles exist in the graph, one or more of them typically appear in the subgraph identified by the predecessor pointers of the nodes after much less iterations. Unfortunately, we do not know the sufficient limit for the number of iterations in advance and searching for cycles using the predecessor pointers at an intermediate step of the algorithm is a relatively slow operation. Therefore, our SCC implementation performs such checking after a successively increasing number of iterations of the Bellman–Ford algorithm. According to our tests, it turned out to be practical to search for negative cycles after executing $\lfloor 2 \cdot 1.5^k \rfloor$ iterations for each $k \geq 0$ until this limit reaches n. It is also beneficial to cancel all node-disjoint negative cycles that can be found at once when Bellman–Ford algorithm is stopped. The worst case time complexity of the SCC algorithm is $O(nm^2CU)$.

Minimum-mean cycle-canceling algorithm. This famous special case of the cycle-canceling method was developed by Goldberg and Tarjan [37]. It selects a negative cycle of minimum mean cost to be canceled at each iteration, which yields the simplest MCF algorithm running in strongly polynomial time. We denote this method and our implementation as MMCC.

Recall from Section 2.2 that the mean cost of a directed cycle is defined as its total cost divided by the number of arcs in the cycle. The MMCC algorithm iteratively identifies a directed cycle W of minimum-mean cost in the current residual network. If the cost of W is negative, then the cycle is canceled and another iteration is performed, otherwise the algorithm terminates with an optimal solution found. It has been proved that this algorithm performs $O(nm^2 \log n)$ iterations for arbitrary real-valued arc costs and $O(nm \log(nC))$ iterations for integer arc costs. The proof of these bounds relies on the concept of ϵ-optimality and is rather involved, see [37, 3, 52], although the algorithm is very simple to state.

The MMCC algorithm relies on finding minimum-mean directed cycles in a graph. This optimization problem has also been studied for a long time and several efficient algorithms have been developed for solving it [46, 42, 20, 19, 31]. The best strongly polynomial-time bound for a minimum-mean cycle algorithm is $O(nm)$ and thus the overall running time of the MMCC algorithm is $O(n^2m^2 \min\{\log(nC), m \log n\})$ for the MCF problem with integer data.

We implemented three known algorithms for finding minimum-mean cycles: Karp's original algorithm [46]; an improved version of this method that is due to Hartmann and Orlin [42]; and Howard's *policy-iteration* algorithm [43, 19]. The first two methods run in strongly polynomial time $O(nm)$. In contrast, Howard's algorithm is not known to be polynomial, but it is one of the fastest solution methods in practice [20, 19].

Our experiments also verified that Howard's algorithm is orders of magnitude faster than the other two methods we implemented. This algorithm gradually approximates the optimal solution by performing linear-time iterations. Relatively few iterations are typically sufficient to find a minimum-mean cycle, but no polynomial upper bound is known. Therefore, we developed a combined method in order to achieve the best performance in practice while keeping the strongly polynomial upper bound on the running time. Howard's algorithm is run with an explicit limit on the number of iterations. If this limit is reached without finding the optimal solution, we stop Howard's algorithm and execute the Hartmann–Orlin algorithm. We set this iteration limit to n, and hence the overall running time of this combined method is $O(nm)$, which equals to the best strongly polynomial bound. In our experiments, the iteration limit was indeed never reached. Thus, the combined method was practically identical to Howard's algorithm but with a guarantee of worst-case running time $O(nm)$.

Cancel-and-tighten algorithm. This algorithm can be viewed as an improved version of the MMCC algorithm, which is also due to Goldberg and Tarjan [37]. It is faster than MMCC both in theory and practice. This algorithm is henceforth denoted as CAT.

The improvement of the CAT algorithm is based on a more flexible selection of cycles to be canceled. The previously studied cycle-canceling algorithms are pure primal methods in a sense that they do not consider the dual solution at all. In contrast, the CAT algorithm explicitly maintains node potentials, which make the detection of negative residual cycles easier and faster. The key idea of the algorithm is to cancel cycles that consist entirely of negative-cost arcs. Note that the sum of the reduced arc costs along a cycle with respect to any potential function is exactly the same as the original cost of the cycle. Therefore, the algorithm can consider the reduced costs with respect to the current node potentials instead of the original costs. A residual arc is called *admissible* if its reduced cost is negative; the subgraph of the residual network consisting only of the admissible arcs is called the *admissible network*; and a directed cycle in the admissible network is referred to as an *admissible cycle.*

The CAT algorithm performs two main steps at every iteration until the current solution becomes optimal. In the *cancel* step, admissible cycles are successively canceled until such a cycle does not exist. In the *tighten* step, the node potentials are modified in order to make more arcs admissible. Despite the MMCC algorithm, this method explicitly utilizes the concept of ϵ-optimality. Recall the corresponding definitions and theorems from Section 2.2. The CAT algorithm ensures ϵ-optimality of the solution for successively smaller values of $\epsilon \geq 0$. In the tighten step, the potentials are modified so as to satisfy the ϵ-optimality conditions for a smaller ϵ that is at most $(1 - 1/n)$ times its former value.

The cancel step is the dominant part of the computation. We implemented a straightforward method for this step based on a depth-first traversal of the admissible network. This implementation runs in $O(nm)$ time as canceling a cycle takes $O(n)$ time and at most $O(m)$ admissible cycles can be successively canceled without modifying the potential function. Goldberg and Tarjan [37] also showed that using dynamic tree data structures [65], the running time of this step can be reduced to $O(m \log n)$ (amortized time $O(\log n)$ per cycle cancellation). However, we did not invest effort in implementing this variant because the cycle-canceling algorithms turned out to be relatively slow in our experimental tests (see Section 4).

The tighten step can be performed in $O(m)$ time based on a topological ordering of the nodes with respect to the admissible network. This implementation, however, does not ensure that the overall running time of the algorithm is strongly polynomial. To overcome this drawback, Goldberg and Tarjan [37] suggested to carry out the tighten step in a stricter way after every $O(n)$ iterations of the algorithm. In these cases, a minimum-mean cycle computation is performed to exactly determine the smallest ϵ value for which the current flow is ϵ-optimal (see Theorem 5). Node potentials are also recomputed to correspond to this ϵ value. Note that the amortized running time $O(m)$ of the tighten step is not affected by this modification. Our implementation, however, performs this stricter tighten step more often, namely after every $\lfloor\sqrt{n}\rfloor$ iterations, because it turned out to be more efficient in practice. This means that the amortized running time of the tighten step becomes $O(m\sqrt{n})$, but it is still less than the $O(nm)$ time of our implementation of the cancel step. The minimum-mean cycle computations are carried out using the same combined algorithm that was applied in the MMCC algorithm.

This algorithm is strongly polynomial. It runs in $O(n^2m^2\log n)$ time for the MCF problem with arbitrary arc costs and in $O(n^2m\log(nC))$ time for integer arc costs, see [37].

The experimental results for these algorithms are presented in Section 4. It turned out that their relative performance depends upon the problem instance, but the CAT algorithm is usually much more efficient than both SCC and MMCC. However, all three of these cycle-canceling algorithms turned out to be slower than the cost-scaling and network simplex methods.

3.2 Augmenting path algorithms

Another fundamental approach for solving the MCF problem is the so-called *successive shortest path* method. It is a dual ascent algorithm that successively augments flow along shortest paths of the residual network to send the required amount of flow from the supply nodes to the demand nodes. In this sense, this method can be viewed as a generalization of the well-known augmenting path algorithms solving the maximum flow problem, namely the Ford–Fulkerson and Edmonds–Karp algorithms [26, 24, 3, 15].

The successive shortest path algorithm in its inital form was developed independently by Jewell [45], Iri [44], and Busacker and Gowen [12]. They showed that the MCF problem can be solved by a sequence of shortest path computations. Later, Edmonds and Karp [24] and Tomizawa [70] independently suggested the utilization of node potentials in the algorithm to maintain nonnegative arc costs for the shortest path problems. This technique greatly improves both the theoretical and the practical performance of the algorithm. Edmonds and Karp [24] also developed a capacity-scaling variant of this method that runs in polynomial time.

We implemented the standard successive shortest path algorithm applying node potentials as well as the capacity-scaling method. These algorithms and the most important aspects of their implementations are discussed below.

Successive shortest path algorithm. This algorithm is henceforth denoted as SSP. In contrast to the cycle-canceling method, which maintains a feasible flow and attempts to achieve optimality, the SSP algorithm maintains an optimal pseudoflow and node potentials and attempts to achieve feasibility. Recall from Section 2.1 that a pseudoflow satisfies the nonnegativity and capacity constraints but might violate the flow conservation constraints at some nodes. Such a node has a certain amount of excess or deficit.

The SSP algorithm begins with constant zero pseudoflow x and a constant potential function π and proceeds by gradually converting x into a feasible solution while throughout maintaining the reduced cost optimality conditions defined in Theorem 2. At every iteration, the algorithm selects a node with positive excess and sends flow from this node to an arbitrary deficit node along a shortest path of the residual network with respect to the reduced arc costs. After that, the node potentials are modified using the computed shortest path distances to preserve the reduced cost optimality conditions. These conditions not only verify the optimality of both the primal and the dual solutions, but they also ensure nonnegative arc costs for the consecutive shortest path computations. By sending flow from excess nodes to deficit nodes, the algorithm iteratively decreases the total excess of the nodes until the solution becomes feasible. At the beginning of the algorithm, the total excess is at most $nU/2$ and each iteration decreases this value by at least one (in case of integer data), thus the SSP algorithm terminates after $O(nU)$ path augmentations. The flow conservation constraints are then satisfied at all nodes and hence the solution is both feasible and optimal.

We implemented the SSP algorithm as follows. At each iteration, flow is augmented from the current excess node v to a deficit node w whose shortest path distance from v is minimal. The shortest path searches are carried out using Dijkstra's algorithm with a heap data structure. We experimented with several heap variants provided by the LEMON library [56] and the standard binary heap structure turned out to be one of the fastest and most robust implementations. Therefore, our SSP implementation uses this data structure by default. This means that a single iteration is performed in $O((n+m)\log n)$ time and the overall complexity of the algorithm is $O(nU(n+m)\log n)$. However, one can easily switch to other data structures (for instance, Fibonacci heaps).

The practical performance of the SSP algorithm mainly depends on the shortest path computations. We applied a significant improvement related to these searches, which is discussed, for example, in [3]. At each iteration, it is not necessary to compute the shortest paths to all nodes from the current excess node v, but the Dijkstra algorithm can be terminated once it permanently labels a deficit node w. The node potentials can also be updated in an alternative way that does not require any modification for those nodes that were not permanently labeled during the shortest path computation. This improvement can be implemented quite easily, but it greatly improves the efficiency of the SSP algorithm in practice.

The representation of the residual network is another important aspect of the implementation. It is possible to implement the SSP algorithm using the original representation of the input network, but in this case, all outgoing and incoming arcs of the current node have to be checked at each step of the shortest path computations. Another possibility is to explicitly maintain the residual network containing only those arcs that have positive residual capacity. However, this implementation would require the updating of the graph structure after each path augmentations, which is time-consuming.

We applied an intermediate solution that turned out to be the most efficient. We store an auxiliary graph G' that contains all possible forward and backward arcs and also maintain their residual capacities explicitly. All shortest path computations run on G' by skipping those arcs whose current residual capacity is zero. The flow augmentations are carried out by decreasing the residual capacities of the arcs on the path and increasing the residual capacities of the corresponding reverse arcs. Therefore, we also store for each arc an index to its *reverse arc* (often referred to as *sister arc*). The major benefit of this implementation is that this auxiliary graph G' allows a quite efficient representation. Note that using G', only the outgoing arcs of a node have to be traversed during the shortest path searches and G' is not modified throughout the algorithm. We can, therefore, represent the outgoing arcs of a node in G' by consecutive integers, which makes it possible to traverse these arcs quite efficiently without iterating over the elements of an array or a linked list.

The reduced arc costs are also required in the shortest path computations. Since these values are frequently modified by adjusting node potentials, it is better to store only the potentials and recompute reduced costs whenever they are needed. Furthermore, we also maintain a signed excess value for each node.

Capacity-scaling algorithm. This algorithm, which we denote as CAS, is an improved version of the SSP method. It uses a capacity-scaling scheme that reduces the number of iterations from $O(nU)$ to $O(m \log U)$. This algorithm was devised by Edmonds and Karp [24] as the first weakly polynomial-time solution method for the MCF problem. Our implementation is based on a slightly modified variant that is due to Orlin [60] and also discussed in [3].

The SSP algorithm has a substantial drawback that the shortest path augmentations might deliver relatively small amounts of flow, which results in a large number of iterations. This is overcome in the CAS algorithm by ensuring that each path augmentation carries a sufficiently large amount of flow and hence the number of augmentations is often reduced. The CAS algorithm performs scaling phases for successively smaller values of a parameter Δ. In a Δ-scaling phase, each path augmentation delivers exactly Δ units of flow from a node v with at least Δ units of excess to a node w with at least Δ units of deficit. The shortest path searches are carried out in the so-called *Δ-residual network*, which contains only those arcs whose residual capacities are at least Δ. When no such augmenting path is found, the value of Δ is halved and the algorithm proceeds with the next phase. Initially, Δ is set to $2^{\lfloor \log_2 U \rfloor}$ and the algorithm terminates at the end of the phase in which $\Delta = 1$.

The CAS algorithm maintains the reduced cost optimality conditions only in the Δ-residual network. Each Δ-scaling phase begins with saturating those newly introduced arcs of the current Δ-residual network that do not satisfy the optimality conditions with respect to the current node potentials. The saturations might increase the excess or deficit of some nodes, but these requirements are satisfied in the subsequent phases. At the end of the last phase, which corresponds to $\Delta = 1$, the solution becomes both feasible and optimal since the Δ-residual network then coincides with the residual network.

In order to ensure the weakly polynomial running time of the CAS algorithm, we need an additional assumption that a directed path of sufficiently large capacity exists between each pair of nodes. This condition, however, can easily be achieved by a simple extension of the underlying network as follows. Let s denote a designated node of the network (or a newly introduced artificial node). For each other node i, we can add new arcs (i, s) and (s, i) to the graph with sufficiently large capacities and costs. Under this additional assumption, the CAS algorithm is proved to solve the MCF problem in $O(m \log U \cdot \mathrm{SP}_+(n, m))$ time [3].

We made some modifications to this version of the CAS algorithm in our implementation. First, it is possible to avoid the above extension of the input graph by allowing that more units of excess or deficit remain at the end of a Δ-scaling phase. In this case, the polynomial running time bound is not proved, but our experiments show that this version does not perform more path augmentations and runs significantly faster in practice. Therefore, our implementation does not extend the input graph by default. Another modification utilizes that the path augmentations of each Δ-scaling phase might be capable of delivering more than Δ units of flow. We send the maximum possible amount of flow along each path similarly to the SSP algorithm. Furthermore, the scaling of the parameter Δ can be carried out using a factor other than two. Let $\alpha \geq 2$ denote an integer scaling factor. Δ is initially set to $\alpha^{\lfloor \log_\alpha U \rfloor}$ and divided by α at the end of each phase. This means that more path augmentations might be required for the same excess or deficit node in a Δ-scaling phase, but the number of phases is reduced. In our experiments, a factor of $\alpha = 4$ turned out to provide the best overall performance, thus this option is used by default.

The CAS algorithm has much in common with the SSP method, thus the practical improvements of the SSP implementation also applies to this algorithm. Our CAS code uses the same representations for the residual network and the associated data. In a Δ-scaling phase, the Δ-residual network is not constructed explicitly, but the arcs with residual capacity less than Δ are skipped during the path searches. Moreover, our CAS implementation also terminates the shortest path computations once an appropriate deficit node is permanently labeled and updates the node potentials accordingly. This idea and the practical data representations substantially improve the performance of the CAS algorithm similarly to the SSP method.

The computational results presented in Section 4 show that the augmenting path algorithms, SSP and CAS, are not robust as their performance greatly depends upon the characteristics of the input. On general problem instances, these algorithms are typically slower than the cost-scaling and network simplex methods, but in certain cases, they turned out to be quite efficient. For example, if the total excess is relatively small and hence a few path augmentations are sufficient to solve the problem, the SSP algorithm is usually the fastest method.

3.3 Cost-scaling algorithm

The cost-scaling technique for the MCF problem was proposed independently by Röck [63] and Bland and Jensen [9]. Goldberg and Tarjan [38] developed an improved method based on these algorithms by also utilizing the concept of ϵ-optimality, which is due to Bertsekas [6] and, independently, Tardos [67].

The cost-scaling algorithm of Goldberg and Tarjan, which we henceforth refer to as COS, can be viewed as a generalization of their well-known push-relabel algorithm for the maximum flow problem [36]. The COS algorithm is one of the most efficient solution methods for the MCF problem, both in theory and practice.

The COS algorithm is a primal–dual method that applies a successive approximation scheme by scaling upon the costs. It iteratively produces ϵ-optimal primal–dual solution pairs for successively smaller values of $\epsilon \geq 0$. (Recall the definitions and results related to ϵ-optimality from Section 2.2.) Initially, $\epsilon = C$ and each phase preforms a *refine* procedure to transform an ϵ-optimal solution into an $(\epsilon/2)$-optimal solution until $\epsilon < 1/n$. At this stage, the algorithm terminates and Lemma 4 implies that an optimal flow is found.

The refine procedure takes an ϵ-optimal primal–dual solution pair (x, π) as input and improves the approximation as follows. First, it saturates each residual arc whose current reduced cost is negative and thereby produces a pseudoflow x that is 0-optimal. This means that x is also ϵ-optimal for any choice of ϵ, but it is not necessarily feasible. After this step, the current approximation parameter ϵ is halved and the pseudoflow x is gradually transformed into a feasible solution again, but in a way that preserves ϵ-optimality for the new value of ϵ. This is achieved by performing a sequence of *push* and *relabel* operations similarly to the push-relabel algorithm for the maximum flow problem.

Let r_{ij} denote the residual capacity of an arc (i, j) in the residual network G_x corresponding to the current pseudoflow x and let e_i denote the signed excess value of node i. We call a node *active* if its current excess is positive. Furthermore, a residual arc (i, j) is called *admissible* if its current reduced cost is negative and the subgraph of the residual network consisting only of the admissible arcs is called the *admissible network*. The refine procedure throughout maintains ϵ-optimality and hence $-\epsilon \leq c^{\pi}_{ij} < 0$ holds for each admissible arc (i, j). A basic operation selects an active node i (i.e., $e_i > 0$) and either *pushes* flow on an admissible residual arc (i, j) or if no such arc exists, updates the potential of node i, which is called *relabeling*.

A push operation on an admissible residual arc (i, j) is carried out by sending $\delta = \min\{e_i, r_{ij}\}$ units of flow from node i to node j and thereby decreasing e_i and increasing e_j by δ. This operation introduces the reverse arc (j, i) into the residual network unless it already had positive residual capacity, but this arc is not admissible since $c^{\pi}_{ji} = -c^{\pi}_{ij} > 0$. If an active node i has no admissible outgoing arc, a relabel operation decreases its potential by ϵ. This means that the reduced cost of each outgoing residual arc of node i is also decreased and the reduced cost of each incoming residual arc is increased by ϵ. Note that this modification preserves the ϵ-optimality conditions while creating new admissible outgoing arcs at node i and thus allowing subsequent push operations to carry the excess of node i. Consequently, the only operation that can introduce a new admissible arc (i, j) is the relabeling of node i. The refine procedure terminates when no active node remains in the network and hence an ϵ-optimal feasible solution is obtained.

It is proved that this generic version of the refine procedure performs $O(n^2)$ relabel operations and $O(n^2m)$ push operations and hence runs in $O(n^2m)$ time [38, 3]. Furthermore, the number of ϵ-scaling phases is $O(\log(nC))$, as ϵ is initially set to C and it is halved at each phase until it decreases below $1/n$. Consequently, the generic COS algorithm runs in weakly polynomial time $O(n^2m\log(nC))$. Note, however, that the order in which the basic operations are performed is not specified. Goldberg and Tarjan [38] showed that applying particular selection rules and using complex data structures yield better theoretical running time. They also developed a generalized framework to obtain a strongly polynomial bound on the number of ϵ-scaling phases by utilizing the same idea that is exploited in the MMCC and CAT algorithms (see Section 3.1). The best variant they devised runs in $O(nm\log(n^2/m)\min\{\log(nC), m\log n\})$ time using dynamic trees [38, 65]. Moreover, the COS algorithm turned out to be quite efficient in practice and several complicated heuristics were also developed to improve its performance [33].

We implemented three variants of the COS method that perform the refine procedure rather differently.

Push-relabel variant. This variant of the COS algorithm is based on the generic version discussed above and hence performs local push and relabel operations in the ϵ-scaling phases. We also applied several improvements and efficient heuristics in this implementation according to the ideas found in [38, 3, 35, 33, 11]. In fact, most of these improvements and heuristics are analogous to similar techniques devised for the push-relabel maximum flow algorithm.

The bottleneck of the COS algorithm corresponds to the searching of admissible arcs for the basic operations. Therefore, we applied the same graph representation that is used in the augmenting path algorithms (see Section 3.2) as it is intended to minimize the time required for iterating over the outgoing residual arcs of a node.

Similarly to the capacity-scaling algorithm, the COS method also allows us to use an arbitrary scaling factor $\alpha > 1$. We found that the optimal value depends on the problem, but it was usually between 8 and 24 and the differences were moderate. The default scaling factor is $\alpha = 16$ in our implementation, which typically performed very well. Another practical modification targets the issue that the generic COS algorithm performs internal computations with non-integer values of ϵ and non-integer node potentials. This drawback can be overcome by multiplying all arc costs by αn for a given integer scaling factor $\alpha \geq 2$ and by scaling ϵ accordingly. Initially, ϵ is set to $\alpha^{\lceil \log_\alpha(\alpha n C)\rceil}$ and is divided by α in each phase until $\epsilon = 1$.

We applied some improvements in the implementation of the refine procedure, as well. For performing the basic operations, we need to check the outgoing residual arcs of each active node for admissibility. These examinations can be made more efficiently if we record a *current arc* for each active node and continue the search for an admissible outgoing arc from this current arc every time. If an admissible arc is found, we perform a push operation and when we reach the last outgoing arc of an active node without finding an admissible arc, the node is relabeled and its current arc is set to the first outgoing residual arc again. (Recall that the definition of the basic operations imply that only the relabeling of node i can introduce a new admissible arc outgoing from node i.) Furthermore, the relabel operations are performed in a stricter way. Instead of simply decreasing the potential of a relabeled node by ϵ, we decrease the potential by the largest possible amount that does not violate the ϵ-optimality conditions. A single relabel operation thereby usually introduce more admissible arcs. This modification significantly improves the overall performance of the algorithm (up to a factor of two).

The strategy for selecting an active node for the next basic operation is also important. The number of active nodes is typically small, thus it is beneficial to keep track of them explicitly. A particular variant of the COS algorithm, known as the *wave implementation*, selects the active nodes according to a topological ordering with respect to the admissible network. This choice is proved to yield an $O(n^3)$-time implementation of the refine procedure (instead of $O(n^2m)$). However, our experiments showed that a simple FIFO selection rule using a queue data structure usually results in less basic operations and better performance in practice, which is in accordance with [11] and [33].

In addition to the implementation aspects discussed so far, some effective heuristics can improve the practical performance of the COS algorithm to a higher extent. We implemented three such improvements out of the four proposed by Goldberg [33]. These heuristics are also discussed in [35] along with detailed experimental evaluation. Their practical effect depends on the problem instances as well as the actual implementation and the parameter settings (for example, the scaling factor α).

The *potential refinement* (or *price refinement*) heuristic is based on the observation that an ϵ-scaling phase may produce a solution that is not only ϵ-optimal, but also (ϵ/α)-optimal or even optimal. Therefore, an additional step is introduced at the beginning of each phase to check if the current solution is already ϵ-optimal. This heuristic attempts to adjust the potentials to satisfy the ϵ-optimality conditions, but without modifying the flow. If ϵ-optimality is verified, the refine procedure is skipped and another phase is performed. We implemented this potential refinement heuristic using an $O(nm)$-time scaling shortest path algorithm [32] as suggested in [33]. Our experiments also verified that this improvement usually eliminates the need for the refine procedure in a few phases, especially the last ones. Furthermore, the potential updates performed in this heuristic step typically reduce the number of basic operations even in case the refine procedure can not be skipped. Consequently, this additional step significantly improves the overall performance of the algorithm in most cases.

Another possible implementation of this heuristic performs a minimum-mean cycle computation in each phase to determine the smallest ϵ for which the current flow is ϵ-optimal and computes corresponding node potentials. This computation may allow us to skip more than one phase at once, but it is usually slower, even using Howard's efficient algorithm. Furthermore, this variant can be used to ensure a strongly polynomial bound on the number of phases and thus on the overall running time, as well. However, our experiments showed that the former variant of the potential refinement heuristic, which we use in our final implementation, is clearly superior to this one. This result contradicts the conclusions of [11].

The *global update* heuristic performs relabel operations on several nodes in one step. It iteratively applies the following *set-relabel* operation. Let $S \subset V$ denote a set of nodes such that it contains all deficit nodes, but at least one active node is in $V \setminus S$. If no admissible arc enters S, then the potential of every node in S can be increased by ϵ without violating the ϵ-optimality conditions. Furthermore, it is also shown in [33] that the theoretical running time of the COS algorithm remains unchanged if the global update heuristic is applied only after every $\Omega(n)$ relabel operations. In practice, this modification turned out to impose a huge improvement in the efficiency of the algorithm on some problem classes, although it does not help to much or even slightly worsens the performance on other instances. Our implementation of this heuristic follows the instructions presented in [11].

The *push-look-ahead* heuristic is another practical improvement for the COS algorithm. Its goal is to avoid pushing flow from node i to node j when a subsequent push operation is likely to send this amount of flow back to node i. To achieve this, the maximum allowed amount of flow to be pushed into a node i is limited by the sum of its deficit and the residual capacities of its admissible outgoing arcs. However, this idea requires the extension of the relabel operation to those nodes at which this limitation is applied regardless of their current excess values. This heuristic is rather effective in practice, it usually decreases the number of push operations significantly and hence the relabel operations dominate the running time of the COS algorithm. For more details about this heuristic, see [35, 33, 11].

Goldberg [33] also suggests an additional improvement, the *arc fixing* heuristic. The best version of this method speculatively fixes the flow values for the arcs on which it is not likely to be changed later in the algorithm. These arcs are excluded from the subsequent arc examinations, but in certain cases, they have to be unfixed again. We did not implement this heuristic yet, because it seems to be rather involved and sensitive to parameter settings. However, it would most likely improve the performance of our implementation.

We also remark that dynamic trees [65] can be used in the COS algorithm to perform a number of push operations at once, which improves the theoretical running time [38]. However, they are not likely to be practical due to the computational overhead that these data structures usually impose and because applying the above heuristics, the relabel operations become the bottleneck of the algorithm instead of pushes (see [35, 33]). Therefore, we did not implement this variant.

Augment-relabel variant. This variant of the COS algorithm performs path augmentations instead of local push operations, but relabeling is heavily used to find augmenting paths. At each step of the refine procedure, this method selects an active node v and performs a depth-first search in the admissible network to find an augmenting path to a deficit node. At an intermediate stage, the algorithm maintains an admissible path from an active node v to the current node i and attempts to extend this path. If node i has an admissible outgoing arc (i, j), then the path is extended with this arc and node j becomes the current node. Otherwise, node i is relabeled and if $i \neq v$, we step back to the previous node by removing the last arc of the current path. When this search process reaches a deficit node, an augmenting path is found.

The flow augmentation on these admissible paths can be performed in two different ways. The first way is to push the same amount of flow on each arc of an augmenting path, which is bounded by the smallest residual capacity on the path as well as the excess of the starting node v. The other apparent implementation pushes the maximum possible amount of flow on every arc of the path. That is, for each arc (i, j) of the path, $\delta_{ij} = \min\{e_i, r_{ij}\}$ units of flow is pushed on the arc, e_i is decreased by δ_{ij}, and e_j is increased by δ_{ij}. According to our experiments, this variant is slightly superior to the former one, thus it is applied in our implementation.

Note that these path search and flow augmentation methods correspond to a particular sequence of local push and relabel operations. However, the actual push operations are carried out in a delayed and more guided manner, in aware of an admissible path to a deficit node. This helps to avoid such problems for which the push-look-ahead heuristic is devised (see above), but a lot of work may be required to find augmenting paths, especially if they are long.

Since this algorithm can be viewed as a special version of the generic COS method, the same theoretical running time bound applies to it as well as most of the practical improvements. We used the same data representation, improvements and heuristics as for the push-relabel algorithm except for the push-look-ahead heuristic, which is obviously incompatible with this variant. These modifications provided similar performance gains to those measured for the push-relabel variant.

Partial augment-relabel variant. The third variant of the COS algorithm can be viewed as an intermediate approach between the other two variants. It is based on the partial augment-relabel technique recently proposed by Goldberg [34] as an improvement for the push-relabel maximum flow algorithm. This method turned out to be more efficient and more robust than the classical push-relabel algorithm and Goldberg also suggested the utilization of the same idea in the MCF context. According to the authors knowledge, our implementation of the COS algorithm is the first to incorporate this technique.

The partial augment-relabel algorithm is quite similar to the augment-relabel variant, but it limits the length of the augmenting paths. The path search process is stopped either if a deficit node is reached or if the length of the path reaches a given parameter $k \geq 1$. In fact, the push-relabel and augment-relabel variants are special cases of this approach for $k = 1$ and $k = n$, respectively. Goldberg [34] suggests small values for the parameter k in the maximum flow context, which turned out to apply to the COS algorithm, as well. In our experiments, the optimal value of this parameter was typically between 3 and 8 and the differences were not substantial for such small values of k. Our default implementation uses $k = 4$, just like Goldberg's maximum flow implementation, as it turned out to be quite robust.

Apart from the length limitation for the augmenting paths, this variant is exactly the same as the augment-relabel method. (Actually, they have a common implementation but with different values of the parameter k.) However, the partial augment-relabel technique attains a good compromise between the former two approaches and turned out to be clearly superior to them, thus it is our default implementation of the COS algorithm. Unless stated otherwise, we refer to this implementation as COS in the followings.

Section 4 provides experimental results for the COS algorithm and its variants compared to other methods. The classical push-relabel algorithm and especially the partial augment-relabel variant using such heuristics and improvements are highly efficient and robust in practice. In contrast, the augment-relabel variant is often significantly slower.

3.4 Network simplex algorithm

The primal *network simplex* algorithm, which we henceforth refer to as NS, is one of the most popular solution methods for the MCF problem in practice. It is a specialized version of the well-known linear programming (LP) simplex method that exploits the network structure of the MCF problem and performs the basic operations directly on the graph representation. The LP variables correspond to the arcs of the graph and the LP bases are represented by spanning trees.

The NS algorithm is devised by Dantzig, the inventor of the LP simplex method. He first solved the uncapacitated transportation problem using this approach and later generalized the bounded variable simplex method to directly solve the MCF problem [18]. Although the generic version of the NS algorithm does not run in polynomial time, it turned out to be rather efficient in practice. Therefore, subsequent research focused on efficient implementation of the NS algorithm [10, 5, 48, 41, 57] as well as on developing special variants of both the primal and the dual network simplex methods that run in polynomial time [68, 39, 62, 61, 69]. Detailed discussion of the NS method considering both theoretical and practical aspects can be found in [3] and [47].

The fundamental concept on which the NS algorithm is based is the notion of *spanning tree solutions.* Such a solution is represented by a partitioning of the node set V into three subsets (T, L, U) such that each arc in L has flow fixed at zero (*lower bound*), each arc in U has flow fixed at the capacity of the arc (*upper bound*), and the arcs in T form an (undirected) spanning tree of the network. The flow on these *tree arcs* also satisfy the nonnegativity and capacity constraints, but they are not restricted to any of the bounds. It can easily be seen that the flow values on the tree arcs are uniquely determined by the partitioning $(\mathsf{T},\mathsf{L},\mathsf{U})$ since there is no cycle in T. Furthermore, it is proved that if an instance of the MCF problem has an optimal solution, then it also has an optimal spanning tree solution, which can be found by successively transforming a spanning tree solution to another (see [3]). Actually, these spanning tree solutions correspond to the LP basic feasible solutions of the problem. This observation allows us to implement the simplex method by performing all operations directly on the network, without maintaining the simplex tableau, which makes this approach very efficient.

The standard simplex method maintains a basic feasible solution and gradually improves its objective function value by small transformations, known as *pivots*. Accordingly, the NS algorithm throughout maintains a spanning tree solution of the MCF problem and successively decreases the total cost of the flow until it becomes optimal. Furthermore, node potentials are also maintained such that the reduced cost of each arc in the spanning tree equals to zero. At each step, a non-tree arc violating its complementary slackness optimality condition (see Theorem 3) is added to the current spanning tree, which uniquely determines a negative cost residual cycle. This cycle is then canceled by augmenting the maximum possible amount of flow on it and a tree arc corresponding to a saturated residual arc is selected to be removed from the tree. The node potentials are also adjusted to preserve the property that the reduced costs of each tree arc is zero and finally, the tree structure is updated. This whole operation transforming a spanning tree solution to another is called *pivot*. If no suitable entering arc can be found, the current flow is optimal and the algorithm terminates.

In fact, the NS algorithm can also be viewed as a particular variant of the cycle-canceling method (see Section 3.1). Due to the sophisticated method of maintaining spanning tree solutions, however, a negative cycle can be found and canceled much faster (in linear time). On the other hand, an additional technical issue, known as *degeneracy*, may arise in the NS algorithm. If the spanning tree contains an arc whose flow value equals to zero or the capacity of the arc, then a pivot step may detect a cycle of zero residual capacity. Such *degenerate* pivots only modify the spanning tree, but the flow itself remains unchanged. Consequently, it is possible that several consecutive pivots do not actually decrease the flow cost (known as *stalling*) or, which is even worse, the same spanning tree solution occurs multiple times and hence the algorithm does not necessarily terminate in a finite number of iterations (known as *cycling*). Experiments with certain classes of large-scale MCF problems showed that more than 90% of the pivots may be degenerate.

A simple and popular technique to overcome such difficulties is based on the concept of *strongly feasible spanning tree solutions.* A spanning tree solution is called strongly feasible if a positive amount of flow can be sent from each node to a designated root node of the spanning tree along the tree path without violating the nonnegativity and capacity constraints. Using an appropriate rule for selecting the leaving arcs, the NS algorithm can throughout maintain a strongly feasible spanning tree. This technique is proved to ensure that the algorithm terminates in a finite number of iterations [3]. Furthermore, it substantially decreases the number of degenerate pivots in practice and hence makes the algorithm faster.

It can be shown, using a perturbation technique, that the NS algorithm maintaining a strongly feasible spanning tree solution performs $O(nmCU)$ pivots for the MCF problem with integer data regardless of the selection rule of entering arcs [2]. An entering arc can be found in $O(m)$ time and using an appropriate labeling technique, the spanning tree structure can be updated in $O(n)$ time. Therefore, a single pivot takes $O(m)$ time and the total running time of the NS algorithm is $O(nm^2CU)$. However, this bound does not reflect to the typical performance of the algorithm in practice.

The implementation of the primal NS algorithm is based on a practical storage scheme of the spanning tree solutions that makes it possible to perform the basic operations of the algorithm efficiently. The book of Kennington and Helgason [48] discusses several such spanning tree data structures along with methods for updating them during the iterations of the algorithm. A quite popular approach, sometimes referred to as the ATI (*Augmented Threaded Index*) method, represents a spanning tree as follows. The tree has a designated root node and three indices is stored for each node in the tree: the *depth* of the node (i.e., the distance from the root node in the tree), the *parent* of the node in the tree, and a *thread* index which is used to define a depth-first traversal of the spanning tree. This storage scheme and its update mechanism are discussed in detail in [47] and [3]. The ATI technique has an improved version, which is due to Barr, Glover, and Klingman [5] and is usually referred to as the XTI (*eXtended Threaded Index*) method. The XTI scheme replaces the *depth* index by two indices for each node: the *number of successors* of the node in the tree and the *last successor* of the node according to the traversal defined by the *thread* index. Other approaches, for example the so-called API and XPI methods, are also often applied.

We implemented the primal NS algorithm using both ATI and XTI techniques. The latter one has two advantages over the ATI method. First, the XTI indices can be updated more efficiently since a tree alteration of a single pivot usually modifies the depth of several nodes in subtrees that are moved from a position to another, while the set of successors is typically modified only for a much smaller number of nodes. In addition, the XTI method also allow an improved updating process for the node potentials. Note that by removing the leaving arc, the current spanning tree is divided into two subtrees, which are then connected again by the entering arc. In order to preserve zero reduced costs for the tree arcs, we have to increase the potential of each node in one of the subtrees by a certain constant value λ or decrease the potential of each node in the other subtree by λ. The XTI scheme makes it possible to immediately determine which subtree is smaller and to perform the update process on the smaller subtree.

Although the XTI labeling method is not as widely known and popular as the simpler ATI method, our experiments showed that it is much more efficient than ATI on all problem instances. Therefore, the final version of our code only implements the XTI technique. The substantial performance gain of this approach is due to the first advantage mentioned above. In contrast, we found that the alternative potential update is not so important, because the subtree containing the root node turned out to be the bigger one in virtually all pivots. Moreover, we can easily avoid overflow problems related to node potentials and reduced costs if the potential of the root node is not modified throughout the algorithm. Therefore, we decided to update potentials in the subtree not containing the root node in every step. We also applied an important improvement in the implementation of the XTI method. An additional *reverse thread* index is also stored for each node and hence the depth-first traversal is represented by a doubly-linked list. This modification turned out to substantially improve the performance of the update process. In fact, the inventors of the XTI technique also discussed this improvement [5], but they did not applied it to reduce the memory requirements of the representation. (However, note that enormous progress has been made on the computers since the time when that paper was written.)

Another interesting aspect of the data representation for the NS algorithm is that we need not traverse the incident arcs of nodes throughout the algorithm, although such examinations are crucial in other algorithms. Therefore, we applied a quite simple and unusual graph representation to implement the NS algorithm. The nodes and arcs are represented by consecutive integers and we store the source and target nodes for each arc (in arrays), but we do not keep track of the incident arcs of a node at all.

The NS algorithm also requires an initial spanning tree solution to start with. It is possible to transform any feasible solution x to a spanning tree solution x' such that the total cost of x' is less than or equal to the total cost of x. Furthermore, the required spanning tree indices can also be computed by a depth-first traversal of the tree arcs. However, artificial initialization techniques are much more common in practice. An artificial root node s with zero supply is typically added to the network as well as artificial arcs with sufficiently large capacities and costs. Recall that the signed supply value of node i is denoted as b_i. For each original node i, we add a new arc (i, s) to the network if $b_i \geq 0$ and add a new arc (s, i) otherwise. We can set the capacity of each new arc to nU and its cost to nC. In this extended network, we can easily construct a strongly feasible spanning tree solution x as follows. For each original arc (i, j), let $x_{ij} = 0$ and for each original node i, let $x_{is} = b_i$ if $b_i \geq 0$ and let $x_{si} = -b_i$ otherwise. The initialization of the tree indices and the node potentials is straightforward in this case. Furthermore, note that an optimal solution in the extended network does not send flow on artificial arcs due to their large costs unless the original problem is infeasible.

We experimented with both ways of initialization and it turned out that the artificial method usually provides better overall performance mainly because of two reasons. First, the artificial spanning tree solution can be constructed easily and quickly. Second, it allows efficient tree update for the first few pivots as the depth of the tree is rather small. Therefore, we decided to use only this variant in our final implementation. The strongly feasibility is preserved throughout the algorithm by carefully selecting the leaving arc whenever multiple residual arcs are saturated by a pivot step (see [47, 3] for details).

One of the most crucial aspects of the NS algorithm, which is not considered so far, is the selection the entering arcs. Recall that the reduced cost of each tree arc is zero and each non-tree arc has a flow value fixed either at zero or the capacity of the arc. Therefore, a non-tree arc (i, j) allows flow augmentation only in one direction. If the reduced cost of the residual arc associated with this direction is negative, the arc (i, j) can be selected to enter the tree. In this case, a negative-cost residual cycle is formed by this arc and the unique tree path connecting nodes i and j (these tree arcs have zero reduced costs). To implement the NS algorithm, we require a method for selecting such an entering arc at each iteration, which is usually referred to as *pivot rule* or *pricing strategy*. The applied method affects the "goodness" of the entering arcs and thereby the number of iterations as well as the average time required for selecting an entering arc, which is a dominant part of each iteration. Consequently, applying different strategies, we can obtain several variants of the NS algorithm with quite different theoretical and empirical behavior.

We implemented five pivot rules, which are discussed in the followings. Four of them are widely known and well-studied rules [47, 3], while the fifth one is an improved version of the *candidate list* rule. In the discussion of these methods, a non-tree arc is called *eligible* if it does not satisfy the complementary slackness optimality condition and hence can be selected as an entering arc. Let π denote the current set of node potentials and let c^{π}_{ij} denote the reduced cost of an arc (i, j). An eligible arc (i, j) either has zero flow and $c^{\pi}_{ij} < 0$ or has a flow equal to its capacity and $c^{\pi}_{ij} > 0$. We refer to $|\,c^{\pi}_{ij}\,|$ as the *violation* of an eligible arc (i, j).

Best eligible arc pivot rule. This is one of the simplest and earliest pivot strategies, which was proposed by Dantzig and is also known as *Dantzig's pivot rule*. At each iteration, this method selects an eligible arc with the maximum violation to enter the tree. This means that a residual cycle having the most negative total cost is selected to be canceled, which causes the maximum decrease of the objective function per unit flow augmentation. Computational studies showed that this selection rule usually results in fewer iterations than other strategies. However, it has to consider all non-tree arcs and recompute their reduced costs to select the best eligible arc at each iteration. Consequently, the overall performance of the NS algorithm with this pivot rule is rather poor despite the small number of iterations.

First eligible arc pivot rule. Another straightforward idea is to select the first eligible arc at each iteration. The practical implementation of this rule examines the arcs cyclically by starting each search process at the position where the previous eligible arc is found. If we reach the end of the arc list, the examination is continued from the beginning of the list again. If a pivot operation examines all non-tree arcs without finding an eligible arc, the solution is optimal and the algorithm terminates. This strategy represents the other extreme way of selecting the entering arcs compared to the previous rule. It rapidly finds an entering arc at each iteration, but these arcs typically have relatively small violation and hence a lot of iterations are usually required.

Block search pivot rule. Since the previous two rules do not perform well in practice, several other strategies have been devised to implement effective compromise between them. A simple *block search* approach is proposed by Grigoriadis [41]. This method cyclically examines blocks of arcs and selects the best eligible candidate among these arcs at each iteration. The search process starts from the position of the previous entering arc and checks a specified number of arcs by recomputing their reduced costs. If this block contains eligible arcs, then the one with the maximum violation is selected to enter the basis. Otherwise, we examine one or more subsequent blocks of arcs until an eligible arc is found.

The block size B is an important parameter of this method. In fact, the previous two rules are special cases of this one with $B = m$ and $B = 1$. Several sources suggest to set B proportionally to the number of arcs, for example, between 1% and 10% [41, 47]. However, our experiments clearly showed that much better overall performance can be achieved on virtually all problem classes if we set $B = \alpha\sqrt{m}$ for small values of α (for example, between 0.5 and 2). In our implementation, $B = \sqrt{m}$ is used, which results in a highly efficient and robust pivot rule.

Similarly to the *first eligible rule*, this strategy also has the inherent advantage that an arc is allowed to enter the basis only periodically, which usually decreases the number of degenerate pivots in practice [17, 40].

Candidate list pivot rule. This is another classical pivot rule, which was proposed by Mulvey [59]. It occasionally builds a list of eligible arcs and selects the best arcs among these candidates at subsequent iterations. A so-called *major* iteration examines the arcs in a wraparound fashion similarly to the previous rules and builds a list containing at most L eligible arcs. After a major iteration, we perform at most K *minor* iterations, each of which scans this list and selects an eligible arc with maximum violation to enter the basis. If an arc is not eligible any more, it is removed from the list. When K minor iterations are performed or the list becomes empty, another major iteration takes place.

This method is similar to the *block search rule*, but it considers the same subset of the arcs in several consecutive pivots, while the previous rule considers only the best arc of a block and then advances to the next block. We obtained the best average running time for this rule using $L = \sqrt{m}/4$ and $K = L/10$. However, this method usually performed worse than the simpler block search strategy.

Altering candidate list pivot rule. This strategy was developed by us as an improved version of the *candidate list rule*. There are various other rules that exploit similar ideas, but the authors are not aware of another implementation of this method. It maintains a candidate list similarly to the previous rule, but it attempts to extend this list at each iteration and keeps only the several best candidates of the previous iterations. The candidate arcs are collected with the search process used in the *block search rule.*

This method has two parameters: a block size B and the maximum length of the altering candidate list, which is denoted by H. At the beginning of each iteration, we check the current candidate list and remove all arcs that are not eligible any more. After that, at least one arc block of size B is examined to extend the candidate list with new eligible arcs. If a nonempty list is obtained, then an arc of maximum violation is selected from the list to enter that basis. The other arcs are then partially sorted and the list is truncated to contain at most H of the best candidates in terms of their current violation. According to our experiments, this method is very efficient using $B = \sqrt{m}$ and $H = B/100$.

Numerous other rules have also been developed applying similar or more complicated partial pricing techniques. We also implemented several variants, but the *block search pivot rule* and the *altering candidate list pivot rule* turned out to provide the best overall performance. Since the block search rule is simpler and turned out to be slightly more robust, it is our default pivot strategy. Unless stated otherwise, we refer to this variant as NS in the followings.

We also developed an additional heuristic based on the artificial initialization procedure of the algorithm to make the first few pivots faster. The initialization of node potentials implies that an arc (i, j) is eligible for the first pivot if and only if $b_i \geq 0$ and $b_j < 0$. After such an arc enters the basis, new arcs incident to its source node may also become eligible. Therefore, we collect several arcs using a partial traversal of the graph starting from the demand nodes and using the reverse orientation of each arc. This arc list is then used by the first few pivots to select entering arcs from. Our computational results showed that this idea slightly improves the overall performance of the NS algorithm by making these pivots substantially faster.

Section 4 provides experimental results comparing the different pivot rules as well as comparing the NS algorithm to other solution methods. Our NS implementation turned out to be highly efficient, especially on relatively small and medium-sized networks, but it is typically outperformed by the cost-scaling codes on the largest problem instances.

4 Experimental study

This section presents an empirical study of the implemented algorithms and also compares them to other efficient MCF solvers. The contribution of these results is twofold. First, a great number of MCF algorithms are compared using the same benchmark suite, which provides insight into their relative performance on different classes of networks. Second, larger problem instances are also considered than that in previous experimental studies of MCF algorithms, which turned out to be important to draw reliable conclusions related to the asymptotic behavior of these algorithms.

The experiments were conducted on a machine with AMD Opteron Dual Core 2.2 GHz CPU and 16 GB RAM (1 MB cache), running openSUSE 11.4 operating system. All codes were compiled with GCC 4.5.3 using -O3 optimization flag.

4.1 Test instances

Our test suite contains numerous problem instances of variable size and characteristics. Most of these instances were generated with standard random generators, NETGEN and GOTO, while the others are based on either real-life road networks or maximum flow problems arising in computer vision applications. The largest networks contain millions of nodes and arcs.

NETGEN instances. NETGEN [51] is a classical generator that produces random instances of the MCF problem and other network optimization problems. It is generally known to produce relatively easy MCF instances. The source code of NETGEN is available at the FTP site of the First DIMACS Implementation Challenge [22].

Our benchmark suite contains four problem families created with NETGEN.

- **NETGEN-8.** This family contains sparse networks, for which the average outdegree of the nodes is 8 (i.e., $m = 8n$). The arc capacities and costs are selected uniformly at random from the ranges [1..1000] and [1..10000], respectively. The number of supply and demand nodes are both set to about $\sqrt{n}$ and the average supply per supply node is 1000.

- **NETGEN-SR.** This family contains relatively dense networks, for which the average outdegree is about $\sqrt{n}$ (i.e., $m \approx n\sqrt{n}$). The other parameters are set the same way as for the NETGEN-8 family.

- **NETGEN-LO-8.** This family is similar to NETGEN-8 with the only difference that the average supply per supply node is much lower, namely 10 instead of 1000. Therefore, the arc capacities incorporate only “loose” bounds for the feasible solutions.

- **NETGEN-DEG.** In these instances, the number of nodes is fixed to $n = 4096$ and the average outdegree ranges from 2 to $n/2$. Other parameters are the same as of NETGEN-8 and NETGEN-SR instances.

GOTO instances. GOTO is another well-known random generator for the MCF problem, which is intended to produce hard instances. It is developed by Goldberg and is described in [35]. The name of the generator stands for *Grid On Torus*, which reflects to the basic structure of the generated networks. Each GOTO problem instance has one supply node and one demand node and the supply value is adjusted according to the arc capacities. This generator is also available at [22].

We used two GOTO families in our experiments, which differ only in the density of the networks.

- **GOTO-8.** This family consists of sparse networks with an average outdegree of 8. Similarly to the NETGEN families, the maximum arc capacity is set to 1000, while the maximum arc cost is set to 10000.

- **GOTO-SR.** This family consists of relatively dense networks with an average outdegree of about $\sqrt{n}$. Other parameters are the same as of the GOTO-8 family.

ROAD instances. We also experimented with MCF problems that are based on real-world road networks. To generate such instances, we used the TIGER/Line road network files of several states of the USA. These data files are available at the web site of the Ninth DIMACS Implementation Challenge [23].

In our experiments, we selected seven states with road networks of increasing size, namely DC, DE, NH, NV, WI, FL, and TX, and generated MCF problem instances as follows. The original undirected graphs are converted to directed graphs by replacing each edge with two oppositely directed arcs. The cost of an arc is set to the travel time on the corresponding road section and the arc capacities are uniformly set to one. This means that we are actually looking for a specified number of arc-disjoint directed paths from supply nodes to demand nodes having minimum total cost. The number of supply and demand nodes are both $\lfloor \sqrt{n}/10 \rfloor$. These nodes are selected randomly and the supply-demand values are determined by a maximum flow computation that maximizes the total supply with respect to the fixed set of supply and demand nodes.

VISION instances. This family consists of MCF instances based on large-scale maximum flow problems arising in computer vision applications. These maximum flow data files were made available at `http://vision.csd.uwo.ca/data/maxflow/` by the Computer Vision Research Group at the University of Western Ontario. They are intended to be used for benchmarking maximum flow algorithms (for example, see [34]).

We used some of the segmentation instances related to medical image analysis. These instances are defined on three-dimensional grid networks. We selected those variants in which the underlying networks are 6-connected and the maximum arc capacity is 100 (namely, the `bone_sub*_n6c100` files). These maximum flow instances were converted to minimum-cost maximum flow problems using random arc costs selected uniformly from the range [1..100]. The original networks also contain arcs of zero capacity, but we skipped these arcs during the transformation and hence did not preserve the 6-connectivity.

We also experimented with several other problem instances and generator parameters, but this collection turned out to be a representative benchmark suite of reasonable size. For all problem families, we generated three instances of each problem size with different random seeds. In all cases, we report the average running time over such three instances to provide more relevant results.

4.2 Comparison of the implemented algorithms

This subsection presents benchmark results for the implemented algorithms and their variants. Each table reports running time results in seconds. The size of a problem instance is indicated by the number of nodes n and the average outdegree deg (i.e., $m = deg \cdot n$). The best running time is highlighted for each problem size. The codes were executed with an explicit timeout limit of one hour and a "–" sign denotes the cases when this timeout limit was reached. Some charts are also presented showing running time as a function of the number of nodes in the network (logarithmic scale is used for both axes).

Figure 1 and Tables 3 and 4 present the running time results in seconds for NETGEN-8 and NETGEN-SR families. On these instances, the SCC algorithm was about 3 times faster than MMCC, while the CAT algorithm greatly outperformed both of them. It is quite interesting that the simple SSP algorithm was an order of magnitude faster than its capacity-scaling variant, CAS. The CAT algorithm performed similarly to SSP on NETGEN-8 instances, but was significantly slower on NETGEN-SR networks. The COS and NS algorithms were the most efficient on these instances. They turned out to be orders of magnitude faster than the other algorithms. COS showed better asymptotic behavior than NS (see Figure 1) and was significantly faster on the largest NETGEN-8 instances. On the other hand, NS was more efficient on the relatively small sparse networks and on all NETGEN-SR instances despite its worse asymptotic trends.

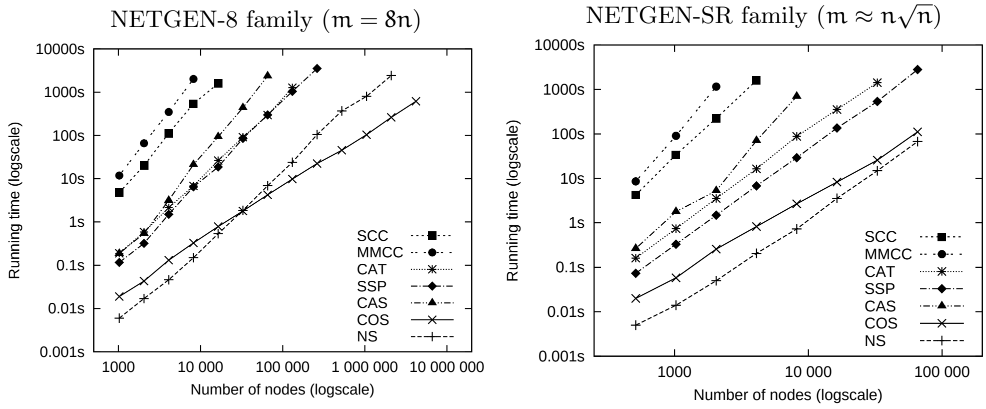

Figure 1: Comparison of our implementations on NETGEN instances

n	deg	SCC	MMCC	CAT	SSP	CAS	COS	NS
2^{10}	8	4.81	11.85	0.19	0.12	0.19	0.02	**0.01**
2^{12}	8	112.40	347.47	2.14	1.50	3.15	0.13	**0.05**
2^{14}	8	1587.01	–	26.36	18.66	93.14	0.78	**0.54**
2^{16}	8	–	–	295.05	298.95	2360.21	**4.24**	6.88
2^{18}	8	–	–	–	3514.72	–	**22.40**	104.69
2^{20}	8	–	–	–	–	–	**103.83**	799.26
2^{22}	8	–	–	–	–	–	**615.42**	–

Table 3: Comparison of our implementations on NETGEN-8 instances

n	deg	SCC	MMCC	CAT	SSP	CAS	COS	NS
2^{10}	32	34.03	90.82	0.74	0.33	1.79	0.06	**0.01**
2^{11}	45	224.79	1158.92	3.54	1.49	5.27	0.26	**0.05**
2^{12}	64	1592.62	–	16.36	6.77	70.35	0.83	**0.21**
2^{13}	91	–	–	88.13	29.16	697.47	2.68	**0.73**
2^{14}	128	–	–	353.23	136.17	–	8.28	**3.55**
2^{15}	181	–	–	1419.60	535.57	–	25.74	**14.90**
2^{16}	256	–	–	–	2799.34	–	111.55	**67.29**

Table 4: Comparison of our implementations on NETGEN-SR instances

Table 5 contains performance results for the NETGEN-LO-8 family. As one would expect, these instances turned out to be easier to solve than NETGEN-8 instances of the same size. For this family, SSP was an order of magnitude faster than CAT, while CAS was even more efficient than the SSP algorithm by a factor between 2 and 3. Note that the relative performance of SSP and CAS is entirely different compared to the NETGEN-8 results. Nevertheless, the fastest methods were COS and NS just like for the NETGEN-8 family and their relationship was similar.

n	deg	SCC	MMCC	CAT	SSP	CAS	COS	NS
2^{10}	8	0.82	2.52	0.14	0.02	0.01	0.01	**0.00**
2^{12}	8	8.49	79.93	1.88	0.13	0.06	0.07	**0.02**
2^{14}	8	73.83	1801.08	22.10	1.29	0.67	0.43	**0.19**
2^{16}	8	668.00	–	183.06	17.79	6.67	2.65	**2.11**
2^{18}	8	–	–	2062.12	172.63	60.16	**13.79**	29.00
2^{20}	8	–	–	–	1342.73	519.06	**68.41**	293.49
2^{22}	8	–	–	–	–	–	**457.02**	2482.50

Table 5: Comparison of our implementations on NETGEN-LO-8 instances

Table 6 shows how the running time of the algorithms depends on the density of the network. NS was clearly the most efficient algorithm in these tests. COS and SSP were also relatively fast, while CAT turned out to be significantly slower and the other methods were not competitive. The CAS algorithm performed much worse than SSP on the dense instances.

n	deg	SCC	MMCC	CAT	SSP	CAS	COS	NS
2^{12}	2	18.43	60.71	1.03	0.33	0.27	0.08	**0.02**
2^{12}	8	112.40	347.47	2.14	1.50	3.15	0.13	**0.05**
2^{12}	32	657.29	2655.80	7.41	4.40	35.54	0.43	**0.13**
2^{12}	128	3523.55	–	32.03	12.73	273.76	1.98	**0.39**
2^{12}	512	–	–	147.58	35.58	2473.16	7.72	**1.31**
2^{12}	2048	–	–	624.44	99.49	–	47.08	**6.68**

Table 6: Comparison of our implementations on NETGEN-DEG instances

Figure 2 and Tables 7 and 8 present the benchmark results for the GOTO families. These problems indeed turned out to be much harder than the NETGEN instances and the relative performance of the algorithms was also rather different than in case of the NETGEN families. Generally, COS and NS were the most efficient to solve these GOTO problems. COS was clearly the best method for the largest instances, even for the relatively dense GOTO-SR networks, which did not hold for the NETGEN-SR instances. Another notable difference compared to the NETGEN families is that CAS was orders of magnitude faster than SSP. It was quite efficient on GOTO-8 instances, although its performance was not stable. Among the cycle-canceling algorithms, CAT was clearly the most efficient similarly to the NETGEN problems. However, MMCC was 3-4 times faster than SCC, which is in contrast to the previous results.

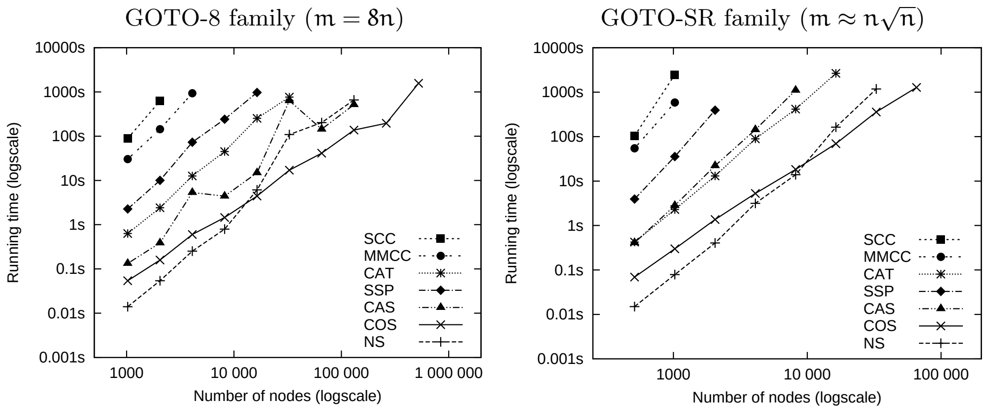

Figure 2: Comparison of our implementations on GOTO instances

n	deg	SCC	MMCC	CAT	SSP	CAS	COS	NS
2^{10}	8	88.85	30.39	0.63	2.27	0.13	0.05	**0.01**
2^{12}	8	–	931.54	12.62	72.93	5.34	0.60	**0.25**
2^{14}	8	–	–	253.54	971.03	14.87	**4.43**	6.11
2^{16}	8	–	–	–	–	143.92	**41.27**	202.47
2^{18}	8	–	–	–	–	–	**195.36**	–
2^{20}	8	–	–	–	–	–	–	–

Table 7: Comparison of our implementations on GOTO-8 instances

n	deg	SCC	MMCC	CAT	SSP	CAS	COS	NS
2^{10}	32	2474.46	584.35	2.30	35.66	2.81	0.30	**0.08**
2^{11}	45	–	–	13.08	393.00	22.20	1.36	**0.41**
2^{12}	64	–	–	89.15	–	143.35	5.27	**3.16**
2^{13}	91	–	–	415.69	–	1101.38	18.23	**13.87**
2^{14}	128	–	–	2650.95	–	–	**69.37**	163.37
2^{15}	181	–	–	–	–	–	**358.44**	1180.26
2^{16}	256	–	–	–	–	–	**1279.72**	–

Table 8: Comparison of our implementations on GOTO-SR instances

Table 9 presents the results obtained for the ROAD family. As one would expect, the SSP algorithm was by far the fastest on these special instances. Our implementation of the CAS algorithm implies that it works exactly the same as SSP on this family since all capacities are set to one. Therefore, CAS is skipped in Table 9. The COS and NS algorithms performed an order of magnitude worse than SSP, while the cycle-canceling algorithms were drastically slower.

n	deg	SCC	MMCC	CAT	SSP	COS	NS
9 559	3.11	2.29	372.91	2.17	**0.01**	0.14	0.04
49 109	2.46	37.25	–	47.65	**0.10**	1.47	0.69
116 920	2.27	150.84	–	264.52	**0.26**	4.61	2.97
261 155	2.38	1984.30	–	1373.49	**0.96**	15.23	14.14
519 157	2.44	–	–	–	**3.29**	35.87	41.38
1 048 506	2.53	–	–	–	**5.32**	94.32	129.04
2 073 870	2.49	–	–	–	**21.99**	238.57	744.36

Table 9: Comparison of our implementations on ROAD instances

Finally, the performance results for the VISION family are presented in Table 10. We do not report running time for SCC and MMCC as they could not solve these problems within the timeout limit of one hour. COS performed clearly the best in these tests with the only exception of the first instance. Similarly to other families, the asymptotic behavior of NS was clearly worse than that of COS. CAS was superior to SSP, while CAT was even slower than SSP.

n	deg	CAT	SSP	CAS	COS	NS
245 762	5.82	630.25	265.25	124.89	23.96	**21.06**
491 522	5.85	2304.01	970.49	622.22	**61.25**	108.64
983 042	5.88	–	–	1998.27	**186.42**	531.59
1 949 698	5.91	–	–	–	**535.36**	3420.76
3 899 394	5.92	–	–	–	**1475.95**	–

Table 10: Comparison of our implementations on VISION instances

Recall from Sections 3.3 and 3.4 that the COS and NS algorithms have several variants. These variants were also compared systematically to determine the default options. Here we only present a representative selection of these results.

Tables 11 and 12 compare the variants of the COS algorithm on the NETGEN-8 and GOTO-8 families, respectively. COS-PR denotes the push-relabel variant, COS-AR denotes the augment-relabel variant, while COS denotes the partial augment-relabel variant, which is the default implementation. This latter technique was clearly faster than the other two approaches on all kinds of problem instances. The COS-AR variant performed similarly to COS-PR on some easy instances, such as the NETGEN-8 family, but was an order of magnitude slower on some other instances, such as the GOTO networks. These results show that COS-AR is not so robust than the other two methods, which is in accordance with Goldberg's experiments in the maximum flow context [34].

n	deg	COS-PR	COS-AR	COS
2^{10}	8	0.03	**0.02**	**0.02**
2^{12}	8	0.17	0.14	**0.13**
2^{14}	8	0.94	1.11	**0.78**
2^{16}	8	6.37	5.72	**4.24**
2^{18}	8	35.17	28.00	**22.40**
2^{20}	8	176.87	179.13	**103.83**
2^{22}	8	1064.62	901.07	**615.42**

Table 11: Comparison of COS variants on NETGEN-8 instances

n	deg	COS-PR	COS-AR	COS
2^{10}	8	0.10	0.16	**0.05**
2^{12}	8	0.89	2.48	**0.60**
2^{14}	8	7.60	33.34	**4.43**
2^{16}	8	78.55	911.05	**41.27**
2^{18}	8	342.75	–	**195.36**

Table 12: Comparison of COS variants on GOTO-8 instances

We implemented five pivot rules for the NS method, which significantly affect the efficiency of the algorithm. Tables 13 and 14 compare the overall performance of these strategies on the NETGEN-8 and GOTO-8 families, respectively. BE, FE, BS, CL, and AL denote the best eligible, first eligible, block search, candidate list, and altering candidate list pivot rules, respectively. These results and many other experiments show that the BS and AL rules are generally the most efficient. On GOTO instances, the FE and CL rules also performed similarly to these methods, but they were much slower in other cases, for example, on NETGEN instances. Since the BS rule turned out to be slightly more robust than AL, it was selected to be the default pivot strategy in our implementation. The BE rule resulted in the worst performance although it yielded less iterations than the other rules.

n	deg	NS-BE	NS-FE	NS-BS	NS-CL	NS-AL
2^{10}	8	0.20	**0.01**	**0.01**	**0.01**	**0.01**
2^{12}	8	3.40	0.14	0.05	0.06	**0.04**
2^{14}	8	60.47	3.64	0.54	1.02	**0.47**
2^{16}	8	1285.91	117.99	6.88	27.68	**6.41**
2^{18}	8	–	–	104.69	808.10	**98.97**
2^{20}	8	–	–	**799.26**	–	800.36

Table 13: Comparison of NS pivot rules on NETGEN-8 instances

n	deg	NS-BE	NS-FE	NS-BS	NS-CL	NS-AL
2^{10}	8	0.50	**0.01**	**0.01**	**0.01**	0.02
2^{12}	8	9.59	0.43	**0.25**	**0.25**	0.28
2^{14}	8	151.92	6.84	6.11	**5.90**	6.16
2^{16}	8	3024.80	251.48	**202.47**	216.21	220.16

Table 14: Comparison of NS pivot rules on GOTO-8 instances

4.3 Comparison to other solvers

The implementations presented in this paper were also compared to the following widely known MCF solvers. These codes were compiled using the same compiler and optimization level as we used for our implementations. The experiments were conducted using the default options of these solvers.

- **CS2.** This is an authoritative implementation of the cost-scaling push-relabel algorithm. It was written by A.V. Goldberg and B. Cherkassky applying all improvements and heuristics described in [33]. CS2 has been widely used as a benchmark for solving the MCF problem for a long time. We used the latest version, CS2 4.6, which is available from the IG Systems, Inc. [16].

- **LEDA.** This is a comprehensive C++ library [54], which also provides an MCF solver in its `MIN_COST_FLOW()` procedure. This method implements the cost-scaling push-relabel algorithm, as well. We used version 5.0 of the LEDA library in our experiments. In fact, LEDA 5.1.1 was also tested, but it turned out to be slower than version 5.0.

- **MCFZIB.** This is the MCF code written by A. Löbel [57] at the Zuse Institute Berlin (ZIB). We denote this code as MCFZIB in order to differentiate it from the problem itself (similarly to the MCFClass project [27]). This solver features both a primal and a dual network simplex implementation, from which the former one is used by default as it is usually more efficient. We used the latest version 1.3, which is available at [58].

- **RelaxIV.** This is a C++ translation of an authoritative implementation of the relaxation algorithm. The original FORTRAN code was written by D.P. Bertsekas and P. Tseng [8] and is available at [7]. The C++ translation was made by A. Frangioni and C. Gentile at the University of Pisa and is available as part of the MCFClass project [27]. This project provides a common C++ interface for several MCF solvers. Apart from RelaxIV, it also features CS2 and MCFZIB, but not their latest versions, thus we used these two solvers directly.

Our codes are part of an open source C++ optimization library, LEMON, which is available at `http://lemon.cs.elte.hu/`.

Tables 15, 16, 17, and 18 compare our implementations to the other four solvers on the NETGEN instances. As before, all codes were executed with a timeout limit of one hour and the average running time over three different random instances is reported for each problem size (in seconds). Our COS code performed similarly to CS2 on NETGEN-8 and NETGEN-SR families, while it was slightly slower on the other two NETGEN families. The solver of the LEDA library was about two times slower than these cost-scaling codes. Furthermore, it failed to solve the largest instances due to a number overflow error, which is denoted as "*error*" in the tables. Since LEDA has closed source, we could not eliminate this problem by replacing the number types used by the algorithm with larger ones. MCFZIB was typically slower than our NS code by a factor between 2 and 10, but they performed similarly on the NETGEN-LO-8 instances. RelaxIV was very efficient on these families. It was typically faster than all other codes for the largest instances, while NS was the most efficient on the smaller networks and on the NETGEN-DEG family.

		LEMON		Other implementations			
n	deg	COS	NS	CS2	LEDA	MCFZIB	RelaxIV
2^{10}	8	0.02	**0.01**	0.02	0.03	0.02	**0.01**
2^{12}	8	0.13	**0.05**	0.11	0.17	0.14	0.06
2^{14}	8	0.78	**0.54**	0.78	1.45	1.75	**0.54**
2^{16}	8	4.24	6.88	4.22	8.34	18.78	**3.58**
2^{18}	8	22.40	104.69	20.81	*error*	207.29	**13.19**
2^{20}	8	103.83	799.26	103.25	*error*	2985.08	**89.90**
2^{22}	8	615.42	–	566.32	*error*	–	**419.38**

Table 15: Comparison to other solvers on NETGEN-8 instances

		LEMON		Other implementations			
n	deg	COS	NS	CS2	LEDA	MCFZIB	RelaxIV
2^{10}	32	0.06	**0.01**	0.05	0.08	0.05	0.03
2^{11}	45	0.26	**0.05**	0.18	0.38	0.20	0.26
2^{12}	64	0.83	**0.21**	0.59	1.41	0.68	1.30
2^{13}	91	2.68	**0.73**	2.04	5.07	3.28	1.94
2^{14}	128	8.28	**3.55**	7.86	19.61	21.68	4.53
2^{15}	181	25.74	14.90	29.00	72.48	111.74	**13.98**
2^{16}	256	111.55	67.29	104.25	274.06	634.38	**44.37**

Table 16: Comparison to other solvers on NETGEN-SR instances

		LEMON		Other implementations			
n	deg	COS	NS	CS2	LEDA	MCFZIB	RelaxIV
2^{10}	8	0.01	**0.00**	0.01	0.02	0.01	0.01
2^{12}	8	0.07	**0.02**	0.07	0.11	0.06	0.04
2^{14}	8	0.43	**0.19**	0.42	0.93	0.62	0.57
2^{16}	8	2.65	**2.11**	2.26	6.48	4.09	2.67
2^{18}	8	13.79	29.00	**10.56**	*error*	28.31	18.73
2^{20}	8	68.41	293.49	**54.48**	*error*	258.99	62.23
2^{22}	8	457.02	2482.50	299.15	*error*	2309.26	**229.67**

Table 17: Comparison to other solvers on NETGEN-LO-8 instances

		LEMON		Other implementations			
n	deg	COS	NS	CS2	LEDA	MCFZIB	RelaxIV
2^{12}	2	0.08	**0.02**	0.04	0.08	0.05	0.04
2^{12}	8	0.13	**0.05**	0.11	0.17	0.14	0.06
2^{12}	32	0.43	**0.13**	0.31	0.70	0.42	0.37
2^{12}	128	1.98	**0.39**	1.12	2.91	1.32	1.40
2^{12}	512	7.72	**1.31**	5.15	12.56	4.74	3.20
2^{12}	2048	47.08	**6.68**	32.72	69.55	18.52	12.81

Table 18: Comparison to other solvers on NETGEN-DEG instances

The performance results for the GOTO families are presented in Figure 3 and Tables 19 and 20. In these tests, COS and CS2 also performed similarly and they were the most efficient on the largest instances of both families. LEDA was also similarly efficient on GOTO-8 networks, but it was 2-3 times slower than CS2 and COS on the GOTO-SR family. NS turned out to be orders of magnitude faster than the other network simplex implementation, MCFZIB. Similarly to the NETGEN families, NS was the most efficient algorithm on the relatively small GOTO instances, but was substantially slower than the cost-scaling codes on the large networks. RelaxIV turned out to be very slow on these hard instances, which is in sharp contrast to its efficiency on the NETGEN instances.

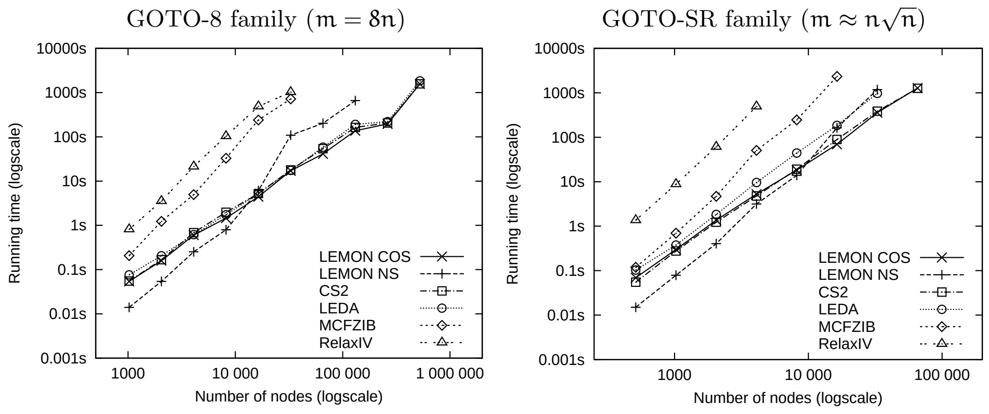

Figure 3: Comparison to other solvers on GOTO instances

		LEMON		Other implementations			
n	deg	COS	NS	CS2	LEDA	MCFZIB	RelaxIV
2^{10}	8	0.05	**0.01**	0.06	0.08	0.21	0.81
2^{12}	8	0.60	**0.25**	0.69	0.62	4.94	21.29
2^{14}	8	**4.43**	6.11	5.23	5.24	239.67	487.23
2^{16}	8	**41.27**	202.47	54.05	58.50	–	–
2^{18}	8	**195.36**	–	206.48	221.14	–	–
2^{20}	8	–	–	–	–	–	–

Table 19: Comparison to other solvers on GOTO-8 instances

		LEMON		Other implementations			
n	deg	COS	NS	CS2	LEDA	MCFZIB	RelaxIV
2^{10}	32	0.30	**0.08**	0.28	0.38	0.70	8.81
2^{11}	45	1.36	**0.41**	1.23	1.83	4.70	60.89
2^{12}	64	5.27	**3.16**	4.78	9.57	50.65	492.38
2^{13}	91	18.23	**13.87**	19.16	44.24	246.38	–
2^{14}	128	**69.37**	163.37	89.32	184.97	2339.57	–
2^{15}	181	**358.44**	1180.26	385.14	973.49	–	–
2^{16}	256	1279.72	–	**1259.63**	–	–	–

Table 20: Comparison to other solvers on GOTO-SR instances

Table 21 presents the results for the ROAD family. In this case, we also report the running time of our SSP implementation since it was by far the most efficient method for this special family. Apart from SSP, CS2 was typically the fastest to solve these instances. COS was slower than CS2 by a factor of at most two, while LEDA failed to solve the large instances due to overflow errors. NS and MCFZIB were significantly slower than CS2 and COS on the large networks. RelaxIV performed much worse than the other algorithms.

		LEMON			Other implementations			
n	deg	SSP	COS	NS	CS2	LEDA	MCFZIB	RelaxIV
9 559	3.11	**0.01**	0.14	0.04	0.12	0.14	0.14	0.58
49 109	2.46	**0.10**	1.47	0.69	0.84	1.07	1.62	22.41
116 920	2.27	**0.26**	4.61	2.97	2.67	3.19	6.09	168.03
261 155	2.38	**0.96**	15.23	14.14	7.53	*error*	27.25	2051.20
519 157	2.44	**3.29**	35.87	41.38	19.26	*error*	81.07	–
1 048 506	2.53	**5.32**	94.32	129.04	49.73	*error*	197.54	–
2 073 870	2.49	**21.99**	238.57	744.36	131.90	*error*	992.32	–

Table 21: Comparison to other solvers on ROAD instances

The benchmark results for the VISION family are presented in Table 22. In these tests, CS2 was clearly the most efficient and COS was about 1.5 times slower than it. All other algorithms performed significantly worse, including the third cost-scaling code, LEDA. NS was superior to MCFZIB and LEDA, but was less efficient than CS2 and COS for the large networks. Similarly to the GOTO and ROAD instances, RelaxIV was much slower than the other solvers.

		LEMON		Other implementations			
n	deg	COS	NS	CS2	LEDA	MCFZIB	RelaxIV
245 762	5.82	23.96	21.06	**19.78**	54.62	131.66	904.74
491 522	5.85	61.25	108.64	**44.18**	207.51	333.20	3518.20
983 042	5.88	186.42	531.59	**139.05**	1250.56	–	–
1 949 698	5.91	535.36	3420.76	**348.28**	–	–	–
3 899 394	5.92	1475.95	–	**916.47**	–	–	–

Table 22: Comparison to other solvers on VISION instances

Finally, on behalf of a brief overview of our experiments, Table 23 presents running time results for one representation of each problem family. Those instances were selected that have about two million arcs. The first part of the table contains running time in seconds, while the second part reports normalized time results. In the cases when the execution of an algorithm reached the timeout limit of one hour, we report a lower bound on the ratio by which it would have been slower than the fastest implementation.

Running time

		LEMON		Other implementations			
Family	m	COS	NS	CS2	LEDA	MCFZIB	RelaxIV
NETGEN-8	2 097 152	22.40	104.69	20.81	*error*	207.29	**13.19**
NETGEN-SR	2 097 152	8.28	**3.55**	7.86	19.61	21.68	4.53
NETGEN-LO-8	2 097 152	13.79	29.00	**10.56**	*error*	28.31	18.73
NETGEN-DEG	2 097 152	7.72	**1.31**	5.15	12.56	4.74	3.20
GOTO-8	2 097 152	**195.36**	–	206.48	221.14	–	–
GOTO-SR	2 097 152	**69.37**	163.37	89.32	184.97	2339.57	–
ROAD	2 653 624	94.32	129.04	**49.73**	*error*	197.54	–
VISION	2 877 382	61.25	108.64	**44.18**	207.51	333.20	3518.20

Normalized time

		LEMON		Other implementations			
Family	m	COS	NS	CS2	LEDA	MCFZIB	RelaxIV
NETGEN-8	2 097 152	1.70	7.94	1.58	*error*	15.72	**1.00**
NETGEN-SR	2 097 152	2.33	**1.00**	2.21	5.52	6.11	1.28
NETGEN-LO-8	2 097 152	1.31	2.75	**1.00**	*error*	2.68	1.77
NETGEN-DEG	2 097 152	5.89	**1.00**	3.93	9.59	3.62	2.44
GOTO-8	2 097 152	**1.00**	> 18	1.06	1.13	> 18	> 18
GOTO-SR	2 097 152	**1.00**	2.36	1.29	2.67	33.73	> 52
ROAD	2 653 624	1.90	2.59	**1.00**	*error*	3.97	> 38
VISION	2 877 382	1.39	2.46	**1.00**	4.70	7.54	79.63

Table 23: Comparison of our implementations to other solvers on various problem instances with roughly the same number of arcs

Table 23 as well as the previous results clearly demonstrate that CS2 and COS are the most robust implementations and NS is the third one in terms of the overall performance. MCFZIB is considerably slower than NS, but it is still robust. LEDA performed well in several cases, but it often failed to solve the large instances due to numerical problems. RelaxIV is not robust at all as it turned out to be much slower than the other implementations on all problem families except for the NETGEN networks.

5 Conclusions

In this paper, we have considered the minimum-cost flow (MCF) problem and various solution methods along with experiments with their efficient implementations. The MCF problem plays a fundamental role in network flow theory and has a wide range of applications. Therefore, efficient implementations of MCF algorithms are essential in practice.

We implemented several algorithms for solving the MCF problem and thoroughly experimented with many variants of them as well as with various practical improvements and heuristics. This work provides insight into these details and gives some guidelines for implementing the considered algorithms efficiently. An interesting novel result is the application of Goldberg's recent partial augment-relabel idea [34] in the cost-scaling algorithm, which turned out to be a significant improvement. Another widely used efficient algorithm is the network simplex method, which was implemented using a quite efficient data structure and various pivot strategies. Moreover, three cycle-canceling algorithms and two augmenting path algorithms were also implemented.

An extensive experimental evaluation was carried out to compare these algorithms. In general, the cost-scaling (COS) and the network simplex (NS) methods turned out to be the most efficient and the most robust. On relatively small instances (up to a few thousands of nodes), NS was clearly the fastest algorithm. However, COS significantly outperformed it on the largest networks due to its better asymptotic behavior in terms of the number of nodes. We also remark that NS usually performed better than other methods on rather dense networks, most likely because it is based on maintaining a spanning tree data structure and the tree update process depends only on the number of the nodes. Apart from COS and NS, the other algorithms usually performed worse and it turned out that their relative performance greatly depends on the characteristics of the problem instance. In certain cases, if the flow need not be split into many paths, however, the augmenting path algorithms are superior to other methods.

The presented implementations were systematically compared to publicly available efficient MCF solvers, as well. It turned out that our cost-scaling code is substantially more efficient and more robust than that of the LEDA library [54] and it performs similarly to or slightly slower than CS2 [33, 16], which is an authoritative implementation of this algorithm. Our implementation of the network simplex method turned out to be significantly faster than the other considered implementation of this algorithm, the MCF solver [57, 58]. Furthermore, another well-known MCF solver, RelaxIV [8, 7, 27] was also tested, but it did not turn out to be robust at all. It was orders of magnitude slower than the other codes on various problem families, although it was rather efficient on particular instances (namely, the NETGEN problems).

Our implementations are not standalone solvers, but they are part of a versatile C++ network optimization library, LEMON [55, 21] (`http://lemon.cs.elte.hu/`). Therefore, these codes have the additional advantage that they can easily be combined with various practical data structures and powerful algorithms related to network optimization. Furthermore, LEMON is an open source library that can be used in both commercial and non-commercial software development under a permissive license. The authors believe that this library with its great variety of efficient algorithms is a viable alternative to the MCFClass project [27], which features several publicly available MCF solvers under a common C++ interface.

Finally, we remark some ideas for future work. First, our implementation of the cost-scaling algorithm currently does not incorporate the speculative arc fixing heuristic, which was suggested by Goldberg [33]. We believe that the efficiency of this implementation could be further improved by also applying this complicated technique. Furthermore, a more comprehensive experimental study could be carried out considering more problem families and more publicly available implementations of MCF algorithms.

Acknowledgements

The authors are grateful to A.V. Goldberg and B. Cherkassky, D.P. Bertsekas and P. Tseng, and A. Löbel for developing CS2, RelaxIV, and MCF codes, respectively and for making these solvers available for academic use. We thank A. Frangioni and C. Gentile for developing the MCFClass project including the C++ translation of the RelaxIV solver, which was used in our experiments. We also thank A. Frangioni for his help in using the MCFClass project. Furthermore, we are grateful to A. Jüttner and B. Dezső for their useful suggestions related to the presented implementations.

Research was supported by grants (no. CNK 77780 and no. CK 80124) from the National Development Agency of Hungary, based on a source from the Research and Technology Innovation Fund, and by TÁMOP grant 4.2.1/B-09/1/KMR-2010-0003.

References

[1] R. K. Ahuja, A. V. Goldberg, J. B. Orlin, and R. E. Tarjan. Finding minimum-cost flows by double scaling. *Math. Program.*, 53:243–266, 1992. ⇒75

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Handbooks in Operations Research and Management Science, Volume 1*, pages 211–369. Elsevier, Amsterdam, The Netherlands, 1989. ⇒93

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1993. ⇒68, 71, 72, 74, 75, 76, 78, 80, 82, 83, 86, 91, 92, 93, 95, 96

[4] F. Barahona and É. Tardos. Note on Weintraub's minimum-cost circulation algorithm. *SIAM J. Comput.*, 18:579–583, 1989. ⇒77

[5] R. S. Barr, F. Glover, and D. Klingman. Enhancements to spanning tree labelling procedures for network optimization. *INFOR*, 17:16–34, 1979. ⇒91, 93, 94

[6] D. P. Bertsekas. A distributed algorithm for the assignment problem. Working Paper, Laboratory for Information and Decision Systems, MIT, Cambridge, MA, 1979. ⇒84

[7] D. P. Bertsekas. Network Optimization Codes. `http://web.mit.edu/dimitrib/www/noc.htm`, 1994. ⇒69, 108, 113

[8] D. P. Bertsekas and P. Tseng. RELAX-IV: A faster version of the relax code for solving minimum cost flow problems. Technical Report LIDS-P-2276, Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA, 1994. ⇒ 68, 69, 108, 113

[9] R. G. Bland and D. L. Jensen. On the computational behavior of a polynomial-time network flow algorithm. *Math. Program.*, 54:1–39, 1992. ⇒84

[10] G. Bradley, G. Brown, and G. Graves. Design and implementation of large scale primal transshipment algorithms. *Manag. Sci.*, 24:1–34, 1977. ⇒68, 91

[11] U. Bünnagel, B. Korte, and J. Vygen. Efficient implementation of the Goldberg-Tarjan minimum-cost flow algorithm. *Opt. Methods and Software*, 10:157–174, 1998. ⇒68, 86, 87, 88, 89

[12] R. G. Busacker and P. J. Gowen. A procedure for determining a family of minimum-cost network flow patterns. Technical Report ORO-TP-15, Operations Research Office, The Johns Hopkins University, Bethesda, MD, 1960. ⇒80

[13] R. G. Busacker and T. L. Saaty. *Finite Graphs and Networks: An Introduction with Applications.* McGraw-Hill, New York, NY, 1965. ⇒68, 71, 72

[14] COIN-OR – Computational Infrastructure for Operations Research. `http://www.coin-or.org/`, 2012. ⇒69

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, Cambridge, MA, 3rd edition, 2009. ⇒75, 77, 80

[16] CS2 Software, Version 4.6. IG Systems, Inc., `http://www.igsystems.com/cs2/`, 2009. ⇒69, 107, 113

[17] W. H. Cunningham. Theoretical properties of the network simplex method. *Math. Oper. Res.*, 4:196–208, 1979. ⇒97

[18] G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, NJ, 1963. ⇒74, 91

[19] A. Dasdan. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *ACM Trans. Des. Autom. Electron. Syst.*, 9:385–418, 2004. ⇒78

[20] A. Dasdan and R. K. Gupta. Faster maximum and minimum mean cycle algorithms for system performance analysis. *IEEE Trans. Computer-Aided Design*, 17:889–899, 1998. ⇒78

[21] B. Dezső, A. Jüttner, and P. Kovács. LEMON – an open source C++ graph template library. *Electron. Notes Theor. Comput. Sci.*, 264:23–45, 2011. ⇒70, 114

[22] DIMACS. Network Flows and Matching: First DIMACS Implementation Challenge. FTP site: `ftp://dimacs.rutgers.edu/pub/netflow/`, 2012. ⇒99, 100

[23] DIMACS. Shortest Paths: Ninth DIMACS Implementation Challenge. WWW site: `http://www.dis.uniroma1.it/~challenge9/`, 2012. ⇒100

[24] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19:248–264, 1972. ⇒75, 80, 81, 83

[25] MTA-ELTE Egerváry Research Group on Combinatorial Optimization (EGRES). `http://www.cs.elte.hu/egres/`, 2012. ⇒69

[26] L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, NJ, 1962. ⇒68, 71, 72, 74, 80

[27] A. Frangioni, and C. Gentile. The MCFClass Project. `http://www.di.unipi.it/di/groups/optimize/Software/MCF.html`, 2011. ⇒69, 107, 108, 113, 114

[28] A. Frangioni and A. Manca. A computational study of cost reoptimization for min-cost flow problems. *INFORMS J. Comput.*, 18:61–70, 2006. ⇒68

[29] A. Frank. *Connections in Combinatorial Optimization.* Oxford Lecture Series in Mathematics and Its Applications, Vol. 38. Oxford University Press, Oxford, UK, 2011. ⇒74

[30] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18:1013–1036, 1989. ⇒75

[31] L. Georgiadis, A. V. Goldberg, R. E. Tarjan, and R. F. Werneck. An experimental study of minimum mean cycle algorithms. In I. Finocchi and J. Hershberger, editors, *Proc. 6th International Workshop on Algorithm Engineering and Experiments*, ALENEX 2009, pages 1–13, New York, NY, 2009. SIAM. ⇒78

[32] A. V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM J. Comput.*, 24:494–504, 1995. ⇒88

[33] A. V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *J. Algorithms*, 22:1–29, 1997. ⇒68, 69, 86, 87, 88, 89, 107, 113, 114

[34] A. V. Goldberg. The partial augment-relabel algorithm for the maximum flow problem. In *Proc. 16th Annual European Symposium on Algorithms*, ESA '08, pages 466–477, Heidelberg, Germany, 2008. Springer-Verlag. ⇒68, 90, 101, 106, 113

[35] A. V. Goldberg and M. Kharitonov. On implementing scaling push-relabel algorithms for the minimum-cost flow problem. In D. S. Johnson and C. C. McGeoch, editors, *Network Flows and Matching: First DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 12, pages 157–198. American Mathematical Society, Providence, RI, 1993. ⇒86, 87, 89, 100

[36] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35:921–940, 1988. ⇒85

[37] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. ACM*, 36:873–886, 1989. ⇒77, 78, 79, 80

[38] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.*, 15:430–466, 1990. ⇒75, 84, 86, 89

[39] D. Goldfarb and J. Hao. Polynomial simplex algorithms for the minimum cost network flow problem. *Algorithmica*, 8:145–160, 1992. ⇒91

[40] D. Goldfarb, J. Hao, and S.-R. Kai. Anti-stalling pivot rules for the network simplex algorithm. *Networks*, 20:79–91, 1990. ⇒97

[41] M. D. Grigoriadis. An efficient implementation of the network simplex method. *Math. Program. Stud.*, 26:83–111, 1986. ⇒68, 91, 96, 97

[42] M. Hartmann and J. B. Orlin. Finding minimum cost to time ratio cycles with small integral transit times. *Networks*, 23:567–574, 1993. ⇒78

[43] R. A. Howard. *Dynamic Programming and Markov Processes.* The MIT Press, Cambridge, MA, 1960. ⇒78

[44] M. Iri. A new method for solving transportation-network problems. *J. Oper. Res. Soc. Japan*, 3:27–87, 1960. ⇒80

[45] W. S. Jewell. Optimal flow through networks. Technical Report Interim Technical Report No. 8, Operations Research Center, MIT, Cambridge, MA, 1958. ⇒80

[46] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Math.*, 23:309–311, 1978. ⇒74, 78

[47] D. J. Kelly and G. M. O'Neill. The minimum cost flow problem and the network simplex solution method. Master's thesis, University College, Dublin, 1991. ⇒ 91, 93, 95, 96, 97

[48] J. L. Kennington and R. V. Helgason. *Algorithms for Network Programming.* John Wiley & Sons, Inc., New York, NY, 1980. ⇒68, 91, 93

[49] Z. Király and P. Kovács. An experimental study of minimum cost flow algorithms. In *Proc. 8th International Conference on Applied Informatics*, Vol. 2., pages 227–235, Eger, Hungary, 2010. ⇒68

[50] M. Klein. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Manag. Sci.*, 14:205–220, 1967. ⇒77

[51] D. Klingman, A. Napier, and J. Stutz. NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *Manag. Sci.*, 20:814–821, 1974. ⇒99

[52] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms.* Algorithms and Combinatorics, Vol. 21. Springer-Verlag, Berlin, Germany, 5th edition, 2012. ⇒68, 71, 72, 75, 78

[53] H. W. Kuhn. The Hungarian Method for the assignment problem. *Naval Res. Logist. Quart.*, 2:83–97, 1955. ⇒74

[54] The LEDA library, Version 5.0. Algorithmic Solutions Software GmbH, `http://www.algorithmic-solutions.com/leda/`, 2004. ⇒69, 107, 113

[55] LEMON – Library for Efficient Modeling and Optimization in Networks. `http://lemon.cs.elte.hu/`, 2012. ⇒69, 76, 114

[56] LEMON Documentation, Version 1.2.3. `http://lemon.cs.elte.hu/pub/doc/1.2.3/`, 2012. ⇒81

[57] A. Löbel. Solving large-scale real-world minimum-cost flow problems by a network simplex method. Technical Report SC 96-7, Zuse Institute Berlin (ZIB), Berlin, Germany, 1996. ⇒68, 69, 91, 107, 113

[58] A. Löbel. MCF Version 1.3 – A network simplex implementation. Available at www.zib.de, 2004. ⇒69, 108, 113

[59] J. M. Mulvey. Pivot strategies for primal-simplex network codes. *J. ACM*, 25:266–270, 1978. ⇒97

[60] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Oper. Res.*, 41:338–350, 1993. ⇒75, 83

[61] J. B. Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Math. Program.*, 78:109–129, 1997. ⇒91

[62] J. B. Orlin, S. A. Plotkin, and É. Tardos. Polynomial dual network simplex algorithms. *Math. Program.*, 60:255–276, 1993. ⇒91

[63] H. Röck. Scaling techniques for minimal cost network flows. In U. Pape, editor, *Discrete Structures and Algorithms*, pages 181–191, München, 1980. Carl Hanser. ⇒84

[64] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Algorithms and Combinatorics, Vol. 24. Springer-Verlag, Berlin, Germany, 2003. ⇒68, 71, 72, 75

[65] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. System Sci.*, 26:362–391, 1983. ⇒79, 86, 89

[66] P. T. Sokkalingam, R. K. Ahuja, and J. B. Orlin. New polynomial-time cycle-canceling algorithms for minimum-cost flows. *Networks*, 36:53–63, 2000. ⇒77

[67] É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5:247–255, 1985. ⇒75, 84

[68] R. E. Tarjan. Efficiency of the primal network simplex algorithm for the minimum-cost circulation problem. *Math. Oper. Res.*, 16:272–291, 1991. ⇒ 91

[69] R. E. Tarjan. Dynamic trees as search trees via euler tours, applied to the network simplex algorithm. *Math. Program.*, 78:169–177, 1997. ⇒91

[70] N. Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1:173–194, 1971. ⇒75, 80

Received: March 15, 2012 • Revised: May 20, 2012

Acta Univ. Sapientiae, Informatica, **4,** 1 (2012) 119–129

Analysis of the picture cube puzzle

Péter BURCSI
Eötvös Loránd University
Budapest
email: bupe@compalg.inf.elte.hu

Abstract. In this paper we give a mathematical model for a game that we call picture cube puzzle and investigate its properties. The central question is the number of moves required to solve the puzzle. A mathematical discussion is followed by the description of computational results. We also give a generalization of the problem for finite groups.

1 Introduction

The picture cube puzzle[1] is a puzzle that consists of usually 6, 9 or 12 painted wooden cubes that can be arranged in a rectangular pattern to obtain six different pictures ("the solutions") seen on the top of the cubes. Each face of the cubes contains one piece of one of the six pictures in such a way that the position of that piece within the large picture is the same for all six faces. Thus, for example, there is a cube whose faces contain the upper left corners of the six pictures, another one that contains the lower right corners, etc.

The cubes are painted so that the solutions can easily be transformed into each other: for each picture there are cube rotations whose simultaneous application to all cubes transforms the picture to another one. For example, imagine that the puzzle is solved and you can see the picture of the dog on top. Pick up the first row of cubes holding them together and rotate the whole row around the axis through the centers of the cubes in that row by 90 degrees. Do this to all the rows and you obtain the picture of the bear. These row-wise or column-wise rotations are our allowable moves for the puzzle, see Figure 1.

Computing Classification System 1998: G.2.0
Mathematics Subject Classification 2010: Primary: 05C25, secondary: 05C12
Key words and phrases: Cayley graph, diameter, combinatorial puzzle

[1]These puzzles are usually sold in Hungary under the name "mesekocka", meaning fairy tale cube.

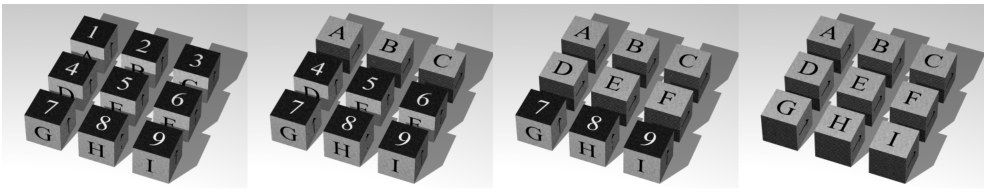

Figure 1: A picture cube puzzle. Rotating each row upwards transforms the picture of the numbers into the picture of the letters. We count this as three moves (there are three rows to rotate).

The number of allowable moves is twice the sum of the number of rows and the number of columns in the configuration. No move changes the position of the cubes within the picture but using the moves we can change which faces are on top and the orientation of the top face. A natural question is if we are given an arrangement of the cubes where the positions are correct but the cubes are arbitrarily rotated in place, can we solve the puzzle?

For the rest of the paper, we will suppose that one of the solved pictures is *the* solution configuration, and the cubes are somehow individually rotated (but kept in place). There are several questions that can be raised about the puzzle, we will formalize them in the following section. What are the configurations that can be reached from the solution configuration using allowable moves? How many such configurations exist? How many moves are needed to reach these configurations in the worst case/on average? We will address these questions after introducing a mathematical model using groups for the puzzle.

These (and other) questions have been raised for more well-known puzzles, notably Rubik's Cube [3]. Answering some of them requires the combination of non-trivial mathematical ideas with large-scale computer calculations. Diameters for permutation groups have been investigated e.g. in [1, 2].

The present paper is built up as follows. Section 2 gives the mathematical model and a generalization for the puzzle. In Sections 3 and 4 we discuss solution methods for the cube puzzle. In Section 5, some computational results are presented. We give further research directions and a short summary in Section 6.

We refer to any standard textbook, e.g. [4], for basic facts about groups.

2 Mathematical formulation of the puzzle

In order to formalize the puzzle, we need some notation. Let m and n be two positive integers and imagine mn cubes arranged in an m by n matrix form in front of us on a table. There is a solution configuration of the cubes when a picture can be seen on the top faces that we will refer to as "up" face, following standard terminology for Rubik's Cube. We will refer to the other faces as down, left, right, front and back faces in the straightforward way.

There are $2(m+n)$ moves that allow us to modify the configuration of the cubes: up_i and down_i, $(i = 1, \ldots, m)$ that rotate the cubes in row i along the axis through their centers in such a way that their front faces move to the up or down positions respectively; and left_j and right_j, $(j = 1, \ldots, n)$ that rotate the cubes in column j along the axis through their centers in such a way that their up faces move to the left or right positions respectively. We note that two consecutive "up" rotations on the same row have the same effect as two consecutive "down" rotations on that row (similarly for columns). We count these as two moves—using Rubik's Cube terminology, we use the quarter-turn metric in the present paper. When these transformations are counted as one move, we get the half-turn metric.

It is a classical observation in many similar puzzles that sequences of moves form a group under composition. Allowable moves form a generator set for that group and we identify the solution of the puzzle with the identity element. Applying a move to a configuration is multiplication by a generator from the right. Below we describe the group and the generators that we will use.

It is a well-known fact of group theory (seen most easily by observing how the transformations act on the main diagonals) that the rotation group of the cube is isomorphic to S_4, the symmetric group of degree 4. Thus we can represent any configuration (maybe not reachable by legal moves) of the puzzle by a matrix $T = (g_{ij}) \in S_4^{m\times n}$, where $S_4^{m\times n}$ is itself a group, being the direct product of mn copies of S_4. If we fix a way we represent rotations of a cube by S_4 then there are elements $u, d, l, r \in S_4$ such that turning a single cube up (resp. down/left/right) corresponds to multiplication (from the right) by u (resp. $d/l/r$). Using cycle notation, one may choose for example $u = (1423)$, $d = (1324)$, $l = (1342)$, $r = (1243)$. Note that all of them are odd permutations. Then up_i is the transformation that replaces g_{ij} by $g_{ij} \cdot u$ for $j = 1, \ldots, n$, in other words, up_i is (coordinate-wise) multiplication in $S_4^{m\times n}$ by an element that has n entries of u in the ith row and the identity element in every other position, and similarly for the other moves. Thus we may identify up_i and the other moves by some elements in $S_4^{m\times n}$. The set of moves in this group will be denoted by M. So $M = \{\text{up}_i, \text{down}_i \mid 1 \leq i \leq m\} \cup \{\text{left}_j, \text{right}_j \mid 1 \leq j \leq n\}$

We may now formulate several questions.

- **Reachability.** What are the states from which the solution is reachable using legal moves, or formally: what is the subgroup H of $S_4^{m\times n}$ generated by M? We are also interested in the order of this subgroup and how membership in this subgroup (solvability of a configuration) can be decided. Note that reachability between configurations is symmetric since every sequence of moves can be executed "backwards".

- **Solution.** Given an element in H, what is a sequence of moves that takes it to the identity (solve the puzzle). We are interested in human-executable ("easy") and computer-aided ("fast") techniques as well. What is the shortest sequence of moves, in other words, what is the shortest product of moves that gives $h \in H$?

- **Diameter.** What is the maximum length, taken over elements $h \in H$, of shortest sequences of moves taking h to the identity? In other words, how many moves are required in the worst case for solving the puzzle?

These questions can be analized by using Cayley graphs that are defined as follows.

Definition 1 *Let H be a group generated by M. The Cayley graph for H and M has H as the set of vertices and there is a directed edge from h_1 to h_2 iff for some $m \in M$, $h_1 m = h_2$.*

Solving the puzzle is the same as finding a path to the identity; and the diameter of this graph is exactly the length of the longest of all shortest paths from some h to the identity in the Cayley graph. We return to these questions in the next section.

2.1 Generalization for arbitrary finite groups

The mathematical formulation above allows us to generalize the problem for arbitrary finite groups. Let G be a finite group and let $R, C \subseteq G$ (row and column moves, respectively). Let n, m be positive integers and consider the group $G^{m\times n}$ with pointwise multiplication. For each $r \in R$ and $1 \leq i \leq m$ let $r_i \in G^{m\times n}$ be a matrix which has n entries equal to r in the ith row and the identity element $e \in G$ in other rows. (Here 'r' stands for row, not to be confused with the r for the original S_4 puzzle, where it is short for right, and is a column move.) For each $c \in C$ and $1 \leq j \leq n$ let $c_j \in G^{m \times n}$ be a matrix which has m entries c in the jth column and the identity element $e \in G$ in other columns:

$$r_i = \begin{pmatrix} e & e & e & \dots & \dots & e & e & e \\ e & e & e & \dots & \dots & e & e & e \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ e & e & e & \dots & \dots & e & e & e \\ r & r & r & \dots & \dots & r & r & r \\ e & e & e & \dots & \dots & e & e & e \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ e & e & e & \dots & \dots & e & e & e \end{pmatrix} \qquad c_j = \begin{pmatrix} e & e & \dots & e & c & e & \dots & e \\ e & e & \dots & e & c & e & \dots & e \\ e & e & \dots & e & c & e & \dots & e \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ e & e & \dots & e & c & e & \dots & e \\ e & e & \dots & e & c & e & \dots & e \\ e & e & \dots & e & c & e & \dots & e \end{pmatrix}$$

Definition 2 *Given* G, R, C, m *and* n *as above, the set of row moves is* $\{r_i \mid r \in R, 1 \leq i \leq m\}$*, the set of column moves is* $\{c_j \mid c \in C, 1 \leq j \leq n\}$*. The set of moves is their union* $M = M(G, R, C, m, n) = \{r_i \mid r \in R, 1 \leq i \leq m\} \cup \{c_j \mid c \in C, 1 \leq j \leq n\}$.

Definition 3 *Given* G, R, C, m *and* n *as above, the set of reachable configurations is* $H = H(G, R, C, m, n) = \langle M \rangle \leq G^{m \times n}$.

We can ask what the subgroup H is, how we can give a product of moves for an $h \in H$ and what the diameter of the Cayley graph for H and M is. These questions seem hard to answer in general, so we only focus on a few special cases that will be useful in the S_4 case which is a special case with $G = S_4$, $R = \{\text{up}, \text{down}\}$, $C = \{\text{left}, \text{right}\}$.

2.2 Abelian groups

If G is Abelian, then so is $G^{m \times n}$, meaning that the order in which we perform the moves has no effect on their product. Therefore, instead of sequences of moves, we can speak of sets of moves. Denote by a_i (resp. b_j) the product of moves performed on the ith row (resp. jth column). Note that a_i is not necessarily an element in R. Then the product of all the moves performed has the matrix form $T = (g_{ij})$ where $g_{ij} = a_i b_j$. We claim that the first row and the first column determine all other elements in T. To see this, let $i, j > 1$, then $g_{ij} = a_i b_j = (a_1 b_j)(a_i b_1)(a_1 b_1)^{-1} = g_{1j} g_{i1} g_{11}^{-1}$.

Consider the special case when $R = C = G$.

Theorem 4 *If* G *is Abelian,* $R = C = G$ *and* $n, m \in \mathbb{N}^+$*, then the reachable configurations are* $H = \{(a_i b_j)_{i=1,\dots,m,j=1,\dots,n} \mid a_i \in G, b_j \in G\}$*, and we have* $|H| = |G|^{m+n-1}$.

Proof. For any configuration in $G^{m\times n}$, we can solve the first row by using column operations, then solve the remaining elements of the first column by row operations. If the rest is not solved at this point, then the configuration is not in H, because the remaining elements of the configuration are uniquely determined by the first row and the first column for elements of H. Thus H has configurations of the form $(a_i b_j)_{i=1,\dots,m,j=1,\dots,n}$ if we let e.g. $a_i = g_{i1}$ and $b_j = g_{1j}/g_{11}$. It is also clear that H contains every element of this form. The number of required moves to solve the puzzle is $m + n - 1$ in the worst case. The order of H is $|G|^{m+n-1}$, because that is the number of ways we can adjust the elements in the first row and the first column. $\square$

If we weaken the conditions but require that both R and C generate G, then the solution method can be the same, but for solving the first row and column, a sequence of moves is required for each element.

We note that at least one special case of this Abelian version is part of mathematical folklore: when the group is the two-element group and the only moves are multiplication by the generator. This is often told with coins (and sometimes with lamps) arranged in a matrix form and allowed moves being the simultaneous turning over of entire rows or columns. The question is how we can tell if an initial configuration can be transformed into the "all heads" configuration.

The discussion gets more complicated when at least one of R and C generates only a nontrivial subgroup of G, but since we do not need this for the cube puzzle case, we omit the analysis of this case for brevity.

2.3 Simple groups

Another special case is when G is a non-abelian simple group. We investigate this case because the method used for solving it – the method of commutators – is also useful for the cube puzzle.

Definition 5 *Let* G *be a group,* $g_1, g_2 \in G$*. The commutator of* g_1 *and* g_2 *is* $[g_1, g_2] = g_1 g_2 g_1^{-1} g_2^{-1}$*. This is the identity element if and only if* g_1 *and* g_2 *commute. Note that in some sources in the literature, the inverses come first in the definition.*

Lemma 6 *Let* G *be a group,* $R, C \subseteq G$ *and* m, n *as above,* $r \in R, c \in C$,

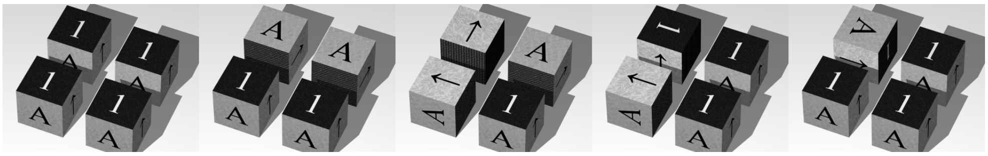
Figure 2: The figure illustrates how a commutator of a row move and a column move modifies only one entry. The sequence of moves up_1, left_1, down_1, right_1 cancels for all entries except at position $1, 1$.

$1 \le i \le m$, $1 \le j \le n$. *Then multiplying a configuration by* $[r_i, c_j]$ *only affects the entry at position* i, j.

Proof. Denote the entries of the configuration by $g_{i'j'}$. The transformation $t = [r_i, c_j] = r_i c_j r_i^{-1} c_j^{-1}$ can only alter elements in the ith row and the jth column. But if $j' \neq j$, then c_j does not modify the elements in column j', hence the effect of t on $g_{ij'}$ is multiplication by $r_i r_i^{-1} = e$, the identity. Similarly, $g_{i'j}$ is left intact by t if $i' \neq i$. Thus, the only element that can be affected is g_{ij}. The effect on this element is multiplication by $[r, c] = rcr^{-1}c^{-1}$. See Figure 2 for an illustration. □

Using this method we can individually modify the elements of (g_{ij}) by commutators of the form $[r, c]$ or $[c, r]$. More generally, using sequences of moves we can individually modify g_{ij} by $[r, c]$ or $[c, r]$ for $r \in \langle R \rangle$, $c \in \langle C \rangle$. Hence we have the following result.

Theorem 7 *Let* G *be a finite non-abelian simple group,* $R = C = G$*,* $m, n \in \mathbb{N}^+$. *Then* $H = H(G, R, C, m, n) = G^{m \times n}$*, in other words, every configuration is solvable.*

Proof. The commutator subgroup of G, denoted by G' is the subgroup generated by all commutators $[g_1, g_2]$ for $g_1, g_2 \in G$. By the above lemma, we can alter any individual entry of a configuration by any element in G'. But G' is always normal in G, hence if G is a finite simple non-abelian group, then $G' = G$. Therefore any configuration can be solved, by individually solving all entries. □

Note that the theorem also holds if we only assume $\langle R \rangle = \langle C \rangle = G$ – note that for $r \in R$, $r^{-1} \in \langle R \rangle$ by the finiteness of G. Also note that for Abelian groups, the commutator method is useless, since we have $G' = \{e\}$.

If G is a finite simple non-abelian group and $\langle R \rangle = \langle C \rangle = G$, then the diameter of the Cayley graph can be bounded both from below and above for general m and n using constants that depend only on G, R, C. The number of vertices of the graph is $|G|^{mn}$, the graph is $(m|R|+n|C|)$-regular. From this we get the lower bound $\log(|G|^{mn})/\log(m|R|+n|C|) - 1 = \text{const}_1 mn/\log(m|R|+n|C|)$. For the upper bound, note that we can solve each configuration in const_2 moves. Thus the number of moves required is at most $\text{const}_2 \cdot mn$:

$$\frac{\text{const}_1 \cdot mn}{\log(m|R|+n|C|)} \leq \text{diam}(\text{Cayley}(H, M)) \leq \text{const}_2 \cdot mn. \qquad (1)$$

The gap between the lower and upper bounds is logarithmic in m and n. It would be interesting to reduce this gap. We conjecture that the diameter is closer to the upper bound.

3 Local solution method for the cube puzzle

We return to the analysis of the original cube puzzle, where $G = S_4$ and the row moves are turning up and down ($R = \{u, d\}$), the column moves are turning left and right ($C = \{l, r\}$). The subgroup in $S_4^{m\times n}$ generated by all moves is again denoted by H. In this section we provide a decision method for solvability and present a method for solving using commutators. The method is simple and easy to perform for humans. We suppose that some configuration $s \in S_4^{m\times n}$ is given and we want to decide if it is in H, and if it is, we want to solve it.

We break the solution of the puzzle into two parts. First, we look for a sequence of moves t that transforms s to $st \in A_4^{m\times n}$, that is, we try to make every entry in s into an even permutation. If this is impossible, then $s \notin H$. If it is possible, we will solve the puzzle by using commutators.

For the first part note that since an up, down, left or right move toggles the parity of the affected elements (seen as permutations in S_4), the task of making every entry an even permutation can be reduced to solving a puzzle for the two-element multiplicative group $\{-1, 1\}$ where the legal row and column operations are multiplication of the row or column by -1. The Abelian analysis in the previous section shows that this can be examined by solving the first row and the first column and seeing if the rest is solved. If the rest is not solved, there is no solution for the puzzle. If there is a solution, it is found in at most $m+n-1$ steps. To perform the first part, the player has to remember which rotations correspond to even and odd permutations, but this is not too hard.

In the second part, we individually rotate all mn cubes to their solved positions, using the method of commutators. One readily verifies that commutators of the form $[u_i, l_j]$, $[u_i, r_j]$ etc. (altogether 8 types for a fixed pair i, j) are sufficient to transform any element of A_4 to the identity in at most 8 steps (two commutators suffice). This part then requires a worst case $8mn$ moves. The overall number of steps needed to solve any solvable configuration using this local method is thus at most $m + n - 1 + 8mn$.

We summarize the above discussion in the following theorem.

Theorem 8 *Let* $m, n \in \mathbb{N}^+$. *Then* $|H| = |H(S_4, R, C, m, n)| = 2^{m+n-1} \cdot 12^{mn}$. *The diameter of the Cayley graph of* H *and* M *is at most* $8mn + m + n - 1$.

4 Solution method using subgroups

Another method borrowing ideas from the Rubik's Cube literature [5, 6] is the method of subgroups generated by restricted moves. The method of the previous section can be considered as a special case of this method, but in general, the method is not intended for human execution. The method is most useful when the search space for finding the overall shortest path in the Cayley graph is too large, but a computer-aided search using subgroups finds shorter paths than the commutator-based method.

Take any nontrivial subgroup H_1 of H, in practice a subgroup with comparable index and order is preferable. Let the input for the method be an $h \in H$, and we proceed in two parts. First find the shortest sequence of moves that takes h into H_1, then solve the problem for H_1. One can also use a chain of subgroups, but we only consider the case of one subgroup. The subgroups of interest here can admit the following form. Let K be a subset of $\{1, \ldots, m\}$. We allow transformations generated by all possible moves with the restriction that for rows with index $i \in K$, only double moves $u_i^2 = d_i^2$ are allowed (in the half-turn metric this is extremely useful, since these moves count as one move). This method can be used to find a shorter move sequence than the naive method that is not necessarily optimal. The optimal choice of K is a nontrivial issue, in practice, computer experiments can be used to choose a good size for K.

5 Computational results

For small values of m and n we can use a computer to determine the diameter or other properties of the Cayley graph in our problem. The number of possible configurations grows exponentially in nm, so even for moderate values,

Size	Diameter	Average	Median	Local est.	Nr. of conf.
1×1	4	2.17	2	9	24
2×1	7	4.44	5	18	576
3×1	10	6.59	7	27	13824
4×1	12	8.59	9	36	331776
2×2	12	7.82	8	35	16588
5×1	15	10.69	11	45	7962624
6×1	17	12.65	13	54	191102976
3×2	14	10.54	11	52	47775744

Table 1: This table lists the maximal/average/median number of optimal moves needed to solve a puzzle of the given size. It also lists the number of moves that the local method from Section 3 takes and the number of reachable configurations.

Distance	0	1	2	3	4
Number of points	1	10	69	456	2846
Distance	5	6	7	8	9
Number of points	16208	84428	395566	1622641	5536264
Distance	10	11	12	13	14
Number of points	13587945	17558644	8100138	843444	27084

Table 2: The number of reachable configurations that are $0, 1, \ldots, 14$ moves away in the 3×2 puzzle.

we quickly run into memory storage problems. We represented configurations in a compact form using base 24 integer numbers. For the determination of the diameter and average distances, a breadth-search is performed, where the neighbors of a configuration are calculated using pre-stored 24-element rotation arrays – we listed in advance how the moves transform cube positions.

The tables summarize the results. Table 1 lists the number of reachable states, the average and median distance and the maximal distance for various puzzle sizes. Table 2 has details about the 3×2 case, listing how many configurations can be found on individual levels of the tree.

6 Summary and further directions

We gave a model for the picture cube puzzle which allowed us to answer some naturally arizing combinatorial questions. Mathematically, the study of general G, R, C and the asymptotic analysis of the diameter is the planned continuation of the present work. From a computational point of view, the further investigation of the exact values of the diameter for small m, n in the cube puzzle is our future plan.

Replacing the 2-dimensional configuration by higher dimensional ones is also a possible extension.

Acknowledgements

The Project is supported by the European Union and co-financed by the European Social Fund (grant agreement no. TÁMOP 4.2.1/B-09/1/KMR-2010-0003).

References

[1] L. Babai, G. L. Hetyei, On the diameter of random Cayley graphs of the symmetric group, *Comb. Probab. Comput.* **1** (1992) 201–208. $\Rightarrow$120
[2] L. Babai, Á. Seress, On the diameter of Cayley graphs of the symmetric group, *J. Comb. Theory, Ser. A* **49,** 1 (1988) 175–179. $\Rightarrow$120
[3] M. Davidson, J. Dethridge, H. Kociemba, T. Rokicki, God's number is 20, http://www.cube20.org/. $\Rightarrow$120
[4] M. Hall, Jr., *The Theory of Groups,* Macmillan, New York, 1959. $\Rightarrow$120
[5] H. Kociemba, http://kociemba.org/math/twophase.htm $\Rightarrow$127
[6] M. Thistlethwaite, http://www.jaapsch.net/puzzles/thistle.htm $\Rightarrow$127

Received: March 15, 2012 • Revised: April 23, 2012

Acta Univ. Sapientiae, Informatica, **4**, 1 (2012) 130–183

Deciding football sequences

Dedicated to the memory of Antal Bege (1962–2012)

Antal IVÁNYI
Eötvös Loránd University, Faculty of Informatics, Hungary
email: `ivanyi.antal2@upcmail.hu`

Jon E. SCHOENFIELD
Huntsville, Alabama, USA
email: `jonscho@hiwaay.net`

Abstract. An open problem posed by the first author [36, 53, 54, 59, 100] is the complexity to decide whether a sequence of nonnegative integer numbers can be the final score of a football tournament. In this paper we propose polynomial time approximate and exponential time exact algorithms which solve the problem.

1 Introduction

Let a, b and n be nonnegative integers ($b \ge a \ge 0$, $n \ge 1$), $\mathcal{T}(a, b, n)$ be the set of directed multigraphs $T = (V, E)$, where $|V| = n$, and each pair of different vertices u, $v \in V$ are connected with at least a and at most b arcs [56, 57]. $T \in \mathcal{T}(a, b, n)$ is called (a, b, n)-*tournament.* $(1, 1, n)$-tournaments are the usual tournaments, and $(0, 1, n)$-tournaments are also called *oriented graphs* or *simple directed graphs* [45, 93]. The set $\mathcal{T}$ is defined by

$$\mathcal{T} = \bigcup_{b \ge a \ge 0,\ n \ge 1} \mathcal{T}(a, b, n).$$

The definition of (undirected) (a, b, n)-*graphs* is similar. The $(0, 1, n)$-graphs are the usual *simple graphs.*

Computing Classification System 1998: G.2.2.
Mathematics Subject Classification 2010: 05C85, 68R10
Key words and phrases: tournament, score sequence, football tournament, polynomial algorithm

An (a, b, n)-tournament is called *complete,* if the set of permitted results is $\{0 : c, 1 : c-1, \dots, c : 0\}$ for all possible c $(a \le c \le b)$. If some of these results are prohibited, then the tournament is called *incomplete* [55, 56, 57] .

For example football is an incomplete $(2, 3, n)$-tournament since the permitted results are $0 : 3$, $1 : 1$ and $3 : 0$, while $0 : 2$, $1 : 2$, $2 : 0$, and $2 : 1$ are prohibited.

According to this definition $\mathcal{T}$ is the set of the finite directed loopless multigraphs. We remark, that if $a' \le a \le b \le b'$ then an (a, b, n)-tournament is also an (a', b', n)-tournament. The outdegree sequence of an (a, b, n)-tournament we call *the score sequence* of the tournament [45, 93, 95].

Let l, u, and m be integer numbers with $u \ge l$ and $m \ge 1$. The sequence $s = (s_1, \dots, s_m)$ of integer numbers with $l \le s_1 \le \cdots \le s_m \le u$ is called (l, u, m)-*regular.* It is well-known that the number of (l, u, m)-regular sequences is

$$R(l, u, m) = \binom{u - l + m}{m}. \tag{1}$$

In this paper we consider only the graph theoretical aspects of the investigated problems, although they have many applications [1, 16, 17, 68, 76, 88, 108]. We analyze only sequential algorithms. The Reader can find parallel results e.g. in [2, 29, 92, 102, 104].

The structure of the paper is as follows. After this introduction in Section 2 we deal with the filtering of potential complete sequences, then in Section 3 describe incomplete sequences. Section 4 contains filtering and Section 5 reconstruction algorithms of potential football sequences. Finally in Section 6 we deal with the enumeration of football sequences.

2 Filtering of potential complete sequences

We are seeking football sequences. Taking into account that a score sequence of an incomplete (a, b, n)-tournament is at the same time a score sequence of the complete (a, b, n)-tournament, the properties of score sequences of complete tournaments allow some filtering among the regular sequences.

In 1953 Landau [75] proved the following popular theorem. About ten proofs are summarized by Reid [95]. Further proofs are in [3, 18, 19, 20, 22, 44, 46, 101, 106, 110].

Theorem 1 (Landau [75]) *A* $(0, n-1, n)$-*regular sequence* $s = (s_1, \dots, s_n)$ *is* *the outdegree sequence of some* $(1, 1, n)$*-tournament if and only if*

$$\sum_{i=1}^{k} s_i \geq \frac{k(k-1)}{2}, \quad 1 \leq k \leq n, \tag{2}$$

with equality when $k = n$.

Proof. See [64, 75, 86]. □

Moon [85] proved the following generalization of Landau's theorem (we present it in reformulated form). Later Takahashi [107] reproved the theorem.

Theorem 2 (Moon [85]) *A* $(0, b(n-1), n)$*-regular sequence* $s = (s_1, \ldots, s_n)$ *is the score sequence of some* (b, b, n)*-tournament if and only if*

$$\sum_{i=1}^{k} s_i \geq \frac{bk(k-1)}{2}, \; 1 \leq k \leq n,$$

with equality when $k = n$.

Proof. See [85]. □

We define a *point-loss function* P_k $(k = 0, \ldots, n)$ by the following recursion: $P_0 = 0$ and if $1 \leq k \leq n$, then

$$P_k = \max\left(P_{k-1}, \; \frac{bk(k-1)}{2} - \sum_{i=1}^{k} s_i\right).$$

Now P_k gives a lower bound for the number of lost points in the matches among the teams $T_1, \ldots, T_k$ (not the exact value since the teams $T_1, \ldots, T_k$ could win points against $T_{k+1}, \ldots, T_n$).

Theorem 3 (Iványi [56]) *A* $(0, b(n-1), n)$*-regular sequence* $s = (s_1, \ldots, s_n)$ *is the score sequence of some complete* (a, b, n)*-tournament if and only if*

$$\frac{ak(k-1)}{2} \leq \sum_{i=1}^{k} s_i \leq \frac{bn(n-1)}{2} - P_k - (n-k)s_k \; (1 \leq k \leq n).$$

Proof. See [56]. □

3 Incomplete tournaments

We know only the following three results on the score sequences of incomplete tournaments.

Semicomplete digraphs (semicomplete tournaments) are defined as $(1, 2, n)$-digraphs in which if two vertices are connected with two arcs then these arcs have different directions.

Theorem 4 (Reid, Zhang [96]) *A* $(0, n-1, n)$*-regular sequence* $s = (s_1, \ldots, s_n)$ *is the score sequence of some semicomplete tournament if and only if*

$$\sum_{i=1}^{k} s_i \geq \frac{k(k-1)}{2} \quad \textit{and} \quad s_k \leq n-1, \quad 1 \leq k \leq n. \tag{3}$$

Proof. See [96]. □

Antal Bege asked in 1999 [7] how many wins are necessary in a football tournament of n teams to get a strictly monotone score sequence. If $n = 2$ then 1, if $n = 3$ then 1, and if $n = 4$ then 2 are sufficient and necessary. The following assertion gives the general answer.

Theorem 5 (Iványi [55]) *If* $N(n)$ *denotes the minimal number of necessary and sufficient wins for different scores in a football tournament of* n *teams then*

$$N(n) = \left(\frac{3}{2} - \sqrt{2}\right) n^2 + \Theta(n). \tag{4}$$

Recently Berger [10] published the following criterion for special incomplete $(0, 2, n)$-tournaments.

Theorem 6 (Berger [10]) *Sequence* $\sigma = \left(\binom{a_1}{b_1}, \ldots, \binom{a_n}{b_n}\right)$ *with* $a_1 \geq \cdots \geq a_n$ *is the score sequence of special incomplete* $(0, 2, n)$*-tournaments—in which* $0:0$*,* $0:1$*,* $1:0$*, and* $1:1$ *are the permitted results—if and only if*

$$\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{k} \min(b_i, k-1) + \sum_{i=k+1}^{n} \min(b_i, k) \tag{5}$$

for all $k = 1, \ldots, n$*, with equality for* n*.*

Proof. See [10]. □

Earlier (weaker) results can be found in [23, 41, 42, 99].

4 Filtering of potential football sequences

There are many exact results deciding whether a given sequence is the degree/outdegree sequence of a given type of undirected (e.g. [24, 25, 30, 31, 32, 33, 34, 35, 43, 46, 47, 48, 50, 51, 61, 69, 74, 83, 84, 90, 116, 118]) or directed (e.g. [12, 13, 75, 85, 56, 57, 94, 115]) graphs. Several authors studied the case when the indegree and outdegree sequences are together prescribed [9, 10, 14, 33, 48, 91].

The score sequences of the football tournaments we call *football sequences.* A $(0, 3n-3, n)$-regular sequence $s = (s_1, \ldots, s_n)$ is called *good* if there exists a football tournament whose score sequence is s, and s is called *bad* otherwise. We denote the football sequences by $f = (f_1, \ldots, f_n)$.

In this section we present approximate algorithms which filter only some part of the bad sequences. Since these filtering algorithms have short running time they help to reduce the expected running time of the exact algorithms.

The filtering algorithms are classified according to their worst running time as constant, linear, and other polynomial type ones.

4.1 Constant time filtering algorithms

The expected running time can be substantially decreased if we can filter some part of the investigated sequences in constant time.

Let $n \geq 2$ and $f = (f_1, \ldots, f_n)$ a football sequence.

Lemma 7 (C1 test) $f_n \neq 3n - 4$.

Proof. If a team wins all matches then its score is $3n-3$. If not, then it loses at least two points making a draw, so its score is at most $3n-5$. □

Lemma 8 (C2 test) *If* $f_n = 3n-3$ *then* $f_{n-1} \leq 3n-6$.

Proof. f_n can be $3n-3$ only so, that T_n wins all matches. Then the score T_{n-1} is at most $3n-6$. □

Lemma 9 (C3 test) *If* $f_1 = 0$ *then* $f_2 \geq 3$.

Proof. If $f_1 = 0$ then T_1 lost all matches therefore T_2 has at least one win and so f_2 is at least 3. □

Lemma 10 (C4 test) *If* $f_1 = f_2 = 1$ *then* $f_3 \geq 6$.

Proof. If $f_1 = f_2 = 1$ then the match of T_1 and T_2 ended with a draw implying that T_3 has at least two wins and so at least six points. □

Lemma 11 (C5 test) *If* $f_n = f_{n-1} = 3n - 5$*, then* $f_{n-2} \leq 3n - 9$.

Proof. If the joint score of T_n and T_{n-1} is $3n - 5$ then the result of their match has to be a draw. In this case T_{n-2} lost at least two matches and so $f_{n-2} \leq 3n - 9$. □

Lemma 12 (C6 test) *If* $f_n = 3n - 3$ *and* $f_{n-1} = 3n - 6$*, then* $f_{n-2} \leq 3n - 9$.

Proof. If $f_n = 3n - 3$, then T_n won all matches. In this case the score of T_{n-1} can be $3n - 6$ only then if T_{n-1} loses against T_n but wins all remaining matches. Then T_{n-2} lost at least two matches and so $f_{n-2} \leq 3n - 9$. □

Lemma 13 (C7 test) *If* $f_1 = 0$ *and* $f_2 = 3$ *then* $f_3 \geq 6$.

Proof. See the proof of Lemma 12. □

Lemma 14 (C8 test) *If* $f_1 = 1$ *and* $f_2 = 2$ *then* $f_3 \geq 4$.

Proof. Since T_1 and T_2 gathered points only with draws their match ended with a draw. Therefore T_3 won against T_1 and either won against T_2 or they made a draw, so T_3 has at least 4 points. □

Lemma 15 (C9 test) *If* $f_n = 3n - 5$ *and* $f_{n-1} = 3n - 7$ *then* $f_{n-2} \leq 3n - 8$.

Proof. If $f_n = 3n - 5$ then T_n has a draw and $n - 2$ wins. If $f_{n-1} = 3n - 7$ then T_{n-1} has two draws and $n - 3$ wins, and the match between T_n and T_{n-1} ended with a draw. In this case T_{n-2} has at least a loss and a draw implying $f_{n-2} \leq 3n - 8$. □

The following program CONSTANT realizes the tests of the previous 9 lemmas. This and later programs are written using the pseudocode conventions described in [27]. In this and in the further pseudocodes input variables are n: the length of the investigated sequence ($n \geq 3$); $s = (s_1, \ldots, s_n)$: a $(0, 3n - 3, n)$-regular sequence; output variable is L: $L = 0$ means that the investigated input is bad, $L = 1$ means that it is good while $L = 2$ shows that the given algorithm could not decide.

```
CONSTANT($n, s$)
01 $L = 0$                                                   // line 01: initialization of $L$
02 if $s_n == 3n - 4$                                        // line 02–03: C1
03     return $L$
04 if $s_n == 3n - 3$ and $s_{n-1} \geq 3n - 5$                  // line 04–05: C2
05     return $L$
06 if $s_1 == 0$ and $s_2 \leq 2$                                // line 06–07: C3
07     return $L$
08 if $s_1 == 1$ and $s_2 == 1$ and $s_3 \leq 5$                   // line 08–09: C4
09     return $L$
10 if $s_n == 3n - 5$ and $s_{n-1} = 3n - 5$ and $s_{n-2} \geq 3n - 8$   // line 10–11: C5
11     return $L$
12 if $s_n == 3n - 3$ and $s_{n-2} == 3n - 6$ and $s_{n-3} \geq 3n - 8$  // line 12–13: C6
13     return $L$
14 if $s_1 == 0$ and $s_2 == 3$ and $s_3 \leq 5$                   // line 14–15: C7
15     return $L$
16 if $s_1 == 1$ and $s_2 == 2$ and $s_3 \leq 3$                   // line 16–17: C8
17     return $L$
18 if $s_n == 3n - 5$ and $s_{n-1} == 3n - 7$ and $s_{n-2} \geq 3n - 8$  // line 18–19: C9
19     return $L$
20 $L = 2$                                  // line 20–21: these tests can not decide
21 return 2
```

Tables 1, 2, and 3 show the filtering results of Constant. The numbers in the tables show how many sequences are accepted from the sequences accepted by the previous filtering algorithm. The exact results in these tables are printed with bold font (such emphasizing will be used in the later tables too).

The programs are written in C by Loránd Lucz and run on an Inter Core i7 processor (3.4 GHz) with optimization level O3. The running times are given in seconds.

Table 2 shows the filtering results of C4, C5, C6 and C7.

Table 3 shows the filtering results of algorithms C8 and C9, further the number of football sequences (F) and the running time of Linear for $n = 1, \ldots, 15$ teams. Column R in Table 1 and column t in Table 3 show that the running time is approximately proportional with the number of the regular sequences.

For example if $n = 2$ then C1, C2 and C3 filter 80 % of the regular and 100 % of the bad sequences. If $n = 3$ then they filter 54 from the 84 regular sequences while C1, ..., C9 filter 70 sequences which represent 90.90 % of the

n	R	C1	C2	C3
1	1	**1**	**1**	**1**
2	10	7	4	**2**
3	84	63	45	30
4	715	550	414	311
5	6 188	4 823	3 718	2 911
6	54 264	42 636	33 320	26 650
7	480 700	379 753	299 421	242 624
8	4 292 145	3 404 115	2 700 775	2 207 800
9	38 567 375	30 678 375	24 452 220	20 116 030
10	348 330 136	277 722 676	222 146 496	183 629 160
11	3 159 461 960	2 523 716 572	2 024 386 180	1 679 655 640
12	28 760 021 745	23 008 017 396	18 498 140 232	15 394 304 500
13	262 596 783 764	210 345 382 913	169 436 070 190	141 355 053 635
14	2 403 979 904 200	1 927 719 734 500	1 555 302 958 664	1 300 210 775 786
15	22 057 981 462 440	17 704 432 489 590	14 303 680 429 990	11 978 596 958 384

Table 1: Number of $(0, 3n - 3, n)$-regular sequences (R) accepted by C1, C2, and C3 for $n = 1, \ldots, 15$ teams.

n	C4	C5	C6	C7
1	**1**	**1**	**1**	**1**
2	**2**	**2**	**2**	**2**
3	26	22	19	17
4	281	255	237	222
5	2 691	2 501	2 374	2 271
6	24 000	23 373	22 302	21 596
7	227 770	215 227	207 042	200 609
8	2 700 775	2 207 800	2 097 803	1 972 783
9	19 155 258	18 065 694	17 460 916	16 989 609
10	175 138 885	165 526 269	160 206 767	156 070 967
11	1 591 808 376	1 518 385 621	1 471 133 714	1 434 460 309
12	14 605 778 836	13 947 629 921	13 524 714 862	13 196 925 716
13	134 230 657 710	128 305 394 396	124 497 616 840	121 549 435 860
14	1 235 669 598 354	1 181 962 750 733	1 147 511 569 252	1 208 609 923 538
15	11 391 620 617 874	10 903 053 416 141	10 590 098 238 918	10 348 178 700 655

Table 2: Number of $(0, 3n - 3, n)$-regular sequences accepted by C4, C5, C6, and C7 for $n = 1, \ldots, 15$ teams.

bad sequences. If $n = 15$ then the nine constant time algorithms filter 54.73 % of the bad sequences. This is surprisingly high efficiency but smaller than the sum of the individual asymptotic efficiency of the 9 algorithms. The reason is simple: e. g. the sequence $s = (0, 0, 5)$ would be filtered by C1 and C3 too.

n	C8	C9	F	t
2	**2**	**2**	**2**	0.000
3	15	14	**7**	0.000
4	209	203	**40**	0.000
5	2 175	2 133	**355**	0.000
6	20 039	20 510	**3 678**	0.000
7	194 333	191 707	**37 263**	0.016
8	1 795 074	1 772 842	**361 058**	0.062
9	16 524 335	16 332 091	**3 403 613**	0.499
10	154 361 149	150 288 309	**31 653 777**	4.602
11	1 398 051 547	1 383 099 467	**292 547 199**	41.771
12	12 870 899 770	12 737 278 674	**2 696 619 716**	380.984
13	118 612 802 828	117 411 184 292		3 489.299
14	1 094 282 911 155	1 083 421 567 482		34 079.254
15	10 106 678 997 431	10 008 094 941 133		316 965.954

Table 3: Number of $(0, 3n-3, n)$-regular sequences accepted by C8 and C9, the number of football sequences (F), and the running time (t) of C9 for $n = 1, \ldots, 15$ teams.

4.2 Efficiency of the constant time testing algorithms

Using (1) we give the efficiency of the nine constant time filtering algorithms.

Lemma 16 (efficiency of C1) *The ratio of sequences with* $s_n = 3n-4$ *among* $(0, 3n-3, n)$*-regular sequences is*

$$\frac{\binom{4n-5}{n-1}}{\binom{4n-3}{n}} = \frac{n(3n-3)}{(4n-4)(4n-3)} = \frac{3}{16} + \frac{9}{16(4n-3)} = \frac{3}{16} + o(1). \tag{6}$$

Proof. The sequences satisfying the given condition are such $(0, 3n-3, n)$-regular ones, whose lower bound is $l = 0$, upper bound is $u = 3n-4$, and contain $m = n-1$ elements. So according to (1) the required ratio is

$$\frac{R(0, 3n-4, n-1)}{R(0, 3n-3, n)} = \frac{n(3n-3)}{(4n-4)(4n-3)} = \frac{3}{16} + o(1). \tag{7}$$

$\square$

Lemma 17 (efficiency of C2) *The ratio of the sequences satisfying the conditions* $s_n = 3n-3$ *and* $s_{n-1} \geq 3n-5$ *among the* $(0, 3n-3, n)$*-regular sequences is*

$$\frac{37}{256} + o(1). \tag{8}$$

Proof. Since $R(0, 3n-3, n-2)$ sequences satisfy the conditions $s_n = 3n-3$ and $s_{n-1} = 3n-3$, the corresponding ratio is

$$\frac{R(0, 3n-3, n-2)}{R(0, 3n-3, n)} = \frac{n(n-1)}{(4n-4)(4n-3)} = \frac{1}{16} + o(1). \tag{9}$$

$R(0, 3n-4, n-2)$ sequences satisfy $s_n = 3n-3$ and $s_{n-1} = 3n-4$, so the corresponding ratio is

$$\frac{R(0, 3n-4, n-2)}{R(0, 3n-3, n)} = \frac{n(n-1)(3n-3)}{(4n-3)(4n-4)(4n-5)} = \frac{3}{64} + o(1). \tag{10}$$

$R(0, 3n-5, n-2)$ sequences have the properties $s_n = 3n-3$ and $s_{n-1} = 3n-5$, so the corresponding ratio is

$$\frac{R(0, 3n-5, n-2)}{R(0, 3n-3, n)} = \frac{n(n-1)(3n-3)(3n-4)}{(4n-3)(4n-4)(4n-5)(4n-6)} = \frac{9}{256} + o(1). \tag{11}$$

Summing up the right sides (9), (10), and (11) we get the value (8). □

Lemma 18 (efficiency of C3) *The ratio of the sequences satisfying the conditions* $s_1 = 0$ *and* $s_2 \le 2$ *among the* $(0, 3n-3, n)$*-regular sequences is*

$$\frac{37}{256} + o(1). \tag{12}$$

Proof. Similar to the proof of Lemma 8. □

Lemma 19 (efficiency of C4) *The ratio of the sequences satisfying the conditions* $s_1 = 1$ *and* $s_2 = 1$ *and* $s_3 \le 5$ *among the* $(0, 3n-3, n)$*-regular sequences is*

$$\frac{2343}{4^8} + o(1). \tag{13}$$

Proof. Since $R(1, 3n-3, n-3)$ sequences satisfy the conditions $s_1 = s_2 = s_3 = 1$ the corresponding ratio is

$$\frac{R(1, 3n-3, n-3)}{R(0, 3n-3, n)} = \frac{n(n-1)(n-2)(3n-3)}{(4n-3)(4n-4)(4n-5)(4n-6)} = \frac{3}{4^4} + o(1). \tag{14}$$

The sequences with $s_1 = s_2 = 1$ and $s_3 = 2$, $s_1 = s_2 = 1$ and $s_3 = 3$, $s_1 = s_2 = 1$ and $s_3 = 4$, and $s_1 = s_2 = 1$ and $s_3 = 5$ have the asymptotic ratio $3/4^5$, $3/4^6$, $3/4^7$, and $3/4^8$ resp.

The sum of the received five ratios is

$$\frac{3}{4^4} + \frac{3^2}{4^5} + \frac{3^3}{4^6} + \frac{3^4}{4^7} + \frac{3^5}{4^7} = \frac{2343}{4^8}, \tag{15}$$

implying (13). □

Lemma 20 (efficiency of C5) *The ratio of the sequences satisfying the conditions of* $s_n = s_{n-1} = 3n-5$ *and* $s_{n-3} \geq 3n-8$ *among the* $(0, 3n-3, n)$*-regular sequences is*

$$\frac{1575}{4^8} + o(1). \tag{16}$$

Proof. We have to sum the contributions of $R(0, 3n-5, n-2)$, $R(0, 3n-6, n-2)$, $R(0, 3n-7, n-2)$, and $R(0, 3n-8, n-2)$ sequences:

$$\frac{3^2}{4^5} + \frac{3^3}{4^6} + \frac{3^5}{4^7} + \frac{3^6}{4^8} = \frac{1575}{4^8}, \tag{17}$$

implying (16). □

Lemma 21 (efficiency of C6) *The ratio of the sequences satisfying the conditions of* $s_n = 3n-3$, $s_{n-1} = 3n-6$, *and* $s_{n-2} \geq 3n-8$ *among the* $(0, 3n-3, n)$*-regular sequences is*

$$\frac{999}{4^8} + o(1). \tag{18}$$

Proof. In this case we sum the contributions of $R(0, 3n-6, n-3)$, $R(0, 3n-7, n-1)$, and $R(0, 3n-8, n-1)$ sequences:

$$\frac{3^3}{4^6} + \frac{3^4}{4^7} + \frac{3^5}{4^8} = \frac{999}{4^8}, \tag{19}$$

implying (18). □

Lemma 22 (efficiency of C7) *The ratio of the sequences satisfying the conditions of* $s_1 = 0$, $s_2 = 3$, *and* $s_3 \leq 5$ *among the* $(0, 3n-3, n)$*-regular sequences is*

$$\frac{999}{4^8} + o(1). \tag{20}$$

Proof. Similar to the proof of Lemma 21. □

Lemma 23 (efficiency of C8) *The ratio of the sequences satisfying the conditions of* $s_1 = 1$*,* $s_2 = 2$*, and* $s_3 \leq 3$ *among the* $(0, 3n-3, n)$*-regular sequences is*

$$\frac{63}{4^6} + o(1). \tag{21}$$

Proof. We sum the contributions of $R(2, 3n-3, n-3)$ and $R(3, 3n-3, n-3)$ sequences:

$$\frac{3^2}{4^5} + \frac{3^3}{4^6} = \frac{63}{4^6}, \tag{22}$$

implying (21). □

Lemma 24 (efficiency of C9) *The ratio of the sequences satisfying the conditions of* $s_n = 3n-5$*,* $s_{n-1} = 3n-7$*, and* $s_{n-2} \geq 3n-7$ *among the* $(0, 3n-3, n)$*-regular sequences is*

$$\frac{3^4}{4^7} + o(1). \tag{23}$$

Proof. Similar to the proof of Lemma 23. □

The cumulated asymptotic efficiency of the constant time algorithms is

$$\frac{3}{16} + \frac{2 \cdot 37}{4^4} + \frac{2343}{4^8} + \frac{1575}{4^8} + \frac{2 \cdot 999}{4^8} + \frac{63}{4^6} + \frac{3^4}{4^7} = \frac{38480}{4^8}. \tag{24}$$

The cumulated efficiency of the nine constant time algorithms is about 58.72 %. According to Table 1 the practical joint efficiency of C1, C2 and C3 is 64.28 % for $n = 3$ and 45.91 % for $n = 14$. According to Table 3 the total practical efficiency of the nine constant time algorithms is 91.67 % for $n = 3$ and 54.93 % for $n = 14$.

The practical cumulated efficiency is smaller than the theoretical one, since some part of the sequences is filtered by several algorithms: e.g. the sequence $s = (0, 0, 5)$ is filtered by C1 and C3 too.

We remark that the algorithms of Constant are sorted on the base of their nonincreasing asymptotic efficiency. We get the same order of the practical efficiency of these algorithms shown on the small values of n.

4.3 Filtering algorithms with linear running time

We investigate the following filtering algorithms whose worst running time is linear: Complete = L1, Point-Losses = L2, Reduction0 = L3, Reduction1 = L4, Draw-Unique = L5, Balanced = L6, Draw-Uniform = L7, Draw-Sorted-Unique = L8.

4.3.1 Linear filtering algorithm L1 = Complete

The first linear time filtering algorithm L1 = Complete is based on the following special case of Lemma 3 in [56].

Corollary 25 ((2,3,n)-complete test, [56]) *If* $n \geq 1$ *and* $(f_1, \dots, f_n)$ *is a football sequence then*

$$2\binom{k}{2} \leq \sum_{i=1}^{k} f_i \leq 3\binom{n}{2} - (n-k)f_k \quad (k = 1, \dots, n). \tag{25}$$

Basic parameters of Complete are the usual ones, further S: the current sum of the first i elements of s.

```
Complete($n, s$)
01 $S = 0$                                              // line 01: initialization of S
02 for $i = 1$ to $n$                                   // line 02–06: test
03     $S = S + s_i$
04     if $(S < 2\binom{i}{2}) \vee (S > 3\binom{n}{2} - (n-i)s_i) =$ TRUE
05         $L = 0$
06         return $L$
07 $L = 2$                                              // line 07–08: s is undecided
08 return $L$
```

4.3.2 Linear filtering algorithm L2 = Point-Losses

The second linear time filtering algorithm L2 = Point-Losses is based on the following assertion which is an extension of Lemma 3 in [56]. The basic idea is, that the small sums of the prefixes of s and the mod 3 remainders of the elements of s signalize lost points.

Lemma 26 *If* $(f_1, \dots, f_n)$ *is a football sequence then*

$$2\binom{k}{2} \leq \sum_{i=1}^{k} f_i \leq 3\binom{n}{2} - (n-k)f_k - P_k \quad (k = 1, \dots, n), \tag{26}$$

where $P_0 = 0$ *and*

$$P_k = \max\left(P_{k-1}, 3\binom{k}{2} - \sum_{i=1}^{k} f_i, \left\lceil \frac{\sum_{i=1}^{k}(f_i - 3\lfloor f_i/3 \rfloor)}{2} \right\rceil\right). \tag{27}$$

Proof. The sum of the k smallest scores is at least $2\binom{k}{2}$ and at most $3\binom{n}{2}$ minus the following point-losses:

1. the sum of the remaining scores, which is at least $(n-k)f_k$;
2. the point-losses due to draws documented by the mod 3 remainders;
3. the point-losses documented by differences $3\binom{k}{2} - \sum_{i=1}^{k} f_i$;

□

Basic parameters of Point-Losses are the usual ones, further S: the current sum of the first i elements of s, and P: the current value of the point-losses.

```
Point-Losses$(n, s)$
01 $S = P = L = 0$                                   // line 01: initialization of S, P, and L
02 for $k = 1$ to $n$                                // line 02–06: filtering
03     $S = S + s_k$
04     $P = \max\left(P_{k-1}, 3\binom{k}{2} - S, \left\lceil \frac{\sum_{i=1}^{k}(s_i - 3\lfloor s_i/3 \rfloor)}{2} \right\rceil\right)$
05     if $S > 3\binom{n}{2} - (n-k)s_k - P$
06         return $L$
07 $L = 2$
08 return $L$                                        // line 08: s is undecided
```

4.3.3 Linear filtering algorithm L3 = Reduction0

The third linear test is based on the observation that if the sum of the k smallest scores is minimal then all matches among the first k teams ended by a draw and if the sum of the k largest scores is maximal then the corresponding scores are multiples of 3 and further if $k < n$ then $f_{n-k} \le 3(n-k-1)$.

Lemma 27 *If* $n \ge 2$, $1 \le k \le n$, *and* $f = (f_1, \dots, f_n)$ *is a football sequence then*

1) if the sum of the first k *scores is* $k(k-1)$ *then* $f_1 = \cdots = f_k = k-1$ *and if further* $k < n$ *then* $f_{k+1} \ge 3k$*;*

2) if the sum of the last k *scores is* $3(n-k)k + 3\binom{k}{2}$ *then* $f_{n-k+1}, \dots, f_n$ *are multiples of 3 and if further* $k < n$ *then* $f_{n-k} \le 3(n-k-1)$.

Proof. If $f_1 + \cdots + f_k = k(k-1)$ then all matches among T_1, …, T_k ended with a draw and these teams lost all matches against the remaining teams implying assertions 1).

If $f_{k+1} + \cdots + f_n = 3(n-k)k + 3\binom{k}{2}$ then $T_{k+1}, \ldots, T_n$ won all matches against the remaining teams and have no draws implying assertion 2. □

Parameters of Reduction0 are the usual ones, further S: the current sum of the i smallest scores; Q: the current sum of the i largest scores; B is a logical variable characterizing the remainders mod 3 of the i largest scores.

```
Reduction0($n, s$)
01 $L = B = S = Q = 0$                    // line 01: initialization of L, B, S, and Q
02 **for** $i = 1$ **to** $n - 1$                  // line 02–12: test of the small scores
03      $S = S + s_i$
04      **if** $S == i(i-1)$
05         **if** $s_1 < i - 1 \vee s_i > i - 1$
06            **return** $L$
07         **if** $s_{i+1} < 3i$
08            **return** $L$
09 $S = S + s_n$
10 **if** $S == n(n-1)$
11    **if** $s_1 < n - 1$
12       **return** $L$
13 **for** $i = n$ **downto** 2                 // line 13–25: test of the large scores
14      $Q = Q + s_i$
15      **if** $s_{i-1} > 3(n - i - 1)$
16         **return** $L$
17      **if** $s_i - 3\lfloor s_i/3 \rfloor > 0$
18         $B = 1$
19      **if** $B == 1$
20         **return** $L$
21 $Q = Q + s_1$
22 **if** $s_1 - \lfloor s_1/3 \rfloor > 0$
23         $B = 1$
24         **if** $B == 1$
25            **return** $L$
26 $L = 2$                                     // line 26–27: $s$ is undecided
27 **return** $L$
```

Even this simple filtering algorithm finds a football sequence: if the condition of line 11 does not hold then the sum of all scores is minimal therefore all matches ended with draw. For the sake of the simplicity of the program we left this sequence undecided.

4.3.4 Linear filtering algorithm L4 = Reduction1

The fourth linear test is based on the observations that if the sum of the i smallest scores is $i(i-1)+1$ then either zero or one match among the first i teams ended with a win and if the sum of the i largest scores has near the maximal $3i(n-i)+3i(i-1)/2$ value then among the i maximal scores $i-2$ are multiples of 3 and 2 give 1 as remainder mod 3.

Lemma 28 *If* $n \geq 3$*,* $f = (f_1, \ldots, f_n)$ *is a football sequence,* $1 \leq k \leq n$ *then*

1) if

$$\sum_{i=1}^{k} f_i = k(k-1)+1 \tag{28}$$

then

a) either $f_1 = \cdots = f_{k-1} = k-1$*,* $f_k = k$*, and if* $k+1 \leq n$*, and* $f_{k+1} \geq 3k-2$*;*

b) or $f_1 = k-2$*,* $f_2, \ldots, f_{k-1} = k-1$*,* $f_k = k+1$*, and* $f_{k+1} \geq 3k$*;*

2) if

$$\sum_{i=1}^{k} f_{n-i+1} = 3k(n-k) + 3\binom{k}{2} - 1 \tag{29}$$

then

a) $\sum_{i=1,\ f_i - 3\lfloor f_i/3 \rfloor = 0}^{k} 1 = k-2$*;*

b) $\sum_{i=1,\ f_i - 3\lfloor f_i/3 \rfloor = 1}^{k} 1 = 2$*;*

c) $\sum_{i=1,\ f_i - 3\lfloor f_i/3 \rfloor = 2}^{k} 1 = 0$*;*

d) if $n-k > 0$ *then* $f_{n-k} \leq 3(n-k-1)$*.*

Proof. 1) If $f_1 + \cdots + f_k = k(k-1)+1$ then either all matches among T_1, ..., T_k ended with a draw and these teams lost all but one matches against the remaining teams and T_k made a draw with one of the teams T_k implying assertions a) or T_k won against T_1, the remaining matches among T_1, ..., T_k ended with a draw and the teams T_{k+1}, ..., T_n has no draw and won all matches against the first $n-k$ teams implying assertions b).

2) In case 2) of the lemma the teams T_{k+1}, ..., T_n won all matches against the first $n-k$ teams, and made exactly one draw. □

Parameters of Reduction1 are the usual ones, further S: the current sum of the first i scores; Q: the current sum of the last i scores; L_1 and L_2: logical variables; B is the number of scores giving remainder 1 mod 3; C is the number of scores giving remainder 0 mod 3.

```
Reduction1$(n, s)$
01 $L = B = C = S = Q = 0$        // line 01: initialization of $L$, $B$, $C$, $S$, and $Q$
```

```
02 for $i = 1$ to $n - 1$                    // line 02–12: test of the small scores
03      $S = S + s_i$
04      if $S == i(i - 1) + 1$
05         $L_1 = (s_1 == i - 1) \wedge (s_{i-1} == i - 1) \wedge (s_i == i) \wedge (s_{i+1} \geq 3i - 2)$
06         $L_2 = (s_1 == i - 2) \wedge (s_2 == i - 1) \wedge (s_{i-1} == i - 1)$
                 $\wedge (s_i == i + 1) \wedge (s_{i+1} \geq 3i)$
07         if $(L_1 ==$ FALSE$) \wedge (L_2 ==$ FALSE$) ==$ TRUE
08            return $L$                              // line 07–08: s is not good
09 $S = S + s_n$
10 if $S == n(n - 1) + 1$
11    if $(s_1 < n - 2) \wedge (s_2 == n - 1) \wedge (s_{n-1} == n - 1) \wedge (s_n == n + 1)$
           $==$ FALSE
12       return $L$
13 for $i = n$ downto 2                       // line 13–35: test of the large scores
14      $Q = Q + s_i$
15      if $s_i - 3\lfloor s_i/3 \rfloor == 2$
16         return $L$
17      if $s_i - 3\lfloor s_i/3 \rfloor == 1$
18         $B = B + 1$
19      if $s_i - 3\lfloor s_i/3 \rfloor > 0$
20         $C = C + 1$
21      if $Q == 3(n - i)i + 3i(i - 1)/2 - 1)$
22         if $s_{n-i} > 3(n - i - 1)$
23            return $L$
24         if $(B == 2) \wedge (C == i - 2) ==$ FALSE
25            return $L$
26 $Q = Q + s_1$
27 if $s_i - 3\lfloor s_i/3 \rfloor == 2$
28    return $L$
29 if $s_i - 3\lfloor s_i/3 \rfloor == 1$
30    $B = B + 1$
31 if $s_i - 3\lfloor s_i/3 \rfloor > 0$
32    $C = C + 1$
33 if $Q == 3n(n - 1)/2 - 1$
34    if $(B == 2) \wedge (C == i - 2) ==$ FALSE
35       return $L$
36 $L = 2$                                       // line 36–37: s is undecided
37 return $L$
```

4.3.5 Linear filtering algorithm L5 = Draw-Unique

A *draw sequence* $d(s) = (d_1, \ldots, d_n)$ belonging to a $(0, 3(n-1), n)$-regular sequence s accepted by L4 is defined as a sequence of nonnegative integers having the following properties for $i = 1, \ldots, n$:

1. $0 \le d_i \le 2$;
2. $d_i = s_i \mod 3$;
3. $d_i \le \min(s_i, n-1)$;
4. $d_i + 3(n-1-d_i) \ge s_i$,

further

$$\sum_{i=1}^{n} d_i = 2\left(3\binom{n}{2} - \sum_{i=1}^{n} s_i\right). \tag{30}$$

A draw sequence $d = (d_1, \ldots, d_n)$ is called $(0, 1, n)$-*graphic* (or simply graphic or good), if there exists a $(0, 1, n)$-graph whose degree sequence is d.

The fifth linear filtering algorithm is based on the following assertion.

Lemma 29 *If a $(2, 3, n)$-regular sequence s has only a unique draw sequence $d(s)$ which is not graphical then s is not football sequence.*

Proof. Since the football sequences have at least one graphical draw sequence, the regular sequences without graphical draw sequence are not football sequences. □

Basic parameters of Draw-Unique are the usual ones, further S: ithe current sum of the elements of s; R: the number of obligatory draws; Dn: the number of the draws in the investigated potential tournament; $d = (d_1, \ldots, d_n)$: d_i is the number of draws allocated to T_i; $r = (r_1, \ldots, r_n)$: r_i is the remainder of s_i mod 3; y: is the current number of allocated draws; x: is the current maximal number of draw packets acceptable by T_i.

```
Draw-Unique($n, s$)
01 $S = R = L = x = y = 0$          // line 01: initialization of $S$, $R$, $L$, $x$ and $y$
02 for $i = 1$ to $n$                 // line 02–03: computation of $S$
03     $S = S + s_i$
04 $Dn = 3\binom{n}{2} - S$                // line 04: computation of $Dn$
05 for $i = 1$ to $n$                 // line 05–17: allocation of draws
06     $r_i = s_i - 3\lfloor \frac{s_i}{3} \rfloor$
```

```
07         $R = R + r_i$
08         $x = \min\left(\frac{s_i - r_i}{3}, \lfloor\frac{n-1-r_i}{3}\rfloor, \lfloor\frac{3(n-1)-2r_i-s_i}{6}\rfloor\right)$
09         $d_i = r_i + 3x$
10         $y = y + d_i$
11 if $R > 2Dn$
12     return $L$, $d$
13 if $y < 2Dn$
14     return $L, d$
15 if $y \geq 2Dn$
16     $L = 2$
17     return $L$, $d$
18 sort $d$ in decreasing order by COUNTING-SORT resulting $d'$
19 HHL($d'$)
20 return $L$, $d$                                      // line 20: $s$ is undecided
```

Procedure HHL (HAVEL-HAKIMI-LINEAR) is described in [60]. We remark that the original Havel-Hakimi algorithm requires in worst case $\Theta(n^2)$ time. Recently Király [70] published a version which uses the data structure proposed by van Emde Boas [71, 114] and requires $O(n \log \log n)$ time. Our algorithm is linear and works also for some multigraphs.

A natural requirement is $d_i \leq n - 1$ but $d_i > n - 1$ can occur only in the cases $s = (0, 2)$ and $s = (1, 2)$ which are filtered by the constant time algorithms.

We get a stronger filtering algorithm DRAW-SORTED-UNIQUE using the definition of the uniqueness of the sorted draw sequence. For example in the case of the sequence $s = (3, 3, 3, 5)$ we have three possibilities to allocate two draw packets but only the teams having 3 points can accept a packet therefore we get in each case the bad draw sequence $(3, 3)$.

We remark that the problem of unicity of graphs determined in a unique way by their degree sequences was studied for some graph classes (see e.g. the papers of Tetali [109], Tyskevich [113], and Barrus [6]).

4.3.6 Linear filtering algorithm L6 = BALANCED-LIN

The sixth linear filtering algorithm L6 = BALANCED-LIN is based on the observation that if the draw sequence is unique, then the victory sequence $w = (w_1, \ldots, w_n)$ and the loss sequence $l = (l_1, \ldots, l_n)$ are also unique. The following assertion gives a necessary condition for the reconstructability of the sequence pair (w, l).

Lemma 30 (Lucz [77]) *If* $n \geq 2$, $w = (w_1, \ldots, w_n)$ *is the win sequence and* $l = (l_1, \ldots, l_n)$ *is the loss sequence of a football sequence* $f = (f_1, \ldots, f_n)$ *with* $\sum_{i=1}^{n} f_i > n(n-1)$ *then let*

$$w_i = \max_{1 \leq j \leq n} w_j \quad \text{and} \quad l_j = \max_{1 \leq i \leq n} l_i. \tag{31}$$

In this case

$$w_i \leq \sum_{j=1}^{i-1} \left\lceil \frac{l_j}{n-1} \right\rceil + \sum_{j=i+1}^{n} \left\lceil \frac{l_j}{n-1} \right\rceil \tag{32}$$

and

$$l_j \leq \sum_{i=1}^{j-1} \left\lceil \frac{w_i}{n-1} \right\rceil + \sum_{i=j+1}^{n} \left\lceil \frac{w_i}{n-1} \right\rceil \tag{33}$$

for $n = 1, \ldots, n$.

Proof. The wins (losses) of the team T_i (T_j) having the maximal number of wins (losses) can be paired with losses (wins) only if there are at least w_i (l_j) teams having at least one loss (win). □

4.3.7 Linear filtering algorithm L7 = Sport-Uniform

The seventh linear filtering algorithm L7 = Sport-Uniform is connected with a popular concept called in the world of sport *sport matrix.* It is an $n \times 5$ sized matrix containing the basic data of the teams of a tournament. We use the following formal definition of *sport matrix* for n teams.

Definition 31 *Let* $n \geq 1$ *and* $s = (s_1, \ldots, s_n)$ *be a* $(0, 3(n-1), n)$*-regular sequence. Then the sport matrices* $S(s)$ *corresponding to* s *are defined by the following properties:*

1. *the size of the matrix is* $n \times 5$*, its elements are nonnegative integers;*

2. $w_i + d_i + l_i = n - 1$ *for* $i = 1, \ldots, n$*;*

3. $3w_i + d_i = s_i$ *for* $i = 1, \ldots, n$*;*

4. $\sum_{i=1}^{n} w_i = \sum_{i=1}^{n} l_i = \sum_{i=1}^{n} s_i - n(n-1)$*;*

5. $\sum_{i=1}^{n} d_i = 2\left(3\binom{n}{2} - \sum_{i=1}^{n} s_i\right)$.

We remark that the ith row of the sport matrices contains data of T_i for $i = 1, \dots, n$: index i, number of wins w_i, number of draws d_i, number of losses l_i and number of points s_i (w_i, d_i and l_i are estimated values). These formal requirements are only *necessary* for $\mathcal{S}$ to contain the basic characteristics of some football tournament.

A sequence $s = (s_1, \dots, s_n)$ is called *sport sequence* if there exists at least one sport matrix corresponding to s.

Another useful concept is the *obligatory sport matrix* belonging to given regular sequence s.

Definition 32 *Let $n \geq 1$ be and $s = (s_1, \dots, s_n)$ be a $(0, 3(n-1), n)$-regular sequence. Then the obligatory sport matrix $\mathcal{O}(s)$ corresponding to s is defined by the following properties:*

1. *the size of the matrix is $n \times 5$, its elements are nonnegative integers;*
2. $wo_i = \max\left(0, \lceil \frac{s_i-(n-1)}{2} \rceil\right)$ *for* $i = 1, \dots, n$*;*
3. $do_i = s_i - 3\lfloor \frac{s_i}{3} \rfloor$ *for* $i = 1, \dots, n$*;*
4. $lo_i = \max(0, n-1-s_i)$ *for* $i = 1, \dots, n$. $\square$

The i-th row of the matrix contains the (partially estimated) data of T_i for $i = 1, \dots, n$: index i, number of obligatory wins wo_i, number of obligatory draws do_i, number of obligatory losses lo_i and number of points s_i (the obligatory values are lower bounds for the correct values, the index and the number of points are exact values).

Definition 33 *We say that the obligatory sport matrix $\mathcal{O}(s)$ of s is extendable to a sport matrix $\mathcal{S}(s)$ corresponding to s if*

1. *$\mathcal{O}(s)$ is a sport matrix belonging to s or*
2. *we can increase some w_i, d_i and l_i values so that the result will be a sport matrix $\mathcal{S}(s)$.*

According to the following assertion we get a linear filtering algorithm using the obligatory sport matrix.

Lemma 34 *The obligatory sport matrix $\mathcal{O}(s)$ belonging to a $(0, 3(n-1), n)$-regular sequence s is unique. If $\mathcal{O}(s)$ is not extendable to a sport matrix $\mathcal{S}(s)$ then s is not a football sequence.*

Proof. The obligatory sport matrix is defined by unique formulas therefore it is unique. If s is a football sequence then its obligatory sport matrix $\mathcal{O}(s)$ contains lower bounds for w_i, d_i, and l_i of any sport matrix $\mathcal{S}(s)$ therefore any sport matrix $\mathcal{S}(s)$ can be constructed by the extension of $\mathcal{O}(s)$. □

The following SPORT-UNIFORM is a *draw-based* algorithm which at first constructs the obligatory sport matrix belonging to s then tries to extend it to a sport matrix so that it allocates the draw packets in a greedy way as uniformly as possible. If the so received draw sequence is not graphic then the investigated sequence is not good.

The base of the uniform allocation of the draws is the following assertion.

Lemma 35 *If* $n \geq 1$, $d = (d_1, \ldots, d_n)$ *is graphical and* $d_i < d_j$ *then the sequence* d'*—received increasing* d_i *by* 1 *and decreasing* d_j *by* 1*—is also graphical.*

Proof. Let G be a $(0, 1, n)$-graph on vertices $V_1, \ldots, V_n$ having the degree sequence $d = (d_1, \ldots, d_n)$ in which $d_i < d_j$. Then there exists a vertex V_k which is connected with V_j and not connected with V_i. In G delete the edge between V_j and V_k and add the edge between V_i and V_k. Then the received new graph is graphical with the required degree sequence. □

This lemma has a useful corollary.

Corollary 36 *If* $n \geq 1$, $s = (s_1, \ldots, s_n)$ *is a* $(2, 3, n)$*-regular sequence, and its uniform draw sequence* $u(s) = (u_1, \ldots, u_n)$ *is not graphical, then* s *is not a football sequence.*

Proof. By the recursive application of Lemma 35 we get that if s has a graphical draw sequence then its uniform draw sequence is also graphical. □

We remark that the problem of the pairing of the draws has a reach bibliography as the problem of degree sequences of simple graphs [24, 32, 47, 50, 51, 62, 80, 89, 107, 110, 111, 112].

Basic parameters of SPORT-UNIFORM are the usual ones further S: the sum of the elements of s; S_0: auxiliary variable; $wo = (wo_1, \ldots, wo_n)$: wo_i is the number of obligatory wins of T_i; $do = (do_1, \ldots, do_n)$: do_i is the number of obligatory draws of T_i; $lo = (lo_1, \ldots, lo_n)$: lo_i is the number of obligatory losses of T_i; $WO = (WO_0, \ldots, WO_n)$: WO_i is the total number of wins of the first i teams; $DO = (DO_0, \ldots, DO_n)$. DO_i is the total number of draws of the first i teams; $LO = (LO_0, \ldots, LO_n)$. LO_i is the total number of the first i teams; $wm = (wm_1, \ldots, wm_n)$: wm_i is the maximal number of wins of T_i;

$dm = (dm_1, \dots, dm_n)$: dm_i is the maximal number of draw packets of T_i; $lm = (lm_1, \dots, lm_n)$: lm_i is the maximal number of losses of T_i; Wn: the number of the wins in the tournament; Ln: the number of the losses in the tournaments; Dn: the number of draws in the tournament; D: the current number of yet not allocated draws; $da = (da_1, \dots, da_n)$: da_i is the number of allocated to T_i draw packets; $wa = (wa_1, \dots, wa_n)$: wa_i is the number of allocated wins of T_i; $la = (la_1, \dots, la_n)$: la_i is the number of allocated losses of T_i; $R = (R_0, R_1, R_2)$: R_i is the number of elements of s giving remainder i mod 3; c: average number of draw packets to allocate for a team.

```
Sport-Uniform($n, s$)
01 $S_0 = WO_0 = DO_0 = LO_0 = R_0 = R_1 = R_2 = L = 0$ // line 01: initialization
02 for $i = 1$ to $n$                  // line 02–03: computation of the parameters
03     $S = S + s_i$
04     $wo_i = \max\left(0, \left\lceil \frac{s_i-(n-1)}{2} \right\rceil\right)$
05     $WO_i = WO_{i-1} + wo_i$
06     $do_i = s_i - 3\lfloor \frac{s_i}{3} \rfloor$
07     $DO_i = DO_{i-1} + do_i$
08     $R_{do_i} = R_{do_i} + 1$
09     $lo_i = \max(n - 1 - s_i, 0)$
10     $LO_i = LO_{i-1} + lo_i$
11     $dm_i = \min\left(\frac{s_i-do_i}{3}, n-1-do_i, \lfloor \frac{3(n-1)-2do_i-s_i}{6} \rfloor\right)$
12     $wm_i = \frac{s_i-do_i}{3}$
13     $lm_i = \left\lfloor \frac{3(n-1)-s_i}{3} \right\rfloor$
14 $Wn = Ln = S_n - n(n-1)$                 // line 14: computation of $Wn$, $Ln$
15 $Dn = D = 3n(n-1)/2 - S$                  // line 15: computation of $Dn$, $D$
16 if $\frac{D-DO_n}{3} > \lfloor \frac{D-DO_n}{3} \rfloor$        // line 16–43: allocation of draw packets
17   return $L$
18 while $D > 0$
19       $c = \left\lfloor \frac{D}{R_0+R_1+R_2} \right\rfloor$
20       while $c \geq 1$
21             $R_0 = R_1 = R_2 = 0$
22             for $i = 1$ to $n$
23                 $da_i = \min(\frac{s_i-d_i}{3}, c)$
24                 $d_i = do_i + 3da_i$
25                 $D = D - 3da_i$
26                 if $d_i < dm_i$
```

```
27                    $R_{do_i} = R_{do_i} + 1$
28              $c = \left\lfloor \frac{D}{R_0+R_1+R_2} \right\rfloor$
29      if $0 < \frac{D}{3} \leq R_0$
30         for $i = 1$ to $n$
31            if $(D > 0 \wedge do_i == 0) ==$ TRUE
32               $d_i = d_i + 3$
33               $D = D - 3$
34      if $R_0 < \frac{D}{3} \leq R_0 + R_1$
35         for $i = 1$ to $n$
36            if $(D > 0 \wedge do_i == 0 \vee do_i == 1) ==$ TRUE
37               $d_i = d_i + 3$
38               $D = D - 3$
39      if $R_0 + R_1 > \frac{D}{3}$
40         for $i = 1$ to $n$
41            if $D > 0$
42               $d_i = d_i + 3$
43               $D = D - 3$
44 sort $d$ in decreasing order resulting $d'$
45 HHL($d'$)                        // line 44–45: sorting of the draw sequence
46 return $L$, $d$          // line 46: $s$ is undecided (if $L = 2$) or bad (if $L = 0$)
```

4.3.8 Linear filtering algorithm L8 = Draw-Sorted-Unique

The fifth linear filtering algorithm Draw-Unique exploits the fact that some football sequences have unique sport matrix implying the uniqueness of the draw sequence. The eighth linear algorithm L8 = Draw-Sorted-Unique exploits that the uniqueness of the sport matrix is not necessary to have a unique sorted draw sequence.

Sorted version of a sport matrix $\mathcal{S}(s)$ is denoted by $\overline{\mathcal{S}}(s)$ and is defined by the following property: if $1 \leq i < j \leq n$ then either $d'_i < d'_j$ or $d'_i = d'_j$ and $w'_i < w'_j$ or $d'_i = d'_j$ and $w'_i = w'_j$ and $i' < j'$ (d'_i is the draw value in the i-th row of the sorted matrix and i' is the original index belonging to d'_i).

Draw-Sorted-Unique is based on the following assertion.

Lemma 37 *If* $n \geq 1$, $s = (s_1, \ldots, s_n)$ *is a* $(2, 3, n)$*-regular sequence, the sorted versions of the sport matrices* $\mathcal{S}(s)$ *are identical and their joint draw sequence is not graphical, then* s *is not a football sequence.*

Basic parameters of Draw-Sorted-Unique are the usual ones, further S: the current sum of the elements of s; D: the number of the draws in the investigated potential tournament; $d = (d_1, \dots, d_n)$: d_i is the number of draws allocated to T_i; $do = (do_1, \dots, do_n)$: do_i is the number of obligatory draws of T_i; DO: the number of the obligatory draws in the tournament; $lo = (lo_1, \dots, lo_n)$: lo_i is the number of obligatory losses of T_i; LO is the number of obligatory losses in the tournament; $wo = (wo_1, \dots, wo_n)$: wo_i is the number of obligatory wins of T_i; WO is the number of obligatory wins in the tournament; $dm = (dm_1, \dots, dm_n)$: dm_i is the maximal number of draw packets which can be accepted by T_i; DM: the sum of the dm_i's; $lm = (lm_1, \dots, lm_n)$: lm_i is of the maximal number of losses of T_i; LM: the sum of the lm_i's; $wm = (wm_1, \dots, wm_n)$: wm_i is the maximal number of wins of T_i; WM: the sum of the wm_i's; Wn: the number of the wins in the tournament; Ln: the number of the losses in the tournaments; Dn: the number of draws in the tournament; D: the number of yet not allocated draws; $da = (da_1, \dots, da_n)$: da_i is the number of allocated to T_i draw packets; $wa = (wa_1, \dots, wa_n)$: w_i is the number of allocated to T_i wins; $la = (la_1, \dots, la_n)$: la_i is the number of allocated to T_i losses; h: the maximal number of draw packets assigned to a team; $R = R_{i,j}$: a $3 \times h$ sized matrix, where $R_{i,j}$ gives the number of teams which are able at most i draw packets and having score of form $3k + j$; $A = (A_0, A_1, A_2)$: A_j is the number of scores giving i mod (3; $B = (B_0, \dots, B_h)$: B_i is the number of teams which are able to accept at most i draw packets; z: number of draw pockets which the program tries to allocate to all teams; fs: first score among the scores receiving maximal number of draw pockets; Rm: critical value of the remainder (mod 3) of the scores.

```
DRAW-SORTED-UNIQUE($n, s$)
01 $S = WO = DO = LO = A_0 = A_1 = A_2 = L = 0$     // line 01: initialization
02 for $i = 1$ to $n$              // line 02–29: test of the obligatory sport matrix
03       $S = S + s_i$
04       $do_i = s_i - 3\lfloor s_i/3 \rfloor$
05       $DO = DO + do_i$
06       $wo_i = \max(0, \lceil \frac{s_i-(n-1)}{2} \rceil$
07       $WO = WO + wo_i$
08       $lo_i = \max(0, n - 1 - s_i)$
09       $LO = LO + lo_i$
10       $dm_i = \min\left(\frac{s_i-do_i}{3}, \frac{n-1-do_i}{3}, \frac{3(n-1)-2d_i-s_i}{6}\right)$
11       $DM = DM + dm_i$
12       $wm_i = \min\left(\frac{s_i-do_i}{3}, (N-1) - DO_i\right)$
```

```
13        $WM = WM + wm_i$
14        $lm_i = \min\left(\lfloor\frac{3(n-1)-s_i}{3}\rfloor, n-1-do_i\right)$
15        $LM = LM + lm_i$
16 $Dn = 3\binom{n}{2} - S$
17 $Wn = Ln = S - 2\binom{n}{2}$
18 if $DO > 2Dn$
19     return L
20 if $3DM < 2Dn$
21     return L
22 if $WO > Wn$
23     return L
24 if $WM < Wn$
25     return L
26 if $LO > Ln$
27     return L
28 if $LM < Ln$
29     return L
30 $h = \lfloor(n-1)/3\rfloor$                 // line 30–45: preparation of the allocation
31 for $i = 0$ to $h$
32        $B_i = 0$
33        for $j = 0$ to 2
34              $R_{j,i} = 0$
35 for $i = 1$ to $n$
36        $R_{do_i,dm_i} = R_{do_i,dm_i} + 1$
37 for $i = 1$ to $h$
38        for $j = 0$ to 2
39              $A_j = A_j + R_{i,j}$
40              $B_i = B_i + R_{i,j}$
41 $q = 0$
42 $A = A_0 + A_1 + A_2$
43 $D = 2Dn - DO$
44 $c = \lfloor\frac{D}{A}\rfloor$
45 $q = q + c$
46 while $c \geq 1$                            // line 46–78: allocation of the draws
47        $z = 0$
48        for $i = q - c + 1$ to $q$
49              $z = z + iB_i$
50              $D = D - 3z$
```

```
51             for $i = 0$ to 2
52                 for $j = q - c + 1$ to $q$
53                     $A_i = A_i - R_{i,j}$
54             $A = A_0 + A_1 + A_2$
55             $c = \lfloor \frac{D}{A} \rfloor$
56 if $q > 0$
57     for $i = 1$ to $n$
58         $d_i = d_i - 3\min(z, dm_i)$
59 if $D == 0$
60     go to 79
61 $fs = -1$
62 $Rm = 2$
63 if $D \leq A_1 + A2$
64     $Rm = 1$
65 if $D \leq A_1$
66     $Rm = 0$
67 for $i = 1$ to $n$
68     if $(dm_i > q) \wedge (do_i \leq Rm) ==$ TRUE
69         if $fs == -1$
70             $fs = s_i$
71         else if $s_i \neq fs$
72                 return L, d
73         if $do_i < Rm$
74             $d_i = d_i + 3$
75             $D = D - 3$
76         if $(do_i == Rm) \wedge (D > 0) ==$ TRUE
77             $d_i = d_i + 3$
78             $D = D - 3$
79 sort d in nonincreasing order resulting $d'$       // line 79–80: sorting of d
80 HHL($d'$)
81 return L, d                                 // line 81: return the result of HHL
```

Procedure HHL (Havel-Hakimi-Linear) is described in [60]. We remark that the original Havel-Hakimi algorithm requires in worst case $\Theta(n^2)$ time. Recently Király [70] published a quicker algorithm which uses the data structure proposed by van Emde Boas [27, 71, 114] and requires only $O(n \log\log n)$ time. Our algorithm is linear and works also for some multigraphs.

A natural requirement is $d_i \leq n - 1$ but $d_i > n - 1$ can occur only in the cases $s = (0, 2)$ and $s = (1, 2)$ which are filtered by the constant time algorithms.

4.3.9 Efficiency of linear time filtering algorithms

Linear is the union of the described linear time algorithm.

```
Linear($n, s$)
01 $L = 0$                // line 01: initialization of L
02 L1($n, s$)             // line 02–04: filtering by Complete
03 if $L = 0$
04     return L
05 L2($n, s$)             // line 05–07: filtering by Losses
06 if $L = 0$
07     return L
08 L3($n, s$)             // line 08–10: filtering by Reduction0
09 if $L = 0$
10     return L
11 L4($n, s$)             // line 11–13: filtering by Reduction1
12 if $L = 0$
13     return L
14 L5($n, s$)             // line 14–16: filtering by Draw-Unique
05 if $L = 0$
16     return L
17 L6($n, s$)             // line 17–19: filtering by Balanced
18 if $L = 0$
19     return L
20 L7($n, s$)             // line 20–22: filtering by Draw-Uniform
21 if $L = 0$
22     return L
23 L8($n, s$)             // line 23–25: filtering by Draw-Sorted-Unique
24 if $L = 0$
25     return L
26 $L = 1$                // line 26–27: the linear time algorithms can not decide
27 return L
```

Since all included algorithms have linear worst case running time, the total running time of Linear is also $O(n)$. Since the best running time of L1 is $O(1)$, therefore the best running time of Linear is also $O(1)$.

Tables 4 and 5 show the concrete filtering results of the linear time filtering algorithms. Table 4 contains the number of regular sequences (R), the number of sequences, accepted by C9, L1 = Complete-Test, L2 = Losses and L3 = Reduction0.

n	C9	L1	L2	L3 + L4
1	**1**	**1**	**1**	**1**
2	**2**	**2**	**2**	**2**
3	14	12	10	10
4	203	134	94	87
5	2133	1230	901	814
6	20518	10947	8348	7526
7	191707	97427	76526	69349
8	1772842	872234	699344	637735
9	16332091	7851193	6387443	5859125
10	150288309	71001641	58367243	53817029
11	1383099467	644668154	538591486	494427384
12	12737278674	5873396400	4888701306	4544762304
13	117411184292	53669099755	44823480671	41804695971
14	1083421567402	491669304392	411496549436	384847810936

Table 4: Results of filtering by linear tests tests L1, L2, and L3 + L4 for $n = 1, \ldots, 14$ teams.

Table 5 contains the number of sequences accepted by L4 = Reduction1, L5 = Draw-Unique, L6 = Balanced, L7 = Sport-Uniform and L8 = Inner-Draw, further the number of the football sequences (F) and the cumulated running time and (the exact values of L7 are bold).

4.4 Quadratic filtering algorithms

In this section the quadratic recursive filtering algorithms Q1 = Balanced-Quad, Q2 = Reduction-Rec-Small, and Q3 = Reduction-Rec-Large are described.

4.4.1 Quadratic filtering algorithm Q1 = Balanced-Quad

The filtering algorithm Q1 = Balanced-Quad is based on the observation that if the draw sequence is unique, then the victory sequence $w = (w_1, \ldots, w_n)$ and the corresponding loss sequence $l = (l_1, \ldots, l_n)$ are also unique, further that the wins (losses) of any subset of teams have to be paired with inner and outer losses (wins). The following assertion gives a necessary condition for the reconstructability of the sequence pair (v, l).

n	L5 + L6	L7 + L8	F	t
1	**1**	**1**	**1**	0.000
2	**2**	**2**	**2**	0.000
3	**7**	**7**	**7**	0.000
4	46	**40**	**40**	0.000
5	475	365	355	0.000
6	4459	4086	3678	0.015
7	47867	44657	37263	0.047
8	460153	451213	361058	0.437
9	4371783	4348655	3403613	4.196
10	41261057	41166157	31653777	40.217
11	387821927	387416935	292547199	393.280
12	3635039265	3633749149	2696619716	3828.002
13	34011137972	33821636274		37611.185
14	317827900632	316291028902		364978.049

Table 5: Results of filtering by linear tests L5 + L6 and L7 + L8, further the number of football sequences (F) and the running time of L8 (t) for 1, ..., 14 teams.

Lemma 38 *If* $(a_1, \ldots, a_n)$ *is the monotone nonincreaing win sequence and* $(b_1, \ldots, b_n)$ *is the corresponding loss sequence of a football tournament then*

$$\sum_{i=1}^{k} a_i \le \sum_{i=1}^{k} \min(b_i, k-1) + \sum_{i=k+1}^{n} \min(b_i, k) \tag{34}$$

for all $k = 1, \ldots, n,$ *with equality for* n.

Proof. The wins included in the sum of the left side of (34) have to be paired with the "inner losses" (losses among T_1, ..., T_k) and "outer losses" (losses of T_1, ..., T_k in the matches against the remaining teams). $\square$

We remark that this lemma is a consequence of Theorem 3 of the recent paper due to Berger [10] containing a necessary and sufficient condition for some incomplete $(0, 2, n)$-tournaments. As the sequence $(1, 1, 8, 9, 9)$ satisfying 34 shows, in our case (34) is only a necessary condition, since s has a unique sport matrix shown in Table 6 which is not reconstructable.

The paper [33] contains an algorithm for our problem but the algorithm does not terminate for some inputs.

i	w_i	d_i	l_i	s_i
1	0	1	3	1
2	0	1	3	1
3	2	2	0	8
4	3	0	1	9
5	3	0	1	9

Table 6: Unique sport matrix belonging to the sequence $s = (1, 1, 8, 9, 9)$.

The following natural implementation BALANCED-QUAD of Lemma 38 requires quadratic time.

Parameters of BALANCED-QUAD are the usual ones, further $w = (w_1, \ldots, w_n)$: w_i is the number of wins allocated to T_i $(0 \le w_i \le n-1)$; $l = (l_1, \ldots, l_n)$: l_i is the number of losses allocated to T_i $(0 \le l_i \le n-1)$; Sw: the current number of the necessary wins; Ss: the maximal number of pairable losses of teams having small indices; Sl: the maximal number of pairable losses of the teams having large indices.

```
BALANCED-QUAD($n, w, l$)

01 $Sw = L = 0$                              // line 01: initialization of $Sw$ and $L$
02 sort $(w, l)$ nonincreasingly in $w$ using COUNTING-SORT
03 **for** $i = 1$ **to** $n$                 // line 03–13: counting of wins and losses
04      $Ss = Sl = 0$
05      $Sw = Sw = w_i$
06      **for** $j = 1$ **to** $i$                        // line 06–07: small indices
07           $Ss = Ss + \min(w_j, i-1)$
08      **for** $j = i+1$ **to** $n$                      // line 08–09: large indices
09           $Sl = Sl + \min(w_j, i)$
10      **if** $Sw > Ss + Sl$                   // line 10–13: $(w, l)$ is not pairable
11         **return** $L$
12 **if** $Sw < Ss + Sl$
13    **return** $L$
14 **else** $L = 2$                                    // line 14–15: $s$ is undecided
15       **return** $L$
```

We yet did not implemented BALANCED-QUAD.

4.4.2 Quadratic filtering algorithm Q2 = Reduction-Rec-Small

Algorithm Q2 = Reduction-Rec-Small is based on the recursive application of Recursive0 and Recursive1. Using Q2 and the next Q3 we shorten the input sequences and often can filter them.

Parameters are the usual ones, further $e = (e_1, \ldots, e_N)$: work version of the investigated sequence; n_l: smallest index of not deleted elements of s.

Reduction-Rec-Small(n, s)

```
01 $L = S = 0$                           // line 01–04: initialization of $L$, $S$, $n_l$, and $e$
02 for $i = 1$ to $n$
03      $e_i = s_i$
04 $n_l = 1$
05 while $n_l \leq n$
06          $S = 0$
07          for $i = n_l$ to $n$
08              $S = S + e_i$
09              if $S == i(i-1)$                         // line 09–21: $S$ is minimal
10                 if $i < n$
11                    if $(e_{n_l} \neq i-1) \vee (e_{n_l+i-1} \neq i-1) ==$ TRUE
12                       return $L$
13                    if $e_{n_l+i} < 3i$
14                       return $L$
15                 if $i == n$
16                    if $(e_{n_l} \neq i-1) \vee (e_n \neq i-1) ==$ TRUE
17                       return $L$
18                    else $L = 1$
19                          return $L$
20                 $n_l = n_l + i$
21                 for $j = n_l$ to $n$
22                      $e_j = e_j - 3i$
23                 go to 05
24              if $S == i(i-1) + 1$          // line 22–35: $S$ is minimum plus one
25                 if $i < n$
26                    $L_1 = (e_{n_l} = i-1) \wedge (e_{n_l+i-2} = i-1) \wedge (e_{n_l+i-1} = i)$
                            $\wedge(e_{n_l+i} \geq 3i-2)$
27                    $L_2 = (e_{n_l} = i-2) \wedge (e_{n_l+i-2} = i-1) \wedge (e_{n_l+i-1} = i+1)$
28                    if $(L_1 ==$ FALSE$) \wedge (L_2 ==$ FALSE$) ==$ TRUE
29                       return $L$         // line 28–29: $s$ is not football sequence
```

```
30             if $i == n$
31                $L_2 = (e_{n_l} = n - 2) \wedge (e_{n_l+1} = n - 1) \wedge (e_{n-1} = n + 1)$
32                if $L_2$ == FALSE   // line 32–33: $s$ is not football sequence
33                   return $L$
34             $n_l = n_l + i$
35             for $j = n_l$ to $n$
36                 $e_j = e_j - 3i$
37             go to 05
38          REDUCTION-REC-LARGE($n - n_l + 1, e$)
39          if $L == 0$
40             return $L$
41 if $n_u > 0$
42    FILTER($n_u, e$)
43 if $L == 0$
44    return $L$
45 $L = 2$                                   // line 45–46: $s$ is undecided
46 return $L$
```

Reduction-Rec-Small calls Filter which is a union of the constant and linear time filtering algorithms and Reduction-Rec-Large which is the next quadratic filtering algorithm.

Filter(n, e)

```
01 CONSTANT($n, e$)   // line 01–03: filtering by the constant time algorithms
02 if $L == 0$
03    return $L$
04 LINEAR($n, e$)        // line 04–06: filtering by the linear time algorithms
05 if $L == 0$
06    return $L$
07 $L = 2$                                   // line 07–08: $s$ is undecided
08 return $L$
```

4.4.3 Quadratic filtering algorithm Q3

Algorithm Q3 = Reduction-Rec-Large is based on the recursive application of Recursive0 and Recursive1.

Parameters are the usual ones, further $e = (e_1, \ldots, e_N)$: work version of the investigated sequence; n_u: smallest index of the not deleted elements of s; Q: the sum of the i largest scores; B: the number of investigated scores giving

0 remainder mod 3; C: the number of investigated scores giving remainder 1 mod 3; D: the number of investigated scores giving remainder 2 mod 3.

```
REDUCTION-REC-LARGE($n, e$)
01 $L = Q = B = C = D = 0$ // line 01–02: initialization of L, Q, B, C, D, $n_u$
02 $n_u = n$
03 while $n_u \leq 1$                                  // line 03–25: recursive reduction
04       for $i = n_u$ downto 1                   // line 04–09: preparing of the filtering
05             $Q = Q + e_i$
06             if $e_i - 3\lfloor e_i/3 \rfloor == 0$
07                $B = B + 1$
08             if $e_i - 3\lfloor e_i/3 \rfloor == 1$
09                $C = C + 1$
10             if $e_i - 3\lfloor e_i/3 \rfloor == 2$
11                $D = D + 1$
12             if $Q == 3i(n_u - i) + 3i(i - 1)/2$   // line 12–17: Q is maximal
13                if $B \neq i$
14                   return $L, n_u$
15                if $i > 1$
16                   if $e_{n_u - i} > 3(n_u - i - 1)$
17                   return $L, n_u$
18                $n_u = n_u - i$
19                go to 03
20             if $Q == 3i(n_u - i) + 3i(i - 1)/2 - 1$
21                if $(B == i - 2) \wedge (C == 2) ==$ FALSE
22                   return $L, n_u$
23                if $i > 1$                     // line 20–25: Q is maximum minus 1
24                   if $e_{n_u - i} > 3(n_u - i - 1)$
25                      return $L, n_u$
26 $L = 2$                                             // line 26–27: $s$ is not decided
27 return $L, n_u$
```

The following Table 7 contains the results of quadratic filtering algorithms.

5 Reconstruction of potential football sequences

In this part we investigate polynomial reconstruction algorithms, as R1 = REDUCTION, R2 = DRAW-UNIFORM-REC, and R3 = DRAW-INNER-REC.

n	Linear	Q2 + Q3	F	t
1	**1**	**1**	**1**	0.000
2	**2**	**2**	**2**	0.000
3	**7**	**7**	**7**	0.000
4	**40**	**40**	**40**	0.000
5	365	**355**	**355**	0.000
6	4086	3760	**3 678**	0.015
7	44657	39417	**37 273**	0.109
8	451213	393072	**361 058**	1.264
9	4348655	3804485	**3 403 613**	15.226
10	41166157	36302148	**31 653 777**	179.249
11	387416935	344012885	**292 547 199**	2066.323
12	3633749149	3246651763	**2 696 619 716**	23429.877
13	33821636274	30405902165		

Table 7: Results of filtering by Linear and quadratic algorithms Q2 + Q3, further the number of football sequences (F) and the running time of Q3 (t) for $n = 1, \ldots, 13$ teams.

5.1 Reconstruction algorithm R1 = Reduction

R1 = Reduction is based on filtering algorithms Reduction0 and Reduction1.

5.2 Reconstruction algorithm R2 = Draw-Uniform-Rec

R2 = Draw-Uniform-Rec is based on filtering algorithms: it tries—using the degree sequence d produced by Sport-Uniform or Draw-Sorted-Unique and using a greedy pairing algorithm "largest wins with largest losses"—to pair the wins and losses.

Parameters of R2 are the usual ones further S: sport matrix computed using the output draw sequence d of Sport-Uniform or Draw-Sorted-Unique and sorted its rows so that either $w_i > w_{i+1}$ or $w_i = w_{i+1}$ and $l_i \leq l_{i+1}$; $d = (d_1, \ldots, d_n)$: draw sequence of S; $\mathcal{M}_{n \times n}$ (result matrix): $\mathcal{M}_{i,j}$ is the number of points received by T_i in the match against T_j; $w = (w_1, \ldots, w_n)$: w_i is the number of wins of T_i; $l = (l_1, \ldots, l_n)$: l_i is the number of losses of T_i.

```
Draw-Uniform-Rec($n, s, d$)
01 for $i = 1$ to $n$                          // line 01–03: initialization of $\mathcal{M}$
```

```
02       for $j = 1$ to $n$
03           $\mathcal{M}_{i,j} = 0$
04 HAVEL-HAKIMI-DRAWS($n, s, d, \mathcal{M}$)   // line 04: HHD allocates the draws
05 for $i = 1$ to $n$                        // line 05–07: computation of $w$ and $l$
06       $w_i = (s_i - d_i)/3$
07       $l_i = n - 1 - d_i - w_i$
08 for $i = n$ downto $1$               // line 08–24: allocation of wins and losses
09       $j = n$
10       while $(w_i > 0) \vee (M_{ij} \neq 1) \vee (j > 0) \vee (i \neq j) \vee (l_j > 0) ==$ TRUE
11             $\mathcal{M}_{ij} = 3$
12             $w_i = w_i - 1$
13             $l_j = l_j - 1$
14             $j = j - 1$
15       if $w_i > 0$                                  // line 15–17: $s$ is undecided
16          $L = 2$
17          return $L$, $\mathcal{M}$
18 $L = 1$                                // line 18–19: $s$ is a football sequence
19 return $L$, $\mathcal{M}$
```

R2 uses a special version of Havel-Hakimi algorithm called HAVEL-HAKIMI-DRAWS (or shortly HHD). While for the classical Havel-Hakimi algorithm the equal scores are equivalent, in this application we have to distinguish them.

Additional parameters are $d = (d_1, \ldots, d_n)$: a draw sequence produced by DRAW-REC; $\mathcal{M}$: $n \times n$ sized matrix where $\mathcal{M}_{ij}$ is the number of points received by T_i in the match with T_j; $\mathcal{E} = (E_1, \ldots, E_n) = ((e_1, h_1), \ldots, (e_n, h_n))$: current extended and sorted version of d; $H = (h_1, \ldots, h_n)$: h_i is the index of e_i in d; n_l: lower index of the essential part of $\mathcal{E}$; n_u: upper index of the essential part of $\mathcal{E}$; $c = (c_0, \ldots, c_n)$: c_i is the number of i's among $e_{n_l}, \ldots, e_{n_u}$; $C = (C_0, \ldots, C_n)$: C_i is the cumulated number of i's among $e_{n_l}, \ldots, e_{n_u}$.

```
HAVEL-HAKIMI-DRAWS($n, d, \mathcal{M}$)
01 $n_l = 1$                        // line 01–05: initialization of $n_l$, $n_u$, and $\mathcal{E}$;
02 $n_u = n$
03 for $i = n_l$ to $n_u$                   // line 03–07: initialization of G and $n_u$;
04       $e_i = d_i$
05       $h_i = i$
07 $n_u = n$
08 for $i = 1$ to $n$                           // line 08–15: pairing of the draws;
09       COUNTING-SORT-DRAWS($n, i, n_u, \mathcal{E}$)               // line 09: sorting
```

```
10       if $e_i = 0$
11          return $\mathcal{M}$
12       for $k = 1$ to $e_i$
13          $\mathcal{M}_{h_i,h_i+k} = \mathcal{M}_{h_i+k,h_i} = 1$            // line 13: a draw is fixed
14          $e_{i+k} = e_{i+k} + 1$
15       while $n_u == 0$
16             $n_u = n_u - h_i$
17 return $\mathcal{M}$        // line 17: return the matrix containing the paired draws
```

Counting-Sort-Draws is a modified version of the well-known linear time sorting algorithm Counting-Sort [27].

Additional parameters are $d = (d_1, \ldots, d_n)$: a draw sequence produced by Draw-Uniform-Rec; n_l: lower index of the essential part of $\mathcal{E}$; n_u: upper index of the essential part of $\mathcal{E}$; $\mathcal{M}$: $n \times n$ sized matrix where $\mathcal{M}_{ij}$ is the number of points received by T_i in the match with T_j; $\mathcal{E} = (E_1, \ldots, E_n) = ((g_{11}, g_{12}), \ldots, (g_{1n}, g_{2n})$: current extended and sorted version of d with the corresponding indices; $\mathcal{G}$: the working version of $\mathcal{E}$; n_l : lower index of the essential part of $\mathcal{E}$; n_u: upper index of the essential part of $\mathcal{E}$; $c = (c_0, \ldots, c_{n-1})$: c_i is the number of i's among $g_{1,n_l}, \ldots, g_{1,n_u}$; $C_n = 0$ working variable; $C = (C_0, \ldots, C_{n-1})$: C_i is the number of investigated scores larger or equal with i.

```
Counting-Sort-Draws($n, d, n_l, n_u, \mathcal{E}$)
01 for $i = n_l$ to $n_u$                    // line 01–05: initialization of G and $c$;
02       $g_{1,i} = e_{1,i}$
03       $g_{2,i} = e_{2,i}$
04 for $i = 0$ to $n - 1$
05       $c_i = 0$
06 for $i = n_l$ to $n_u$                 // line 06-10: computation of the counters
07       $c_{g_{1i}} = c_{g_{1i}} + 1$
08 $C_n = 1$
09 for $n - 1$ downto $0$
10       $C_i = C_{i+1} + c_i$
11 for $i = n_l$ to $n_u$                   // line 11-16: computation of the new $\mathcal{E}$
12       $x = C_{g_{1,i}} + 1$
13       $e_{1,x} = g_{1,i}$
14       $e_{2,x} = g_{2,i}$
15       $C_x = C_x + 1$
16 return $\mathcal{E}$
```

The running time of Counting-Sort-Draw is $\Theta(n)$, of Havel-Hakimi-Draw is $O(n^2)$ and the one of Draw-Uniform-Rec is also $O(n^2)$.

As an example let $s = (1, 1, 7, 7)$. Then s has a unique draw sequence $(1, 1, 1, 1)$ and unique sport matrix shown in Table 8.

i	w_i	d_i	l_i	s_i
1	0	1	2	1
2	0	1	2	1
3	2	1	0	7
4	2	1	0	7

Table 8: Unique sport matrix belonging to the sequence $s = (1, 1, 7, 7)$.

According to the relatively quick version HAVEL-HAKIMI-SHIFTING [62] T_1 plays a draw with T_4 and T_2 with T_3 resulting the partial result matrix shown in Table 9.

i	T_1	T_2	T_3	T_4	s_i
1	—	?	?	1	1
2	?	—	1	?	1
3	?	1	—	?	7
4	1	?	?	—	7

Table 9: Partial result matrix belonging to the draws of $s = (1, 1, 7, 7)$.

The partial result matrix containing the draws in Table 9 is not reconstructible since no acceptable result for the match between T_1 and T_2.

If we use the classical Havel-Hakimi algorithm then the draws are between T_1 and T_2, further between T_3 and T_4 and our greedy algorithm DRAW-UNIFORM-REC reconstructs the received partial result matrix.

Another example let $s = (1, 1, 8, 8, 10, 13)$. Then s has a unique draw sequence $(1, 1, 2, 2, 1, 1)$ and a unique sport matrix shown in Table 10.

i	w_i	d_i	l_i	s_i
1	0	1	4	1
2	0	1	4	1
3	2	2	1	8
4	2	2	1	8
5	3	1	1	10
6	4	1	0	13

Table 10: Unique sport matrix belonging to the sequence $s = (1, 1, 8, 8, 10, 13)$.

In this case at first $\mathcal{E} = ((2,3),(2,4),(1,1),(1,2),(1,5),(1,6))$. The draws allocated by HHD are shown in Table 11.

i	T_1	T_2	T_3	T_4	T_5	T_6	s_i
1	—	?	1	?	?	?	1
2	?	—	?	1	?	?	1
3	1	?	—	1	?	?	8
4	?	1	1	—	?	?	8
5	?	?	?	?	—	1	10
6	?	?	?	?	1	—	13

Table 11: Partial result matrix belonging to the draws of $s = (1,1,8,8,10,13)$.

The partial result matrix in Table 11 is not reconstructible since no acceptable result for the match between T_1 and T_2.

5.3 Reconstruction algorithm R3 = Draw-Inner-Rec

Reconstruction algorithm R3 = Draw-Inner-Rec is an improved version of R2: it takes into account the obligatory inner draws.

The base of Inner-Draws is the following lemma.

Lemma 39 *If* $n \geq 1$, $f = (f_1, \ldots, f_n)$ *is a football sequence,* $1 \leq k \leq n$ *and*

$$\sum_{i=1}^{k} f_i < 3\binom{k}{2}, \tag{35}$$

then among the teams T_1, ..., T_k *there are at least*

$$\left\lceil \left(3\binom{k}{2} - \sum_{i=1}^{k} f_i\right) /2 \right\rceil \tag{36}$$

draws.

Proof. If

$$\left\lceil 2\left(3\binom{k}{2} - \sum_{i=1}^{k} f_i\right) /2 \right\rceil = q > 0, \tag{37}$$

then the first k teams lost at least q points due to inner draws (or even more, if they gathered points in the matches against the remaining teams). □

Trying to reconstruct the sequence $s = (1, 1, 8, 8, 10, 13)$ which was the last example of the previous Section 5.2 Draw-Inner-Rec (see Table 10 and 11) recognizes that $s_1 + s_2 = 2$ therefore according to Lemma 39 the obligatory result between T_1 and T_2 is a draw. Then Draw-Inner-Rec finishes the allocation of the draws as it is shown in Table 12.

1	–	1	?	?	?	?	1
2	1	–	?	?	?	?	1
3	?	?	–	1	1	?	8
4	?	?	1	–	?	1	8
5	?	?	1	?	–	?	10
6	?	?	?	1	1?	–	13

Table 12: Partial result matrix belonging to the draws of $s = (1, 1, 8, 8, 10, 13)$ allocated by Draw-Inner-Rec.

Using the matrix of the allocated draws shown in Table 12 Draw-Uniform-Rec produces the complete result matrix shown in Table 13. proving that $s = (1, 1, 8, 8, 10, 13)$ is a football sequence.

i	T_1	T_2	T_3	T_4	T_5	T_6	s_i
i	T_1	T_2	T_3	T_4	T_5	T_6	s_i
1	–	1	0	0	0	0	1
2	1	–	0	0	0	0	1
3	3	3	–	1	1	0	8
4	3	3	1	–	0	1	8
5	3	3	1	3	–	0	10
6	3	3	3	1	3	–	13

Table 13: Partial result matrix belonging to the draws of $s = (1, 1, 8, 8, 10, 13)$ allocated by Draw-Inner-Rec.

The algorithm based on this lemma yet is is not implemented.

6 Enumeration of football sequences

There are many publications connected with the generation [5, 52, 58, 100] and enumeration of degree sequences of graphs, e.g. [4, 5, 8, 21, 26, 49, 62, 63, 67, 72, 78, 79, 81, 87, 92, 97, 98, 105, 117]. The problems connected with directed graphs sometimes are considered as problems of orientation of undirected graphs [37, 36, 38, 39].

The enumeration of degree [4, 21, 40, 62, 63] and score [49, 53] sequences also has a reach literature.

The first published enumeration results connected with football score sequences belong to Gábor Kovács, Norbert Pataki, Zoltán Hernyák and Tamás Hegyessy [73] who computed $F(n)$ for $n = 1, \ldots, 8$ in 2002. N. J. A. Sloane in May 2007 determined $F(9)$, then in June 2008 Min Li computed $F(10)$. The newest results were received by J. E. Schoenfield who computed $F(11)$ in September of 2008 and $F(12)$ in December of 2008 [100].

Connected problems are the listing of all degree sequences and sampling of degree sequences [11, 15, 28, 65, 66, 82].

Our basic method is similar as we enumerated the degree sequences of simple graphs [62, 103].

¿From one side we try to test the elements of the possible smallest set, and from the other side we try to use quick as possible testing and reconstruction algorithms.

A natural idea is to investigate only the nonincreasing sequences of integers having 0 as lower bound and $3(n-1)$ as upper bound. Paul Erdős and Tibor Gallai called such sequences *regular* [32]. The number of such sequences is given by (1).

6.1 Decreasing of the number of the investigated sequences

A useful tool of the enumeration of the number of football sequences is the decreasing of the number of the considered sequences.

In Section 4 we proposed and analyzed filtering of regular sequences with constant, linear and quadratic time algorithms. For 14 teams we excluded more then the half of the regular sequences by the constant time algorithms. For 13 teams the linear and quadratic algorithms left less then 10.58 percent of the regular sequences. In Section 5 the polynomial reconstruction algorithms decreased the fraction of the undecided regular sequences to 4.68 percent of the regular sequences.

6.2 Backtrack filtering and accepting test

This method is due to Antal Iványi [54, 73].

The results of the filtering algorithms are summarized in Table 14.

The running time of the filtering algorithms are presented in Table 15. The times are cumulated and contain the time necessary for the generation of the sequences too.

n	CONSTANT	LINEAR	QUAD	BACKTRACK = F
1	**1**	**1**	**1**	**1**
2	**2**	**2**	**2**	**2**
3	14	**7**	**7**	**7**
4	203	**40**	**40**	**40**
5	2 133	365	**355**	**355**
6	20 518	4 086	3 760	**3 678**
7	191 707	44 657	39 417	27 263
8	1 772 442	451 213	393 072	361 058
9	16 332 091	4 348 655	3 804 485	3 403 613
10	150 288 309	41 166 157	36 302 148	31 653 777
11	1 383 099467	387 416 935	344 012 885	292 547 199
12	12 737 278 674	3 633 749 149	3 246 651 763	2 696 619 716
13	117 411 154 292	33 821 636 274	30 405 902 165	
14	1 083 421 567 482			

Table 14: Numbers of sequences accepted by constant, linear and quadratic time and BACKTRACK filtering algorithms for $n = 1, \ldots, 14$ teams.

n	CONSTANT	LINEAR	QUAD	BACKTRACK = F
1	0.000	0.000	0.000	0.000
2	0.000	0.000	0.000	0.000
3	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.015
7	0.016	0.031	0.042	0.172
8	0.046	0.375	0.577	52.603
9	0.468	3.572	5.772	
10	4.134	34.632	54.741	
11	37.612	329.816	525.752	
12	343.575	3 145.494	4 998.831	
13	3 142.469	30 541.260	49 035.625	
14	29 438.094			

Table 15: Running times of constant, linear and quadratic time filtering algorithms for $n = 1, \ldots, 14$ teams.

The individual results of the reconstruction algorithms are summarized in Table 16.

The running times of the reconstruction algorithms are shown in Table 17.

n	R1	R2 + R3	Backtrack	F
1	1	0	0	1
2	2	0	0	2
3	6	1	0	7
4	18	22	0	40
5	50	305	0	355
6	137	3 460	81	3 678
7	375	33 993	2 895	37 263
8	1 023	304 349	56 909	361 058
9	2 776	2 576 124		3 403 613
10	7 498	21 453 751		31 653 777
11	20 177	177 819 555		292 547 199
12	54 127	1 476 661 425		2 696 619 716
13	144 708	12 300 060 430		

Table 16: Number of $(0, 3n-3, n)$-regular sequences reconstructed by reconstruction algorithms R1, R2 + R3 and Backtrack for $n = 1, \ldots, 14$ teams.

n	R1	R3	Backtrack
2	0.000	0.000	0.000
3	0.000	0.000	0.000
4	0.000	0.000	0.000
5	0.000	0.000	0.000
6	0.000	0.015	0.015
7	0.063	0.109	0.172
8	0.546	1.264	52.603
9	5.491	15.226	
10	53.880	179.249	
11	522.386	2 066.323	
12	4 998.831	23 429.877	
13	49 035.625	261 904.750	

Table 17: Running times of the R1, R3 and Backtrack reconstructing algorithms for $n = 1, \ldots, 13$ teams.

6.3 Recursive accepting test

This method is due to Schoenfield [100]. According to this method we compare the sequences of length n passed through the filtering and accepting tests with the good sequences of length $n-1$ whether they can be derive from them.

Since if we omit a team with its results from a football matrix of size $n \times n$, then we get a football matrix of size $(n-1) \times (n-1)$, therefore we regularly delete the *first* elements of the investigated n-length sequences.

Let $n \geq 2$. We suppose that when we enumerate the n-length good sequences then we know the $F(n-1) \times (n-1)$ sized matrix M containing the $(n-1)$-length good sequences in lexicographically increasing order, and also know the vector $(P_0, \ldots, P_k)$, where $k = \lfloor 3(n-1)/2 \rfloor$ and P_i gives the number of $(n-1)$-length good sequences starting with i.

Let start the recursion with $n = 2$. Matrix M_1 contains only one row (0) and P contains one element $P(1) = 1$.

The constant time filtering algorithms accept only the sequences $(0,3)$ and $(1,1)$. At first we omit 0 from the first sequence and state that the remaining sequence (3) can be derived from (0) only if the team having zero points in the shorter sequence wins against the omitted player. So the omitted player has to have zero points. Since the omitted score is exactly zero, (0,3) *is a good sequence.*

Then we delete the first element from the sequence (1,1) and state that the player having zero points has to play a draw with the omitted team. Since it has exactly one point, therefore (1,1) is also *a good sequence* and so $F(2) = 2$.

Now let $n = 3$. Then M_2 contains two rows: (0,3) and (1,1). In this case the filtering algorithms accept only the seven good sequences: (0,3,6), (0,4,4), (1,1,4), (1,2,4), (1,3,4), (2,2,2) and (3,3,3).

At first we delete 0 from $(0,3,6)$ and compare the remaining $(3,6)$ with the known good sequences. There are thee possibilities: the first team of the good sequence received 3, 1 or 0 points against the omitted one. If 3, then the good sequence has to start with 0. There is only one sequence $(0,3)$ requiring two losses for the omitted team. Since the omitted element is exactly zero, $(0,3,6)$ *is a good sequence.*

The second accepted sequence is $(0,4,4)$. Omitting 0 and comparing $(4,4)$ with the good sequences we get, that $(1,1)$ is the only potential ancestor requiring zero points for the deleted team. Since it has exactly zero points, $(0,4,4)$ is also a *good sequence.*

In a similar way we can prove that the remaining five accepted sequences are also good.

When $n = 4$ then M contains seven elements and $P = (1, 3, 6, 7)$.

Reconstruct executes this recursive step. Its additional parameters are $F(n-1)$: the number of $(n-1)$-length good sequences; $\mathcal{M}_{F(n-1)\times(n-1)}$: matrix of good sequences of length $n-1$ (this matrix consists of submatrices containing the good sequences having identical first element; $P = (P_0, \ldots, P_k)$, where $k =$ $k = \lfloor 3(n-1)/2 \rfloor$and P_i is the number of $n-1$ length football sequences starting with i; $\mathcal{N}_{F(n)\times n}$: matrix of good sequences of length n; $m = (m_1, \ldots, m_{n-1})$: the current reduced version of s; d: the current score of the deleted team.

```
Reconstruct($n, s, F, M, P$)
01 $L = 1$ line 01–02: initialization of $L$ and $u$
02 $u = \lfloor 3(n-1)/2 \rfloor$
03 if $s_2 \le u$                    // line 03-21: omitted element starts with a loss
04     $j \leftarrow P_{s_2}$
05     while $\mathcal{M}_{j,1} == s_2$
06         $d \leftarrow 0$
07         $k \leftarrow 2$
08         while $k \le n$
09             if $s_k - \mathcal{M}_{j,k} == 3$
10                 $d = d + 0$
11                 go to 19
12             if $s_k - \mathcal{M}_{j,k} == 1$
13                 $d = d + 1$
14                 go to 19
15             if $s_k - \mathcal{M}_{j,k} == 0$
16                 $d = d + 3$
17                 go to 19
18             go to 22
19             $k \leftarrow k + 1$
20     if $d == s_1$
21         return $L$
22 if $0 \le s_2 - 1$                 // line 22-40: omitted element starts with a draw
23     $j \leftarrow P_{s_2-1}$
24     while $\mathcal{M}_{j,1} == s_2 - 1$
25         $d \leftarrow 1$
26         $k \leftarrow 2$
27         while $k \le n$
28             if $s_k - \mathcal{M}_{j,k} == 1$
29                 $d = d + 1$
```

```
30                  go to 38
31                if $s_k - \mathcal{M}_{j,k} == 1$
32                  $d = d + 1$
33                  go to 38
34                if $s_k - \mathcal{M}_{j,k} == 1$
35                  $d = d + 1$
36                  go to 38
37                go to 39
38                $k = k + 1$
39    if $d == s_1$
40      return L
41 if $0 \leq s_2 - 3$                 // line 41-59: omitted element starts with a win
42       $j \leftarrow P[s_2 - 3]$
43    while $\mathcal{M}_{j,1} == s_2 - 3$
44          $d \leftarrow 3$
45          $k \leftarrow 2$
46          while $k \leq n$
47                if $s_k - \mathcal{M}_{j,k} == 3$
48                  $d = d + 3$
49                  go to 57
50                if $s_k - \mathcal{M}_{j,k} == 1$
51                  $d = d + 1$
52                  go to 57
53                if $s_k - \mathcal{M}_{j,k} == 1$
54                  $d = d + 1$
55                  go to 57
56                go to 58
57                $k \leftarrow k + 1$
58    if $d == s_1$
59      return L
60 $L = 0$
61 return L
```

Table 18 shows the number of regular sequences ($R(n)$, the number of football sequences ($F(n)$, the ratio ($R(n+1)/R(n)$), the ratio $F(n+1)/F(n)$, and the ratio ($F(n)/R(n)$ for $n = 1, \ldots, 12$. In this table if $n \geq 2$ then $R(n)$ is decreasing.

Lemma 40 *If* n *tends to infinity then* $R(n+1)/R(n)$ *tends to* $256/27$.

n	$R(n)$	$\frac{R(n+1)}{R(n)}$	$F(n)$	$\frac{F(n+1)}{F(n)}$	$\frac{F(n)}{R(n)}$
1	1	10.000	1	2.000	1.0000
2	1	8.400	2	3.500	0.2000
3	84	8.512	7	5.714	0.0833
4	715	8.655	40	8.875	0.0559
5	6188	8.769	355	10.361	0.0574
6	54264	8.859	3678	10.131	0.0678
7	480700	8.929	37263	9.689	0.0775
8	4292145	8.986	361058	9.427	0.0841
9	38567100	9.032	3403613	9.300	0.0883
10	348330136	9.070	31653777	9.242	0.0909
11	3159461968	9.103	292547199	9.217	0.0926
12	28760021745	9.131	2696619716		0.0938
13	262596783864	9.155			
14	240397990420				

Table 18: Number of regular and football sequences and the ratio of these numbers for neighboring numbers of teams

Proof. According to (1)

$$\frac{R(n+1)}{R(n)} = \frac{(4n+1)(4n)(4n-1)(4n-2)}{(n+1)(3n)(3n-1)(3n-2)} = \frac{256}{27} + o(1), \tag{38}$$

implying the required limit. □

If $n \geq 1$ then in Table 18 $F(n+1)/F(n)$ is nondecreasing. We suppose that it tends to 1.

If $5 \leq n \leq 12$ then $F(n)/R(n)$ is increasing. It is easy to see that

$$\lim_{n\to\infty} \frac{F(n+1)}{F(n)} \leq \frac{R(n+1)}{R(n)}. \tag{39}$$

The behavior of $F(n)/R(n)$ is a bit surprising since the similar relative density of tournaments score sequences tends to zero (see [21]). We suppose that $F(n)/R(n)$ also tends to zero but the convergence is slow.

Acknowledgements. The authors thank Péter Burcsi (Eötvös Loránd University) for many useful comments, András Pluhár (University of Szeged) and Zoltán Király (Eötvös Loránd University) for the recommended references, Loránd Lucz (Eötvös Loránd University) and Tamás Iványi (Economical Politechnicum) for the computer experiments. The European Union and the European Social Fund have provided financial support to the project under the grant agreement no. TÁMOP 4.2.1/B-09/1/KMR-2010-0003.

References

[1] M. Anholcer, V. Babiy, S. Bozóki, W. W. Koczkodaj, A simplified implementation of the least squares solution for pairwise comparisons matrices. *CEJOR Cent. Eur. J. Oper. Res.* **19,** 4 (2011) 439–444. ⇒131

[2] S. R. Arikati, A. Maheshwari, Realizing degree sequences in parallel, *SIAM J. Discrete Math.* **9,** 2 (1996) 317–338. ⇒131

[3] Ch. M. Bang, H. Sharp, Jr., Score vectors of tournaments. *J.* Combin. *Theory Ser. B* **26,** 1 (1979) 81–84. ⇒131

[4] T. M. Barnes, C. D. Savage, A recurrence for counting graphical partitions, *Electron. J. Combin.* **2** (1995), Research Paper 11, 10 pages (electronic). ⇒169, 170

[5] T. M. Barnes, C. D. Savage, Efficient generation of graphical partitions, *Discrete Appl. Math.* **78,** 1–3 (1997) 17–26. ⇒169

[6] M. D. Barrus, Havel-Hakimi residues of unigraphs, *Inf. Proc. Letters* **112** (2012) 44–48. ⇒148

[7] A. Bege, Personal communication, Visegrád, 1999. ⇒133

[8] S. Bereg, H. Ito, Transforming graphs with the same degree sequence, in: *Kyoto Int. Conf. on Computational Geometry and Graph Theory,* (ed. H. Ito et al.) LNCS **4535** Springer-Verlag, Berlin, Heidelberg. 2008. pp. 25–32. ⇒169

[9] A. Berger, *Directed degree sequences*, PhD Dissertation, Martin-Luther-Universität Halle-Wittenberg, 2011. ⇒134

[10] A. Berger, A note on the characterization of digraph sequences, *arXiv*, arXiv:1112.1215v1 [math.CO] (6 December 2011) ⇒133, 134, 159

[11] A. Berger, M. Müller-Hannemann, Uniform sampling of digraphs with a fixed degree sequence, in (ed. D. M. Thilikos) *36th Int. Workshop on Graph Theoretic Concepts in Computer Science* (June 28 - 30, 2010, Zarós, Crete, Greece), LNCS **6410** (2010) 220–231. ⇒170

[12] A. Berger, M. Müller-Hannemann, Dag realizations of directed degree sequences, in (ed. O. M. Steffen, J. A. Telle) *Proc. 18th FCT* LNCS **6914** (2011) 264–275. Full version with proofs: Technical Report 2011/5 of University Halle-Wittenberg, Institute of Computer Science. ⇒134

[13] A. Berger, M. Müller-Hannemann, Dag characterizations of directed degree sequences, i Technical Report 2011/6 of University Halle-Wittenberg, Institute of Computer Science. ⇒134

[14] A. Berger, M. Műller-Hannemann, How to attack the NP-complete dag realization problems in practice.
arXiv, arXiv:1203.36v1, 2012. http://arxiv.org/abs/1203.3636 ⇒134

[15] J. K. Blitzstein, P. Diaconis, A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet Mathematics* **6,** 4 (2011) 489–522. ⇒170

[16] S. Bozóki, J. Fülöp, A. Poesz, On pairwise comparison matrices that can be made consistent by the modification of a few elements. *CEJOR Cent. Eur. J. Oper. Res.* **19** (2011) 157–175. ⇒131

[17] Bozóki S., J. Fülöp, L. Rónyai, On optimal completion of incomplete pairwise comparison matrices, *Math. Comput. Modelling* **52** (2010) 318–333. ⇒131

[18] A. Brauer, I. C. Gentry, K. Shaw, A new proof of a theorem by H. G. Landau on tournament matrices. *J. Comb. Theory* **5** (1968) 289–292. ⇒131

[19] A. R. Brualdi, K. Kiernan, Landau's and Rado's theorems and partial tournaments, *Electron. J. Combin.* **16,** (#N2) (2009) 6 pages. ⇒131

[20] A. R. Brualdi, J. Shen, Landau's inequalities for tournament scores and a short proof of a theorem on transitive sub-tournaments, *J. Graph Theory* **38,** 4 (2001) 244–254. ⇒131

[21] J. M. Burns: *The number of degree sequences of graphs* PhD Dissertation, MIT, 2007. ⇒169, 170, 176

[22] A. N. Busch, G. Chen, M. S. Jacobson, Transitive partitions in realizations of tournament score sequences, *J. Graph Theory* **64,** 1 (2010), 52–62. ⇒131

[23] W. Chen, On the realization of a (p,s)-digraph with prescribed degrees, *J. Franklin Institute* **281,** (5) 406–422. ⇒133

[24] S. A. Choudum, A simple proof of the Erdős-Gallai theorem on graph sequences, *Bull. Austral. Math. Soc.* **33** (1986) 67–70. ⇒134, 151

[25] V. Chungphaisan, Conditions for sequences to be r-graphic. *Discrete Math.* **7** (1974) 31–39. ⇒134

[26] J. Cooper, L. Lu, Graphs with asymptotically invariant degree sequences under restriction, *Internet Mathematics* **7,** 1 (2011) 67–80. ⇒169

[27] T. H. Cormen, Ch. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms* Third edition, The MIT Press/McGraw Hill, Cambridge/New York, 2009. ⇒135, 156, 166

[28] C. I. Del Genio, H. Kim, Z. Toroczkai, K. E. Bassler, Efficient and exact sampling of simple graphs with given arbitrary degree sequence, *PLoS ONE* **5,** 4 e10012 (2010). ⇒170

[29] A. Dessmark, A. Lingas, O. Garrido, On parallel complexity of maximum f-matching and the degree sequence problem. *Mathematical Foundations of Computer Science 1994* (Košice, 1994), LNCS **841,** Springer, Berlin, 1994, 316–325. ⇒131

[30] R. B. Eggleton, Graphic sequences and graphic polynomials: a report, in *Colloq. Math. Soc. J. Bolyai* **10,** North Holland, Amsterdam, 1975, 385–392. ⇒134

[31] R. B. Eggleton, D. A. Holton, Graphic sequences. *Lecture Notes in Mathematics* **10**, Springer Verlag, Berlin, 1979, 1–10. ⇒134

[32] P. Erdős, T. Gallai, Graphs with vertices having prescribed degrees (Hungarian), *Mat. Lapok* **11** (1960) 264–274. ⇒134, 151, 170

[33] P. L. Erdős, I. Miklós, Z. Toroczkai, A simple Havel-Hakimi type algorithm to realize graphical degree sequences of directed graphs. *Electronic J. Combin.* **17,** 1 R66 (2011). ⇒134, 159

[34] P. Erdős, L. B. Richmond, On graphical partitions, *Combinatorica* **13,** 1 (1993) 57–63. ⇒134

[35] L. R. Ford, D. R. Fulkerson, *Flows in Networks.* Princeton University, Press, Princeton, 1962. ⇒134

[36] A. Frank, *Connections in Combinatorial Optimization,* Oxford University Press, Oxford, 2011. ⇒130, 169

[37] A. Frank, On the orientation of graphs. *J. Combin. Theory Ser. B* **28,** 3 (1980) 251–261. ⇒169

[38] A. Frank, A. Gyárfás, How to orient the edges of a graph? In *Combinatorics. Vol. 1* (ed. A. Hajnal and V. T. Sós), North-Holland, Amsterdam-New York, 1978. pp. 353–364. ⇒169

[39] A. Frank, T. Király, Z. Király, On the orientation of graphs and hypergraphs. *Discrete Appl. Math.*, **131,** 2 (2003) 385–400. ⇒169

[40] D. A. Frank, C. D. Savage, J. A. Sellers, On the number of graphical forest partitions, *Ars Combin.* **65** (2002) 33–37. ⇒170

[41] D. R. Fulkerson, Zero-one matrices with zero trace, *Pacific J. Math* **10** (1960) 831–836. ⇒133

[42] D. Gale, A theorem on flows in networks, *Pacific J. Math.* **7** (1957) 1073–1082. ⇒133

[43] A. Garg, A. Goel, A. Tripathi, Constructive extensions of two results on graphs sequences. *Discrete Appl. Math.* **159,** 17 (2011) 2170–2174. ⇒134

[44] J. Griggs, K. B. Reid, Landau's theorem revisited, *Australas. J. Comb.* **20** (1999), 19–24. ⇒131

[45] J. L. Gross, J. Yellen, *Handbook of Graph Theory,* CRC Press, Boca Raton, 2004. ⇒130, 131

[46] B. Guiduli, A. Gyárfás, S. Thomassé, P. Weidl, 2-partition-transitive tournaments. *J.* Combin. *Theory Ser. B* **72**, 2 (1998) 181–196. ⇒131, 134

[47] S. L. Hakimi, On the realizability of a set of integers as degrees of the vertices of a simple graph. *J. SIAM Appl. Math.* **10** (1962) 496–506. ⇒134, 151

[48] S. L. Hakimi, On the degrees of the vertices of a graph, *F. Franklin Institute,* **279,** 4 (1965) 290–308. ⇒134

[49] H. Harborth, A. Kemnitz, Eine Anzahl der Fussballtabelle. *Math. Semester.* **29** (1982) 258–263. ⇒169, 170

[50] V. Havel, A remark on the existence of finite graphs (Czech), *Časopis Pěst. Mat.* **80** (1955) 477–480. ⇒134, 151

[51] P. Hell, D. Kirkpatrick, Linear-time certifying algorithms for near-graphical sequences. *Discrete Math.* **309,** 18 (2009) 5703–5713. ⇒134, 151

[52] R. Hemasinha, An algorithm to generate tournament score sequences, *Math. Comp. Modelling* **37,** 3–4 (2003) 377–382. ⇒169

[53] G. Isaak, Tournaments and score sequences, in ed. by D. B. West *REGS in Combinatorics*, 2010, No. 7, http://www.math.uiuc.edu/ west/regs/fifa.html ⇒130, 170

[54] A. Iványi, Testing of football score sequences (Hungarian), in: *Abstracts of 25th Hungarian Conf. on Operation Research* (Debrecen, October 17–20, 2001), MOT, Budapest, 2001, 53–53. ⇒130, 170

[55] A. Iványi, Maximal tournaments. *Pure Math. Appl.* **13,** 1–2 (2002) 171–183. ⇒ 131, 133

[56] A. Iványi, Reconstruction of complete interval tournaments, *Acta Univ. Sapientiae, Inform.*, **1,** 1 (2009) 71–88. ⇒130, 131, 132, 134, 142

[57] A. Iványi, Reconstruction of complete interval tournaments. II, *Acta Univ. Sapientiae, Math.* **2,** 1 (2010) 47–71. ⇒130, 131, 134

[58] A. Iványi, Directed graphs with prescribed score sequences (ed by S. Iwata), *The 7th Hungarian-Japanese Symposium on Discrete Mathematics and Applications* (Kyoto, May 31 - June 3, 2011, ed by S. Iwata), 114–123. ⇒169

[59] A. Iványi, Deciding the validity of the score sequence of a soccer tournament, in (ed. by A. Frank): *Open problems of the Egerváry Research Group,* Budapest, 2012. http://lemon.cs.elte.hu/egres/open/ ⇒130

[60] A. Iványi, Degree sequences of multigraphs. *Annales Univ. Sci. Budapest., Sect. Comp.* **37** (2012), 195–214. ⇒148, 156

[61] A. Iványi, L. Lucz, Degree sequences of multigraphs (Hungarian), *Alkalm. Mat. Lapok* **29** (2012) (to appear). ⇒134

[62] A. Iványi, L. Lucz, T. F. Móri, P. Sótér, On the Erdős-Gallai and Havel-Hakimi algorithms. *Acta Univ. Sapientiae, Inform.* **3,** 2 (2011) 230–268. ⇒ 151, 167, 169, 170

[63] A. Iványi, L. Lucz, T. F. Móri, P. Sótér, The number of degree-vectors for simple graphs, in: ed. by N. J. A. Sloane, *The On-Line Encyclopedia of Integer Sequences,* 2011. http://oeis.org/A004251 ⇒169, 170

[64] A. Iványi, S. Pirzada, Comparison based ranking, in: ed. A. Iványi, *Algorithms of Informatics, Vol. 3,* AnTonCom, Budapest 2011, 1209–1258. ⇒132

[65] R. Kannan, P. Tetali, S. Vempala, Simple Markovian-chain algorithms for generating bipartite graphs and tournaments. *Random Struct. Algorithms* **14,** 4 (1999) 293–308. ⇒170

[66] K. Kayibi, M. A. Khan, S. Pirzada, A. Iványi, Random sampling of minimally cyclic digraphs with given imbalance sequence. *Acta Univ. Sapientiae, Math.* (submitted). ⇒170

[67] A. Kemnitz, S. Dolff, Score sequences of multitournaments. *Congr. Numer.* **127** (1997) 85–95. ⇒169

[68] G. Kéri, On qualitatively consistent, transitive and contradictory judgment matrices emerging from multiattribute decision procedures, *CEJOR Cent. Eur. J. Oper. Res.* **19,** 2 (2011) 215–224. ⇒131

[69] H. Kim, Z. Toroczkai, I. Miklós, P. L. Erdős, L. A. Székely, Degree-based graph construction, *J. Physics*: *Math. Theor. A* **42** 39 (2009), 392001-1-3920001.10. ⇒ 134

[70] Z. Király, Recognizing graphic degree sequences and generating all realizations, Technical Report of Egerváry Research Group, TR-2011-11, Budapest. Last modification 23 April, 2012. `http://www.cs.elte.hu/egres/` ⇒148, 156

[71] Z. Király, *Data Structures* (Lecture notes in Hungarian), Eötvös Loránd University, Mathematical Institute, Budapest, 2012. `http://www.cs.elte.hu/~kiraly/Adatstrukturak.pdf` ⇒148, 156

[72] D. J. Kleitman, K. J. Winston, Forests and score vectors, *Combinatorica* **1,** 1 (1981) 49–54. ⇒169

[73] G. Zs. Kovács, N. Pataki, Analysis of ranking sequences (Hungarian), Scientific student paper, Eötvös Loránd University, Faculty of Sciences, Budapest 2002. ⇒170

[74] M. D. LaMar, Algorithms for realizing degree sequences of directed graphs. arXiv, (2010). http://arxiv.org/abs/0906.0343. ⇒134

[75] H. G. Landau, On dominance relations and the structure of animal societies. III. The condition for a score sequence, *Bull. Math. Biophys.* **15,** (1953) 143–148. ⇒ 131, 132, 134

[76] F. Liljeros, C. R. Edling, L. Amaral, H. E. Stanley, Y. Aberg, The web of human sexual contacts. *Nature* **411** (2001) 907–908. ⇒131

[77] L. Lucz, *Analysis of degree sequences of graphs* (Hungarian), MSc Thesis, Eötvös Loránd University, Faculty of Informatics, Budapest, 2012. http://people.inf.elte.hu/lulsaai/diploma ⇒149

[78] L. Lucz, A. Iványi, Testing and enumeration of football sequences, in: *MaCS'12. 9th Joint Conference in Mathematics and Computer Science* (Siófok, Hungary, February 9–12, 2012, ed. by Z. Csörnyei), ELTE IK, Budapest, 2012, 63–63. ⇒ 169

[79] B. D. McKay, X. Wang, Asymptotic enumeration of tournaments with a given score sequence. *J. Comb. Theory A*, **73,** 1 (1996) 77–90. ⇒169

[80] D. Meierling, L. Volkmann, A remark on degree sequences of multigraphs. *Math. Methods Oper. Res.* **69,** 2 (2009) 369–374. ⇒151

[81] N. Metropolis, P. R. Stein, The enumeration of graphical partitions, *European J. Comb.* **1,** 2 (1980) 139–153. ⇒169

[82] I. Miklós, P. L. Erdős, L. Soukup, Towards random uniform sampling of bipartite graphs with given degree sequence, *arXiv* 1004.2612v3 [math.CO] (14 Sep 2010), http://arxiv.org/pdf/1004.2612v3.pdf ⇒170

[83] J. W. Miller, Reduced criterion for degree sequences, *arXiv*, arXiv:1205.2686v1 [math.CO] 11 May 2012, 18 pages. ⇒134

[84] J. W. Moon, On the score sequence of an n-partite tournament. *Can. Math. Bull.* **5** (1962) 51–58. ⇒134

[85] J. W. Moon, An extension of Landau's theorem on tournaments, *Pacific J. Math.* **13** (1963), 1343–1345. ⇒132, 134

[86] J. W. Moon, *Topics on Tournaments.* Holt, Rinehart and Winston. New York, 1968. ⇒132

[87] T. V. Narayana, D. H. Bent, Computation of the number of score sequences in round-robin tournaments, *Canad. Math. Bull.* **7,** 1 (1964) 133–136. ⇒169

[88] M. Newman, A. L. Barabási, D. J. Watts, *The Structure and Dynamics of Networks.* Princeton University Press, (2006). ⇒131

[89] S. Özkan, Generalization of the Erdős-Gallai inequality. *Ars Combin.* **98** (2011) 295–302. ⇒151

[90] D. Pálvölgyi, Deciding soccer scores and partial orientations of graphs. *Acta Univ. Sapientiae,* Math. **1,** 1 (2009) 35–42. ⇒134

[91] A. N. Patrinos, S. L. Hakimi, Relations between graphs and integer-pair sequences. *Discrete Math.* **15** 4 (1976) 347–358 ⇒134

[92] G. Pécsy, L. Szűcs, Parallel verification and enumeration of tournaments, *Stud. Univ.* Babeş-Bolyai, *Inform.* **45,** 2 (2000) 11–26. ⇒131, 169

[93] S. Pirzada, *An Introduction to Graph Theory.* Orient BlackSwan, Hyderabad, 2012. ⇒130, 131

[94] S. Pirzada, G. Zhou, A. Iványi, Score lists of multipartite hypertournaments, *Acta Univ. Sapientiae, Inform.* **2,** 2 (2011) 184–193. ⇒134

[95] K. B. Reid, Tournaments: Scores, kings, generalizations and special topics, *Congr. Numer.* **115** (1996) 171–211. ⇒131

[96] K. B. Reid, C. Q. Zhang, Score sequences of semicomplete digraphs, *Bull. Inst. Combin. Appl.* **24** (1998) 27–32. ⇒133

[97] Ø. J. Rødseth, J. A. Sellers, H. Tverberg, Enumeration of the degree sequences of non-separable graphs and connected graphs. *European J. Comb.* **30,** 5 (2009) 1309–1317. ⇒169

[98] F. Ruskey, F. R. Cohen, P. Eades, A. Scott, Alley CATs in search of good homes. *Congr. Numer.* **102** (1994) 97–110. ⇒169

[99] H. J. Ryser, Combinatorial properties of matrices of zeroas and ones, *Canad. J. Math.* **9** (1957) 371–377. ⇒133

[100] J. E. Schoenfield, The number of football score sequences, in: ed. by N. J. A. Sloane, *The On-Line Encyclopedia of Integer Sequences,* 2012. http://oeis.org/A064626 ⇒130, 169, 170, 173

[101] G. Sierksma, H. Hoogeveen, Seven criteria for integer sequences being graphic, *J. Graph Theory* **15,** 2 (1991) 223–231. ⇒131

[102] B. Siklósi, *Comparison of sequential and parallel algorithms solving sport problems.* Master thesis. Eötvös Loránd University, Faculty of Sciences, Budapest, 2001. ⇒131

[103] N. J. A. Sloane, The number of degree-vectors for simple graphs. In (ed. N. J. A. Sloane): *The On-Line Encyclopedia of the Integer Sequences.* 2011. http://oeis.org/A004251 ⇒170

[104] D. Soroker, *Optimal parallel construction of prescribed tournaments, Discrete Appl. Math.* **29,** 1 (1990) 113–125. ⇒131

[105] R. P. Stanley, A zonotope associated with graphical degree sequence, in: *Applied Geometry and Discrete Mathematics, Festschr. 65th Birthday Victor Klee.* DIMACS Series in Discrete Mathematics and Theoretical Computer Science. **4** (1991) 555–570. ⇒169

[106] L. A. Székely, L. H. Clark, R. C. Entringer. An inequality for degree sequences. *Discrete Math.* **103,** 3 (1992) 293–300. ⇒131

[107] M. Takahashi, *Optimization Methods for Graphical Degree Sequence Problems and their Extensions*, PhD thesis, Graduate School of Information, Production and systems, Waseda University, Tokyo, 2007. http://hdl.handle.net/2065/28387 ⇒132, 151

[108] J. Temesi, Pairwise comparison matrices and the error-free property of the decision maker, *CEJOR Cent. Eur. J. Oper. Res.* **19,** 2 (2011) 239–249. ⇒ 131

[109] P. Tetali, A characterization of unique tournaments. *J. Combin. Theory Ser. B* **72,** 1 (1998) 157–159. ⇒148

[110] A. Tripathi, H. Tyagy, A simple criterion on degree sequences of graphs. *Discrete Appl. Math.* **156,** 18 (2008) 3513–3517. ⇒131, 151

[111] A. Tripathi, S. Vijay, A note on a theorem of Erdős & Gallai. *Discrete Math.* **265,** 1–3 (2003) 417–420. ⇒151

[112] A. Tripathi, S. Venugopalan, D. B. West, A short constructive proof of the Erdős-Gallai characterization of graphic lists. *Discrete Math.* **310,** 4 (2010) 833–834. ⇒151

[113] R. Tyskevich, Decomposition of graphical sequences and unigraphs, *Discrete Math.* **220,** 1–3 (2000) 201–238. ⇒148

[114] P. van Emde Boas, Preserving order in a forest in less than logarithmic time, *Proc. 16th Annual Symp. Found Comp. Sci.* **10** (1975) 75–84. ⇒148, 156

[115] E. W. Weisstein, *Degree Sequence,* ¿From MathWorld—Wolfram Web Resource, 2012. ⇒134

[116] E. W. Weisstein, *Graphic Sequence,* ¿From MathWorld—Wolfram Web Resource, 2012. ⇒134

[117] K. J. Winston, D. J. Kleitman, On the asymptotic number of tournament score sequences. *J. Combin. Theory Ser. A.* **35** (1983) 208–230. ⇒169

[118] I. E. Zverovich, V. E. Zverovich, Contribution to the theory of graphic sequences, *Discrete Math.* **105** (1992) 293–303. ⇒134

Received: March 5, 2012 • Revised: June 7, 2012

Acta Universitatis Sapientiae

The scientific journal of Sapientia Hungarian University of Transylvania publishes original papers and surveys in several areas of sciences written in English. Information about each series can be found at `http://www.acta.sapientia.ro`.

Acta Universitatis Sapientiae, Informatica

Each volume contains two issues.

Sapientia University

Scientia Publishing House

ISSN 1844-6086
http://www.acta.sapientia.ro

Information for authors

Acta Universitatis Sapientiae, Informatica publishes original papers and surveys in various fields of Computer Science. All papers are peer-reviewed.

Papers published in current and previous volumes can be found in Portable Document Format (pdf) form at the address: `http://www.acta.sapientia.ro`.

The submitted papers should not be considered for publication by other journals. The corresponding author is responsible for obtaining the permission of coauthors and of the authorities of institutes, if needed, for publication, the Editorial Board is disclaiming any responsibility.

Submission must be made by email (`acta-inf@acta.sapientia.ro`) only, using the LaTeX style and sample file at the address `http://www.acta.sapientia.ro`. Beside the LaTeX source a pdf format of the paper is needed too.

Prepare your paper carefully, including keywords, ACM Computing Classification System codes (`http://www.acm.org/about/class/1998`) and AMS Mathematics Subject Classification codes (`http://www.ams.org/msc/`).

References should be listed alphabetically based on the Intructions for Authors given at the address `http://www.acta.sapientia.ro`.

Illustrations should be given in Encapsulated Postscript (eps) format.

One issue is offered each author free of charge. No reprints will be available.

Contact address and subscription:
Acta Universitatis Sapientiae, Informatica
RO 400112 Cluj-Napoca
Str. Matei Corvin nr. 4.
Email: acta-inf@acta.sapientia.ro

Printed by Gloria Printing House
Director: Péter Nagy

ISSN 1844-6086
http://www.acta.sapientia.ro