PERGAMON

Computers & Operations Research 29 (2002) 13–31

computers & operations research

www.elsevier.com/locate/dsw

Finding all k-cliques in k-partite graphs, an application in textile engineering

Tore Grünert[a,*], Stefan Irnich[a], Hans-Jürgen Zimmermann[b], Markus Schneider[c], Burkhard Wulfhorst[b]

[a]*Lehr-und Forschungsgebiet Unternehmensforschung (Operations Research), Institut für Wirtschaftswissenschaften, Rheinisch-Westfälische Technische Hochschule Aachen, Templergraben 64, 52056, Aachen, Germany*

[b]*Lehrstuhl für Unternehmensforschung (Operations Research), Institut für Wirtschaftswissenschaften, Rheinisch-Westfälische Technische Hochschule Aachen, Templergraben 64, 52056, Aachen, Germany*

[c]*Institut für Textiltechnik, Rheinisch-Westfälische Technische Hochschule Aachen, Eilfschornsteinstr. 18, 52056 Aachen, Germany*

Received 1 April 1999; received in revised form 1 October 1999

Abstract

In many practical cases one has to choose an arrangement of different objects so that they are compatible. Whenever the compatibility of the objects can be checked by a pair-wise comparison the problem can be modelled using the graph–theoretic notion of cliques. We consider a special case of the problem where the objects can be grouped so that exactly one object in every group has to be chosen. This object has to be compatible to every other object selected from the other groups. The problem was motivated by a braiding application from textile technology. The task is to route a set of thread-spools (bobbins) on a machine from their origins to their destinations so that collisions are avoided. We present a new model and algorithm in order to solve this important practical problem.

Scope and purpose

An important contribution of Operations Research is to develop computer-based tools for assistance in decision making. In this paper we show how techniques from combinatorial optimization can assist textile engineers in finding a control of a newly developed braiding machine. We have developed new models and have employed shortest path algorithms (Ahuja RK, Magnanti TL, Orlin JB. Network flows, theory, algorithms, and applications, Englewood Cliffs, NJ: Prentice-Hall, 1993. p. 93–165), and a branch and bound algorithm for the maximum clique problem (Johnson DS, Trick MA, (editors). Cliques, coloring, and satisfiability, DIMACS Series in discrete mathematics and theoretical computer science, Providence, RI: American Mathematical Society, 1996.) to compute controls. The determination of controls is a highly complex task, which can take days or even weeks if performed manually. Hence, computer-assistance is

* Corresponding author. Tel.: + 49-241-8061-85; fax: + 49-241-8888-168.
E-mail address: tore@or.rwth-aachen.de (T. Grünert).

PII: S0305-0548(00)00053-8

a necessity in practice. Indeed, the tools we have developed are employed in practice and have increased productivity enormously.

Keywords: Maximum clique; k-partite graph; Branch and bound; Braiding

1. Introduction

Many practically relevant problems in the design of complex systems include settings where a number of objects have to be selected from a large set of objects so that they are compatible. Compatibility can be checked by pair-wise comparison of objects. These problems can be modelled using the graph–theoretic notion of cliques. Given an undirected graph $G = (V, E)$ with vertex set V and edge set E a clique $C \subseteq V$ is a subset of the vertex set so that every vertex in C is connected to every other vertex in C. A clique of size $s = |C|$ is called a s-clique. If we define a vertex for every object of the design problem and connect compatible objects by an edge in the respective graph, then every clique is a collection of compatible objects. Different objectives can be formulated in the context of cliques. Here we will restrict ourselves to search for k-cliques in k-partite graphs. A graph is called k-partite if the node set can be partitioned into k disjoint sets (partitions) so that no vertices within any partition are connected by an edge. Obviously a k-clique is a maximum clique in a k-partite graph, i.e. a clique with maximal cardinality.

Maximum clique problems in general graphs are difficult to solve – both from a practical and theoretical point of view. The maximum clique problem is NP-hard [1], which implies that it is unlikely that there exists a polynomial time algorithm for solving the problem. Moreover, the problem is extremely difficult to solve to optimality in practice. Depending on the structure of the graph, problems with more than about 500 nodes are beyond the reach of the best optimisation algorithms. Larger problems can only be attacked by heuristics, which do not guarantee that maximum cliques are found.

Our research was motivated by a design problem in textile engineering, more specifically in computer-aided braiding. The task is to find a compatible movement of bobbins along non-colliding paths so that a desired pattern is manufactured. We will show how the problem can be modelled as a maximum k-clique problem in k-partite graphs. From a designer's point of view this means that the objects can be grouped into k groups and that one searches for exactly one object from every group, which is compatible to all selected objects from the other groups. The size of practically relevant instances of our problem with up to a few thousand nodes in the graph require the development of a new specialized algorithm for the k-partite graph case.

This paper is organised as follows: Section 2 gives a short description of the applications, technical properties, and relevant objectives of the braiding process. The construction of a convenient model is explained in Section 3. Section 4 gives a short review of relevant research on maximum clique problems. A new branch and bound optimisation algorithm which solves the model is introduced in Section 5. Computational results for real-world braiding instances and randomly generated instances are presented in Section 6. Finally, some conclusions, possibilities for improvement and future research are given in Section 7.

2. Braiding technology

Fibre-reinforced plastics, which are composed of a textile reinforcement and a polymer matrix and, therefore, called composites, are successfully used in many industrial branches. Applications can be found in such diverse fields as aerospace industry, mechanical engineering, sports gear, and medical materials. Compared to more traditional materials composites can be flexibly adapted so that the material properties specifically satisfy given demands.

However, a wide introduction of these materials in the market conflicts with the high costs. The largest fraction of manufacturing costs stems from the lay-up of two-dimensional textile semi-manufactures (e.g. fabrics) to a three-dimensional component. Therefore, braiding is an appealing alternative. The basic principle of all braiding machines is that on the machine plate, pairs of horn gears rotate in opposite directions to each other so that the thread spools, the so-called bobbins, travel around the horn gears following the guide tracks cut into the bedplate. In this way the threads interlace each other to the braid, which is then taken up vertically.

Taking conventional braiding technology as a basis, a method of producing three-dimensional (3D) braids (Fig. 1) has been developed during the last years at the Institut für Textiltechnik of the RWTH Aachen, the so-called 3D rotary braiding technique [2,3]. With this world-wide unique prototype it is for the first time possible to position every single of the 400 threads at any place in the cross-section of the textile. It consequently becomes possible to produce an architecture of the threads in accordance with the given load situation or the asked geometry. Due to this fact the novel braiding technology creates outstanding properties in 3D technical textiles. 3D braids offer the opportunity of producing the required component configurations in one processing stage without the need for any further stages of cutting to shape and forming into plied assemblies. The consequences are significantly higher impact damage tolerance levels and structural energy absorption (work capacity and vibration damping) [2].

In contrast to conventional braiding machines the horn gear modules in the 3D rotary braiding machine are assembled in columns and rows to form a braiding bedplate (Fig. 2). The horn gear consists of four wings; a wing can be empty or carry one bobbin. The braiding plate of the

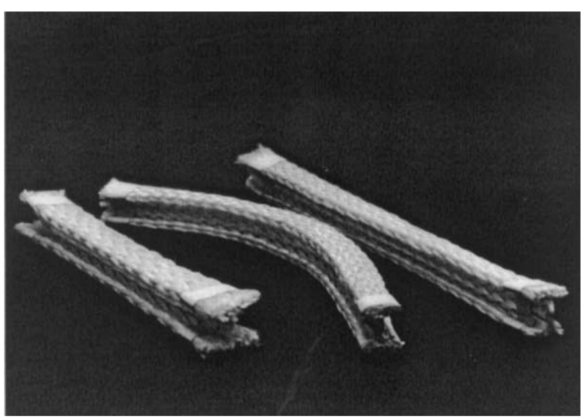

Fig. 1. 3D braid.

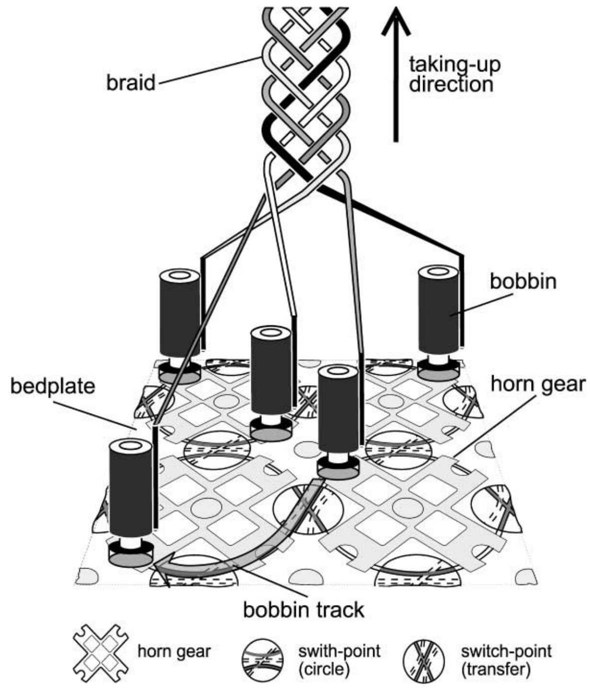

Fig. 2. Principle of a 3D rotary braiding machine.

above-mentioned prototype with an area of 2.20 m^2 consists of 10×10 horn gears, so that 400 wings are at one's disposal.

In order to make the bobbin movement flexible, a horn gear module incorporates a controlled combined clutch-brake in such a way that each individual horn gear and consequently the bobbins situated on it can be individually activated or stopped (Fig. 2). This means that the three movement states of the horn gears are "90°-rotation, left", "90°-rotation, right", and "stop". In addition, rotary switch-points are incorporated between adjacent rotors which, according to the status of the points, retain the bobbin on the horn gear (i.e. status "cycle", cf. switch-points between the upper and lower horn gears in Fig. 2) or transfer it to the adjacent horn gear (i.e. status "transfer", cf. switch-points between the left and the right horn gears in Fig. 2).

The introduction of this element of flexibility into the braiding process requires a synchronisation of all horn gears and switch-points. For this reason the normally continuous braiding process is clocked. It is separated into single steps in such a way that during one step the horn gears rotate 90° or stand still. Before every new step all switch-points can be changed either to the status "cycle" or "transfer". For a transfer of a bobbin to an adjacent horn gear, both horn gears have to rotate clock and counterclockwise and the switch-point between them has to be turned to the status "transfer". Otherwise the bobbin will remain on the first horn gear or a so-called bobbin-horn gear collision will occur (Fig. 3). The two possible cases for a bobbin-horn gear collision are presented in Fig. 3. Another situation which leads to an error is if two bobbins try to pass one switch-point at the same time (Fig. 4). For both possible states of the switch-point a bobbin–bobbin collision takes place.

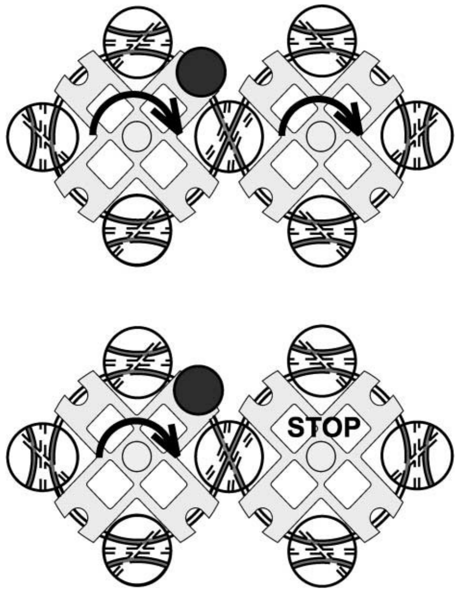

Fig. 3. Bobbin–horn gear collision.

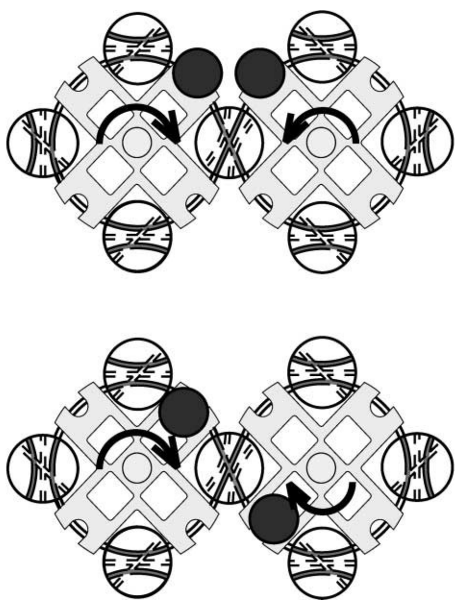

Fig. 4. Bobbin–bobbin collision.

The clocking of the process allows the definition of the braiding pattern by initial and end positions of all bobbins on the bedplate. Starting with an initial configuration each bobbin has to reach its end position within a given number of steps.

With these novel technical facilities it is possible for the bobbin paths to be deliberately varied at every stage. With a view to textile engineering it is desired that approximately 50% of the wings are mounted with bobbins, which move on the machine in such a way that the bobbin paths interlace each other frequently and symmetrically. From the textile engineer's point of view the new braiding technique drastically increases the potential applications of braiding. But a high productivity of the machine has to be ensured in such a way that for a quick production of the braids as many threads as possible interlace each other at every step. Due to this the number of moving bobbins has to be as large as possible.

However, the difficult question to answer is how a given braiding pattern can be manufactured, i.e. how the clutch and horn gear control should be operated, so that collisions are avoided and the braiding pattern becomes feasible. The enormous number of possible controls motivated the modelling of the problem as an optimisation problem and the implementation of an algorithm in order to assist the engineer during the design process.

3. Model

This section describes the construction of a suitable optimisation model for the braiding problem. It is constructed in a two-stage process. In the first stage possible paths for each bobbin are enumerated using a shortest path calculation and an enumeration scheme for different paths. This is described in Section 3.1. The compatibility of different paths depends on the control of the horn gears and the switch points. Section 3.2 shows how different paths can be compared. This leads to the definition of the so-called path compatibility graph. Section 3.3 eventually establishes the relationship between the path compatibility graph and the maximum clique problem in k-partite graphs.

3.1. Alternative paths for each bobbin

The task of the first stage is to create a set P_b of alternative paths for each bobbin $b \in B$. There exists a braiding digraph $D = (V, A)$ for each 3D-rotary machine. Its vertices V correspond to the positions on the braiding field. For a 3D-rotary machine with $m \times n$ horn gears D has $4mn$ vertices.

The information represented by D is whether a bobbin can move from one position of the braiding field to another within only one step. Two vertices v and v' are connected by an arc $a = (v, v')$ if and only if the 3D-rotary machine can move a bobbin from v to v' in one step. The neighbourhood of a position $v_0 \in V$ is the set $N_D(v_0) = \{v \in V \mid (v_0, v) \in A\}$. Since some of the machines can only move a gear horn in one direction or keep it fixed, the resulting braiding digraph may not be symmetric and contains loops $\{(v, v) \mid v \in V\} \subset A$.

A path p is a $(T + 1)$-vector $(v_0, v_1, \ldots, v_T)$ of positions $v_i \in V$ where each member is a neighbour of its predecessor, i.e. $v_{i+1} \in N_D(v_i)$. A path p belongs to the set of alternative paths P_b corresponding to bobbin b if $v_0 = v_0^b$ and $v_T = v_T^b$. v_0^b and v_T^b are the initial and end positions of bobbin b on the bedplate.

In order to compute all alternative paths P_b from position v_0^b to position v_T^b we use enumeration: the first step is to compute the length d_{ij} of a shortest path between position v_i and position v_j for all pairs $(v_i, v_j) \in V \times V$. We can then decide if it is possible to reach the end position v_T^b from a given position $v \in V$ at time t in $T - t$ steps. Starting with a sub-path $(v_0 = v_0^b, v_1, \ldots, v_{t-1})$ for bobbin b all successor positions $v_t \in V$ must be neighbours $v_t \in N_D(v_{t-1})$ and a path of length up to $T - t$ must exist between the position v_t and the end position v_T^b. We do not give further details of the enumeration procedure since this is not the interesting or critical part of our model.

Creating alternative paths for each bobbin in the first phase has an additional advantage: technical criteria to judge the quality of the braid can be used within the enumeration or as a kind of post-processing to filter out those paths which do not meet certain requirements. One example is the use of tracks or corridors. For each time $t \in \{0, 1, \ldots, T\}$ the track or corridor describes one or several neighbouring positions a bobbin is allowed to move to.

3.2. The path compatibility graph

The model we use is based on the concept of compatible paths. Two paths are compatible if they do not belong to the same bobbin, do not block the same positions in the braiding field at the same time, do not use the same switch point for a transition between the periods $t - 1$ and t, $t \in 1, \ldots, T$, and the implied horn gear rotation does not conflict. The required information can be computed for two paths p and p' in $O(T)$ as follows: the blocking of identical positions requires the elements of the path vectors to be pair-wise distinct. For each transition from period $t - 1$ to period t one can compute the index of the affected switch point from the entries of the path vector v_{t-1} and v_t. The implied rotation of the horn gear can be computed equivalently. Both computations can be done in constant time. It remains to compare this information for every transition t to $t + 1$ of the paths resulting in an overall effort of $O(T)$.

The path compatibility graph $G = (P, E)$ has exactly one node for each generated path, i.e. $P = \bigcup_{b \in B} P_b$. Two paths p and p' are connected by an edge $e \in E$ if and only if they are compatible. Recall that two paths for the same bobbin are incompatible. The path compatibility graph is, therefore, $|B|$-partite with respect to the bobbins, i.e. there do not exist edges e between nodes p and p' where $p, p' \in P_b$, $b \in B$. This is an important characteristic, which is explicitly exploited in our algorithm. The construction of G requires all pairs of paths to be compared. This implies a quadratic effort in the number of paths. The entire graph can consequently be constructed in $O((|P|^2 \cdot T))$ time and requires $O(|P|^2)$ memory, which can indeed be prohibitive for very large instances.

3.3. A solution of the braiding problem

The original braiding problem was to determine a path from an origin to a destination for each bobbin on the braiding field so that collisions are avoided. We now show that the problem corresponds to the determination of a clique $C \subseteq P$ of size $|B|$ in the path compatibility graph. Firstly, note that every clique in G corresponds to the selection of a set of compatible paths. Secondly, each path belongs to a different bobbin since paths belonging to the same bobbin are not connected. Thirdly, a complete solution requires that exactly one path is chosen for every bobbin. This requires the clique to be of size $|B|$. Note that there cannot exist any larger cliques in G due to the partition-property. The problem is, therefore, equivalent to the maximum clique problem in $|B|$-partite graphs.

4. Algorithms for the maximum clique problem

The maximum clique problem has been studied extensively in literature. Solution approaches include optimisation algorithms such as branch and bound [4–11], cutting plane techniques [12–14], and methods based on nonlinear (quadratic) programming [15,16]. The most efficient optimisation algorithms are capable of solving problems with upto about 500 vertices to optimality. As can be expected their efficiency tends to decrease with increasing density of the graph. With the limited applicability of the optimisation algorithms in mind, researchers have proposed heuristics, which do not guarantee that the maximum clique is found. These methods include metaheuristics such as tabu search [17,18], genetic algorithms [19], restricted backtracking [20], and neural networks [21,22] as well as problem-oriented heuristics, which apply results from graph theory, for example, for special types of graphs in a heuristic way [23,24].

The earliest approaches to the maximum clique problem were based on branch and bound (or implicit enumeration). Most branch and bound methods exploit the relationship between the maximum clique problem and related combinatorial problems such as coloring, independent set, and vertex cover. For example, an upper bound on the size of a maximum clique is given by the chromatic number, i.e. the minimum number of colors necessary to color all adjacent vertices with different colors. The chromatic number is difficult to find and one, therefore, uses coloring heuristics instead. Such a coloring can be used as a bounding criterion. Branching is usually performed by growing a clique by the addition of one vertex in each node of the search tree. Such an operation is feasible if and only if the added vertex is adjacent to all vertices in the current clique.

It is beyond the scope of this paper to give a complete overview of the different approaches to the maximum clique problem. We have decided to emphasize the important implications for the problem discussed here instead. Firstly, we are not aware of any specialised algorithm for the case of k-partite graphs. Secondly, the optimisation algorithms proposed so far are not able to solve problems of the size necessary in our case. The main reason for this is that they do not utilize the partition-structure of the graph. In particular, a comparison between general clique algorithms and our algorithm is not fair and does not make sense. Thirdly, it has been shown that any heuristic solution with clique size less than k is useless since it does not correspond to a feasible solution for our problem. Moreover, the user may want to evaluate different solutions (i.e. different cliques of the same size) from a different, i.e. textile technology, point of view. We have, therefore, decided to develop a specialised optimisation algorithm for finding all maximum k-cliques in k-partite graphs.

5. The algorithm

We now focus on the branch and bound algorithm for finding all k-cliques in a k-partite graph $G = (P, E)$ where the set of nodes is partitioned according to $P = \bigcup_{b \in B} P_b$.

The main idea is to construct a partial solution $S \subseteq P$ in every step of the solution process. A partial solution corresponds to a clique of G, so starting with an empty set $S = \emptyset$ we insert one node to the set S in every step of the branch-and-bound search tree.

Given a partial solution S the corresponding set $PN = PN(S) \subseteq P$ contains all nodes compatible to all nodes of S. Remember that only nodes of different partitions P_b can be compatible. Let the partial solution S consist of exactly one element of the partitions $B_0 \subseteq B$. Then $PN(S)$ has a representation as

$$PN(S) = \bigcup_{b \in B_0^C = B \setminus B_0} PN_b,$$

where $PN_b \subseteq P_b$ is defined by

$$PN_b = \bigcap_{s \in S} (P_b \cap N_G(s)).$$

The set $N_G(s)$ includes all nodes adjacent to node s, i.e. $N_G(s)$ is the neighbourhood of s in G. If any of the sets PN_b is empty, we know that no clique of size $|B|$ with $S \subseteq C$ can exist. So we can terminate the current node of the search tree corresponding to the partial solution S (bounding). It must be pointed out that the bounding procedure does not look for any objective values since there is no objective function available.

A simple idea for preprocessing is to check whether every node $s \in P_b$ is adjacent to any other partition, i.e. $N_G(s) \cup P_{b'} \neq \emptyset$ for $b' \neq b$. Otherwise s could be eliminated since it cannot be a member of $|B|$-clique. Nevertheless computing the neighbourhood of each node means to compute the entire compatibility-information which in our application is computationally expensive.

Otherwise (if all $PN_b \neq \emptyset$) we have to do a branching step: we first choose one partition $b^* \in B \setminus B_0$. Second we loop over all elements $v_{b^*} \in PN_{b^*}$ and re-start the search with the new partial solution $S \cup \{v_{b^*}\}$.

The way in which the partition $b^* \in B \setminus B_0$ for branching is chosen is of great importance. In order to keep the search tree small we decided to choose PN_{b^*} with minimum cardinality among all sets PN_b with $b \in B \setminus B_0$, i.e. $|PN_{b^*}| \leqslant |PN_b|$ for all $b \in B \setminus B_0$.

When looking for *all* $|B|$-cliques the ordering of the nodes in every partition P_b has no influence on the overall computation time. If the goal were to compute *only one* feasible solution as fast as possible it should be better to look at promising nodes first, i.e. to order the nodes $s \in P_b$ in decreasing sizes of their neigbourhoods $|N_G(s)|$. As pointed out before in the textile application this is computationally expensive and therefore left out. As we will see later (see Section 6) both the branching rule and the simple bounding procedure require only a very small subset of possible edges to be computed. This is an important advantage of the proposed algorithm seen from the application point of view.

The following three data structures describe the state of the solution process. They are given in a syntax similar to template classes in the C + + programming language [25]. The new C + + standard template library (STL) supports data structures composed of lists, vectors, maps, queues, and simple types.

1. The first data-structure `solution` is a vector of nodes with B components:

   `vector⟨node⟩ solution` $[|B|]$.

   It represents a partial solution, i.e. the ith entry of this vector can either be a node of the ith partition `solution` $[i] \in P_i$ or be undefined `solution` $[i] = \perp$. All defined components

$\mathtt{solution}\ [i] = v_i \in P_i$ are compatible nodes and, therefore, they form a clique $C = \{\mathtt{solution}\ [i] | i \in B, \mathtt{solution}\ [i] \neq \perp\}$. At the end of the solution process when all components of solution are defined C is a maximum clique of size $|B|$.

2. The second data-structure compatible_nodes is a vector of size $|B|$, its components are lists of nodes, i.e.

   vector⟨list⟨node⟩⟩ compatible_nodes $[|B|]$.

   The idea behind this definition is to record in the ith component all nodes of partition P_i which are compatible to all defined members of the partial solution stored in solution.

   So if $B_0 \subset B$ is the subset of all defined components of the vector solution, every list compatible_nodes $[b]$ for $b \in B_0$ is an empty list. For its complement set $B_0^C = B \backslash B_0 \subset B$ the list compatible_nodes $[b]$ for $b \in B_0^C$ contains exactly all nodes of partition P_b which are compatible to all solution $[i] \in P_i$ for $i \in B_0$.

3. The algorithm presented below is called recursively. For every level l of the recursion the vector erased

   vector⟨list⟨node⟩⟩ erased $[|B|]$

   contains in its lth component a list of all nodes which are erased from the data-structure compatible_nodes in level l of the recursive process.

One additional vector to store the number of the partition $b \in B$ a given node $v \in P$ belongs to is needed:

vector⟨int⟩ partition $[|P|]$

The entry $b = \mathtt{partition}\ [v]$ means that $v \in P_b$. The initialization of the vectors partition, compatible_nodes and solution is simple. At the beginning erased is a vector of empty lists, so no initialization is necessary.

```
(* Initialization *)
FORALL (b ∈ B)
   FORALL (v ∈ P_b)
      LET partition [v] := b
FORALL (v ∈ P)
   INSERT (compatible_nodes [partition [v]], v)
FORALL (b ∈ B)
   LET solution [b] = ⊥
```

The function *FINDCLIQUE* loops over all nodes of the partition one wants to start with, that is the partition with number *start_partition_no*. The idea is to enlarge the actual solution stored in solution with a node i. All defined components of solution together with the new node i represent the current, partial solution. One now has to decide if the partial solution is useless (bounding) or if it may possibly be enlarged to a complete $|B|$-clique (branching).

The first-inner loop updates the set of compatible nodes compatible_nodes according to the new partial solution. All the incompatible nodes must be erased from compatible_nodes. This is recorded in erased. If the new partial solution were a part of a $|B|$-clique, only one set of compatible nodes could become empty, that is the set compatible_nodes [*start_partition_no*] corresponding to the partition of the new node. By counting with *new_empty_partitions* the number of lists that become empty one has to stop, if *new_empty_partitions* is greater than one.

Otherwise the partial solution could be a subset of a $|B|$-clique. So node i is stored in solution. If solution has exactly $|B|$ defined components, or equivalently, if the depth *level* in the search tree is equal to $|B|$, then a clique is found and the result can be displayed. If solution has undefined components, a new partition for branching must be chosen. The depth-first search continues with the partition corresponding to the smallest, non-empty list in the vector compatible_nodes. This heuristic approach tries to keep the number of iterated calls of the function *FINDCLIQUE* small.

The second inner loop reverses all decisions made for node i: all nodes erased from compatible_nodes have to be inserted into their corresponding list of compatible nodes. The algorithm is stated in pseudo-code below. The procedure is started with $level = 1$ and $start\_partition\_no = b^*$, where b^* is the index of a partition P_b of minimal size, i.e. $|P_{b^*}| \leqslant |P_b|$ for all $b \in B$:

```
FINDCLIQUE(1,b*)

FINDCLIQUE(level, start_partition_no)
{
FORALL (i in compatible_nodes [start_partition_no])
  {
  LET new_empty_partitions := 0
  (* first inner loop *)
  FORALL (b ∈ B)
    IF (new_empty_partitions ≤ 1)
      FORALL (j in compatible_nodes [b])
        IF (not compatible(i,j))
          INSERT (erased [level],j)
          ERASE (compatible_nodes [b],j)
          IF (compatible_nodes [b] is empty)
            LET new_empty_partitions := new_empty_partitions + 1
  IF (new_empty_partitions ≤ 1)
    LET (solution [start_partition_no] := i)
    IF (level = |B|)
      OUTPUT (solution)
    ELSE
      (* branching *)
      LET new_part_no := Index of smallest, non-empty list in
                             {compatible_nodes [k]|k ∈ B}
      CALL FINDCLIQUE(level + 1, new_part_no)
    LET solution [start_partition_no] := ⊥
  (* second inner loop *)
```

```
    FORALL (j in erased [level])
       INSERT (compatible_nodes [partition [j]],j)
       ERASE (erased [level],j)
    }
}
```

6. Computational results

This section presents computational results for both real-world braiding problems and randomly generated problems. All computational tests were performed on a 100 MHz Pentium PC under Windows NT 4.0 with 32 MB RAM. The algorithm was programmed in C++ and compiled using the Microsoft Visual C++ compiler, version 5.0. The compiler target option was set to 'release'.

6.1. Real world problems

From a textile engineer's point of view real-world problems can be described by the braiding pattern they correspond to and their size. The input to our algorithm, however, is not the braiding pattern but the initial and final position of the bobbins on the bedplate together with the number of steps which are allowed before all bobbins have to reach their destination. The number of possible paths tends to increase drastically with the number of steps. Moreover, one usually prefers solutions with the minimal number of steps since they correspond to maximum productivity.

The computational results for the real-world problems are given in Table 1. The first column gives the name of the problem. Attributes of the path compatibility graph are described next. Important attributes are the number of bobbins, which is equal to the number of partitions, the minimal and maximal size of a partition, the number of nodes, i.e. the number of generated paths, and the density of the graph. The density is equal to the number of edges divided by the possible number of edges, i.e. density $= 2|E|/|P|(|P|-1)$. The density of the graphs we consider here is extremely high. This follows from the fact that a path is compatible to almost all other paths since the other paths cover other regions of the bedplate.

The next columns summarize important aspects of the computation results. The first column gives the number of cliques. If the number is greater than 1000, then computation is halted. The tests column gives the number of calls to the routine that evaluates the compatibility of two paths. This number is also given as the percentage of calls relative to the possible number of edges $|P|\cdot(|P|-1)/2$. This indicates whether it is useful to pre-compute the compatibility information before the initialisation of the algorithm or to compute them as needed during the branch and bound phase. The double-test column gives the number of compatibility tests which are computed at least twice. The following four columns show the computational times in milliseconds. The time the algorithm requires before the first clique is found is given in the first column. The second column gives the time until termination, i.e. the calculation of all cliques or the first 1000 cliques. The next two columns give the same values for the case where the compatibility information was computed before the branch and bound phase.

The tests and double tests columns show that only a small fraction of the possible compatibility tests were performed. However, for some of the instances the fraction is about 50%. This is the case

Table 1
Results for some test instances from 3D rotary braiding

Name	Graph $G=(\bigcup_{b\in B} P_b, E)$					Results						
	Part. $\lvert B\rvert$	min $\lvert P_b\rvert$	max $\lvert P_b\rvert$	Nodes $\lvert P\rvert$	Density (%)	Cliques	Tests	Double tests	Eval. time on-line (ms)		Eval. time off-line (ms)	
									First	Overall	First	Overall
A1.5 × 5.2p.5t	36	1	5	108	81	0	1184	0	—	90	—	10
							10.3%	0.0%				
A1.5 × 5.2p.6t	36	6	15	348	80	2	26,552	10,592	941	1862	31	61
							22.0%	8.8%				
A1.5 × 5.2p.7t	36	21	35	948	80	> 1000	679,783	522,655	15,682	38,906	331	2093
							75.7%	58.2%				
A1.5 × 5.3p.7t	36	1	26	416	77	0	729	0	—	80	—	10
							0 4%	0.0%				
A1.5 × 5.3p.8t	36	8	44	876	76	1	34,651	5527	4136	8452	40	100
							4.5%	0.7%				
A1.5 × 5.3p.9t	36	36	85	2136	75	> 1000	1,476,010	1,077,683	30,163	256,489	130	4576
							32.4%	23.6%				
A2.5 × 5.2p.5t	48	1	5	208	89	1	2488	0	200	200	10	10
							5.8%	0.0%				
A2.5 × 5.2p.6t	48	6	15	648	86	20	74,779	22,685	2774	9464	30	151
							17.9%	5.4%				
A2.5 × 5.3p.7t	48	1	8	288	86	0	264	0	—	30	—	1
							0.3%	0.0%				
A2.5 × 5.3p.8t	48	8	29	1132	82	1	71,140	29,762	7401	14,802	80	190
							5.6%	2.32%				
A2.5 × 5.3p.9t	48	36	85	3488	81	37	8,056,642	6,462,768	185,576	1,808,840	581	21,871
							66.3%	53.1%				
LZ.10 × 10.5t.0a	134	1	50	1966	93	0	58	0	—	10	—	1
							0.002%	0.0%				
LZ.10 × 10.5t.2a	134	1	80	2984	93	2	122540	1	83190	83200	211	221
							1.4%	0.00001%				
A1.10 × 10.2p.5t	188	1	6	756	96	0	4568	0	—	621	—	10
							0.8%	0.0%				
A1.10 × 10.2p.6t	188	6	15	2328	96	2	478,323	75,635	132,721	243,851	340	781
							8.8%	1.4%				
A1.10 × 10.2p.7t	188	21	35	5768	95	> 1000	13,678,336	10,097,286	822,152	5,232,620	1101	28,050
							41.1%	30.4%				
A.20 × 20.4t.0a	631	8	20	5072	99%	0	3288	28	—	671	—	20
							0.01%	0.0001%				
A.20 × 20.4t.2a	631	17	40	12,413	—	> 1000	3,544,057	5826	5,862,220	6,019,770	5458	6289
							2.3%	0.003%				

whenever the number of cliques is large. It is also important to store compatibility information which already has been computed. This is revealed by the comparison of the tests and double tests columns. The relevant percentages lie very close to each other in most cases, indicating that it is likely that a compatibility information is used more than once.

It can be seen in the time columns that the algorithm is very fast when the compatibility information is already available (eval. off-line column). The largest value is about 28 s for the 5768 node problem. It is, on the other hand, always necessary to compute the compatibility information on-line in practice since larger instances do not allow the information to be pre-computed off-line. The 12,413 node problem would, for example, require about 10 days for the computation of the entire compatibility matrix.

The data do not reveal any other significant relationship between problem factors such as, for example, the number of nodes, the partition size, and the number of cliques and the computational time. This may also be attributed to the small number of real-world instances which are currently available.

6.2. Randomly generated problems

We also tested our branch and bound algorithm on randomly generated problems. The procedure is similar to that described in [17].

The generator works with 5 input parameters ($\bar{b}$, s, $\bar{s}, a_1, a_2$). The number $\bar{b}$ determines the number of partitions $|B|$. The parameters $s \leqslant \bar{s}$ control the size of each partition P_b, which is randomly chosen from $\{\underline{s}, \ldots, \bar{s}\}$ where all sizes occur with equal probability $1/(1 + \bar{s} - \underline{s})$. The union of the partitions P_b gives the set of nodes P. The real numbers a_1 and a_2 control the density of the random graph and satisfy $0 \leqslant a_1 \leqslant a_2 \leqslant 1$. The procedure is given below:

$\hat{p}$-generator($\bar{b}$, $\underline{s}$, $\bar{s}, a_1, a_2$)
{
LET $B = \{1, \ldots, \bar{b}\}$
(* Construct nodes *)
LET $\mathit{offset} := 0$
FORALL $(b \in B)$
    LET $s[b] :=$ uniform($\{\underline{s}, \ldots, \bar{s}\}$);
    LET $P_b := \{\mathit{offset} + 0, \ldots, \mathit{offset} + s[b]\}$
    $\mathit{offset} := \mathit{offset} + s[b]$
    LET $P := P_1 \cup \ldots P_{\bar{b}}$
(* Construct edges *)
FORALL $(i \in P)$
    LET $\hat{p}[i] :=$ uniform(a_1, a_2);
FORALL $(b \in B)$
    FORALL $(i \in P_b)$
        FORALL $(j \in P_{b+1} \cup \cdots \cup P_{\bar{b}})$
            generate edge (i, j) with probability $\frac{\hat{p}[i] + \hat{p}[j]}{2}$
}

We now focus on the case $a_1 = a_2$, i.e. the $\hat{p}$-generator is equivalent to the uniform random generator on a $\bar{b}$-partite graph. We set $p := a_1 = a_2$. For nodes $i < j$ let $\mathscr{X}_{ij}$ be the random variable which is equal to one if the nodes i and j are connected by an edge and zero otherwise. For i,j belonging to different partitions the random variables $\mathscr{X}_{ij}$ are i.i.d. and the probability for the existence of an edge is $P(\mathscr{X}_{ij} = 1) = p$. The number of cliques in the random graph can be modelled by the random variable

$$C = \sum_{(i_1, \ldots, i_{\bar{b}}) \in P_1 \times \cdots \times P_{\bar{b}}} \prod_{1 \leqslant l < k \leqslant \bar{b}} \mathscr{X}_{i_l i_k}.$$

It is easy to see that the expectation of C is

$$E(C) = \left(\prod_{b=1}^{\bar{b}} |P_b| \right) p^{\bar{b}(\bar{b}-1)/2}. \tag{1}$$

We would like to construct graphs with a small number of $\bar{b}$-cliques. One such clique can be expected if we choose p so that $E(C) = 1$ in (1). This gives an analytical expression for the value of p, given the number and the sizes of the partitions:

$$p = {}^{\bar{b}(\bar{b}-1)/2]}\sqrt{\frac{1}{|P_1| \ldots |P_{\bar{b}}|}}. \tag{2}$$

The computational experiments confirm the theoretically derived attributes of the randomly generated graphs. The most difficult problems are those for which the probability of a $|B|$-clique is about one, according to the formula (2). Slight modifications of the values of a_1 and a_2 around this value strongly influence the difficulty (and computation time). Most problems in Table 2 are of this difficult type. The problems $\hat{p}$(30, 10, 30,.,.) and $\hat{p}$(100, 10, 10,. , .) prove that difficulties are indeed encountered for a narrow range of node degree values. Consider the problems with 1000 vertices. According to formula 2 exactly one clique can be expected if $a_1 = a_2 = 0.954$. The two instances closest to this value, $\hat{p}$(100, 10, 10, 0.94, 0.94) and $\hat{p}$(100, 10, 10, 0.95, 0.95), could not be solved by our algorithm within a reasonable amount of time. It could be proved that no clique exists for smaller values of a_1 and a_2 and that more than 1000 cliques exist for larger values.

An analysis of the relationship between the density and the computational time reveals that it is almost linear for fixed values of the number of partitions and the size of the partitions. Moreover, the time to find the first or a single maximum clique tends to fall with increasing density. This can be expected since there are more maximum cliques in denser graphs and these tend to share a large number of nodes, which restricts the computational effort which is necessary in order to traverse the search tree between the cliques.

7. Conclusions and future research

This paper was motivated by a braiding application from textile technology. The task was to route a given number of bobbins from their origins to their destinations on a bedplate avoiding possible collisions and conflicting controls. The complexity of this task increases drastically with the size of the problem and engineers encounter massive difficulties when attempting to solve this

Table 2
Results for some randomly generated instances

Graph $G = (\bigcup_{b \in B} P_b, E)$					$\hat{p}$-generator		Results			
Part. $\lvert B\rvert$	min $\lvert P_b\rvert$	max $\lvert P_b\rvert$	Nodes $\lvert P\rvert$	Density (%)	a_1	a_2	Cliques	Tests	Time (ms) First	Time (ms) Overall
5	50	50	250	10.93	0.14	0.14	0	17,554	—	130
5	50	50	250	11.62	0.15	0.15	1	18,605	120	140
5	50	50	250	15.78	0.20	0.20	29	28,590	1	210
5	50	50	250	19.63	0.25	0.25	267	46,041	1	310
5	50	50	250	12.12	0.00	0.3	9	20,851	10	150
5	50	50	250	16.02	0.00	0.4	148	33,726	1	260
5	50	50	250	18.12	0.00	0.45	568	45,989	1	360
5	50	50	250	20.11	0.00	0.50	> 1000	46,273	1	411
10	26(20)	37(40)	295	43.99	0.49	0.49	3	182,184	31	842
10	26(20)	37(40)	295	44.85	0.50	0.50	14	219,144	240	1182
10	26(20)	37(40)	295	45.75	0.51	0.51	29	265,537	10	1192
10	26(20)	37(40)	295	45.19	0.40	0.60	48	245,387	40	1652
10	26(20)	37(40)	295	45.66	0.30	0.70	433	372,726	30	2073
10	50	50	500	37.77	0.42	0.42	0	747,648	—	4747
10	50	50	500	38.64	0.43	0.43	3	892,442	2303	4787
10	50	50	500	39.53	0.44	0.44	9	1,088,444	1633	5568
10	50	50	500	41.35	0.46	0.46	61	1,684,557	350	8152
10	50	50	500	43.10	0.48	0.48	440	2,717,872	120	12,828
10	50	50	500	44.91	0.50	0.50	> 1000	2,003,282	10	9233
50	5(5)	15(15)	501	88.99	0.91	0.91	0	171,220,194	—	358,084
50	5(5)	15(15)	501	89.77	0.918	0.918	482	1,534,853,121	50,462	3,854,740
50	5(5)	15(15)	501	89.94	0.92	0.92	> 1000	1,235,851,864	37,974	2,579,290
20	23(20)	39(40)	594	66.46	0.70	0.70	0	73,493,399	—	2,80,082
20	23(20)	39(40)	594	67.38	0.71	0.71	8	127,138,450	19,288	472,900
20	23(20)	39(40)	594	68.32	0.72	0.72	156	229,690,558	3626	848,350
20	23(20)	39(40)	594	67.91	0.70	0.73	55	170,665,743	40,258	627,412
20	23(20)	39(40)	594	68.15	0.65	0.78	608	235,265,070	4387	863,632
30	11(10)	30(30)	611	57.83	0.60	0.60	0	94,733	—	371
30	11(10)	30(30)	611	67.53	0.70	0.70	0	862,871	—	3625
30	11(10)	30(30)	611	77.08	0.80	0.80	0	174,211,708	—	634,393
30	11(10)	30(30)	611	78.06	0.81	0.81	0	446,547,173	—	1,438,960
30	11(10)	30(30)	611	79.01	0.82	0.82	12	1,283,321,587	706,776	3,575,110
30	11(10)	30(30)	611	80.96	0.84	0.84	> 1000	72,706,651	3425	1,86,849
30	11(10)	30(30)	611	84.80	0.88	0.88	> 1000	48,581	10	281
100	10	10	1000	69.22	0.70	0.70	0	45,168	—	150
100	10	10	1000	79.10	0.80	0.80	0	248,668	—	661
100	10	10	1000	84.06	0.85	0.85	0	1,664,263	—	4447
100	10	10	1000	89.02	0.90	0.90	0	91,863,172	—	221,499
100	10	10	1000	90.99	0.92	0.92	0	2,271,710,076	—	4,738,870
100	10	10	1000	92.98	0.94	0.94	—	—	—	—
100	10	10	1000	94.00	0.95	0.95	—	—	—	—
100	10	10	1000	96.00	0.97	0.97	> 1000	114,376	50	400

problem manually. We have, therefore, modelled the problem as the problem of finding all k-cliques in k-partite graphs.

The partition property of the graph can be exploited algorithmically. This leads to the development of a new branch and bound algorithm. The algorithm is able to solve fairly large practical size problems to optimality within a short time. The computation is delayed by the fact that the compatibility information, i.e. the edges of the graph, has to be computed on-line during the branch and bound phase. This time-consuming task increases the computation time substantially in practice.

From a textile engineering point of view this optimisation model only supports a limited part of the design process. It still requires the engineer to define origins and destinations for all bobbins on the bedplate. This is not effective when up to 1000 bobbins have to be considered. One, therefore, essentially needs a computer-assisted system which transforms braiding patterns directly into positions for the bobbins, which can then be routed on the bedplate according to the solution of our algorithm. It is, in addition, desirable to allow a user to evaluate the mechanical properties of different braids which correspond to different cliques in the solution.

Seen from an OR perspective, we believe that the problem of determining all k-cliques in k-partite graphs has many other applications in practice. One might, for example, try to route a number of vehicles on a rail-network or automatically guided vehicles in a factory so that collisions are avoided. It should also be pointed out that the algorithm can be used to find cliques in general graph: first determine different partitions by a graph coloring heuristic where each of the k colors corresponds to one partition. Next, add one dummy node to each partition and connect it to every other node in the other partitions. Clearly the set of all dummy nodes is a k-clique. Now the objective is to include as many non-dummy nodes as possible in the partial solution S. The bounding procedure can now desire feasibility *and* this objective. It becomes possible to input a required clique size before the problem is solved by a slight modification of our algorithm.

A promising path for further improvements is to add a heuristic procedure to the branch and bound algorithm. This procedure can be called at different levels of the branch and bound tree in the hope of generating a maximum clique quicker. Such a procedure is especially relevant in the cases where no clique is found by the exploration of the branch and bound tree and the search cannot be terminated since the bounding criteria do not apply.

References

[1] Garey MR, Johnson DS. Computers and Intractability. A Guide to the theory of NP-completeness. New York: Freeman, 1979.

[2] Obolenski B, Pickett A. Final report to the BMBF-Project Composites with three-dimensional braided reinforcement. Aachen, 1996.

[3] Wulfhorst B, Schneider M. Novel three-dimensional braided structures with continuous transition between different cross-sectional forms. Band- und Flechtindustrie 1996;33(3):88–92.

[4] Bron C, Kerbosch J. Finding all cliques of an undirected graph. Communications of the ACM 1973;16(9): 575–77.

[5] Gerhards L, Lindenberg W. Clique detection for nondirected graphs: two new algorithms. Computing 1979;21:295–322.

[6] Hsu W-L, Ikura Y, Nemhauser G. A polynomial algorithm for maximum weighted vertex packings on graphs without long odd cycles. Mathematical Programming 1981;20:225–32.

[7] Balas E, Yu CS. Finding a maximum clique in an arbitrary graph. SIAM Journal on Computing 1986;15(4):1054–68.
[8] Friden C, Hertz A, De Werra D. TABARIS: an exact algorithm based on tabu search for finding a maximum independent set in a graph. Computers and Operations Research 1990;17(5):437–45.
[9] Babel L, Tinhofer G. A branch and bound algorithm for the maximum clique problem. ZOR — Methods and Models of Operations Research 1990;34:207–17.
[10] Babel L. Finding maximum cliques in arbitrary and in special graphs. Computing 1991;46:321–41.
[11] Mannino C, Sassano A. Edge projection and the maximum cardinality stable set problem. In: Johnson DS, Trick MA, editors. Cliques, coloring, and satisfiability, DIMACS series in discrete mathematics and theoretical computer science. Providence, RI: American Mathematical Society, 1996. p. 205–20.
[12] Nemhauser G, Trotter Jr. LE. Vertex packings: structural properties and algorithms. Mathematical Programming 1975;8:232–48.
[13] Balas E, Samuelson H. A node covering algorithm. Naval Research Logistics Quarterly 1977;24(2):213–33.
[14] Balas E, Ceria S, Cornuéjols G, Pataki G. Polyhedral methods for the maximum clique problem. In: Johnson DS, Trick MA, editors. Cliques, coloring, and satisfiability, DIMACS series in discrete mathematics and theoretical computer science. Providence, RI: American Mathematical Society, 1996. p. 11–28.
[15] Gibbons LE, Hearn DW, Pardalos PM. A continuous based heuristic for the maximum clique problem. In: Johnson DS, Trick MA, editors Cliques, coloring, and satisfiability, DIMACS series in discrete mathematics and theoretical computer science. Providence, RI: American Mathematical Society, 1996. p. 103–124.
[16] Bourjolly J-M, Gill P, Laporte G, Mercure H. An exact quadratic 0-1 algorithm for the stable set problem. In: Johnson DS, Trick MA, editors. Cliques, coloring, and satisfiability, DIMACS series in discrete mathematics and theoretical computer science. Providence, RI: American Mathematical Society, 1996. p. 53–74.
[17] Gendreau M, Soriano P, Salvail L. Solving the maximum clique problem using a tabu search approach. Annals of Operations Research 1993;41:385–403.
[18] Soriano P, Gendreau M. Tabu search algorithms for the maximum clique problem. In: Johnson DS, Trick MA, editors. Cliques, coloring, and satisfiability, DIMACS series in discrete mathematics and theoretical computer science. Providence, RI: American Mathematical Society, 1996. p. 221–42.
[19] Fleurent C, Ferland JA. Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. In: Johnson DS, Trick MA, editors. Cliques, coloring, and satisfiability, DIMACS series in discrete mathematics and theoretical computer science. Providence, RI: American Mathematical Society, 1996. p. 53–74.
[20] Goldberg MK, Rivenburgh RD. Constructing cliques using restricted backtracking. In: Johnson DS, Trick MA, editors. Cliques, coloring, and satisfiability, DIMACS series in discrete mathematics and theoretical computer science. Providence, RI: American Mathematical Society, 1996. p. 89–102.
[21] Grossman T. Applying the INN model to the maximum clique problem. In: Johnson DS, Trick MA, editors. Cliques, coloring, and satisfiability, DIMACS series in discrete mathematics and theoretical computer science. Providence, RI: American Mathematical Society, 1996. p. 125–46.
[22] Jagota A, Sanchis L, Ganesan R. Approximately solving maximum clique using neural network and related heuristics. In: Johnson DS, Trick MA, editors. Cliques, coloring, and satisfiability, DIMACS series in discrete mathematics and theoretical computer science. Providence, RI: American Mathematical Society, 1996. p. 169–204.
[23] Balas E, Niehaus W. Finding large cliques in arbitrary graphs by bipartite matching. In: Johnson DS, Trick MA, editors. Cliques, coloring, and satisfiability, DIMACS series in discrete mathematics and theoretical computer science. Providence, RI: American Mathematical Society, 1996. p. 29–52.
[24] Homer S, Peinado M. Experiments with polynomial-time CLIQUE approximation algorithms on very large graphs. In: Johnson DS, Trick MA, editors. Cliques, coloring, and satisfiability, DIMACS series in discrete mathematics and theoretical computer science. Providence, RI: American Mathematical Society, 1996, p. 147–68.
[25] Stroustrup B. The C++ programming language, 3rd ed. Reading, MA: Addison-Wesley, 1997.

Tore Grünert has a diploma in electrical engineering from the Technical University Darmstadt and received his master and doctoral degrees from the Aachen Institute of Technology (RWTH Aachen). He has published papers in several books and international journals. He is co-author of the forthcoming book 'Optimization in Transportation Logistics'

(with S. Irnich and H.-J. Sebastian). His main interests lie in the field of logistics and the analysis of complex systems.

Stefan Irnich has a Diploma in Mathematics and a Master in Operations Research from the Aachen Institute of Technology (RWTH Aachen). He is now at the Department of Operations Research at RWTH Aachen. He is co-author of the forthcoming book 'Optimization in Transportation Logistics' (with T. Grünert and H.-J. Sebastian). His research interests are network-design problems and large-scale optimization (branch-and-prize, branch-and-cut).

Markus Schneider studied Mechanical Engineering at the Aachen University of Technology (RWTH Aachen) and at the Imperial College of Science, Technology and Medicine, London with the study focus in Plastics Processing. Since 1995 he is scientific employee at the Department of Textile Technology of the RWTH Aachen (ITA) specialising in 3D Braids as reinforcement for composites and in Finite Elements Analysis.

Burkhard Wulfhorst studied textile engineering at college in Wuppertal and at the Technical University Aachen, where he did the German equivalent of a Ph.D. in mechanical engineering. He was head of Research & Development at the spinning machinery company Schubert & Salzer AG, Ingolstadt/Germany. He is Chair of textile technology at the RWTH Aachen and Director of the Institut für Textiltechnik.

H.-J. Zimmermann is Professor Emeritus of the Aachen Institute of Technology. Until 1999 he was Chairman of the Department of Operations Research at this Technical University. He has published more than 230 papers and 30 books primarily in the areas of Operations Research, Decision Theory, Fuzzy Set Theory and Multi Criteria Decision Technology.