

Large grids and local information flow as a reason for high complexity

*Seese, D.**

*Institute AIFB
University Karlsruhe
D-76128 Karlsruhe
Germany

Tel: +49 (0) 721 608 – {6037 | 6557}

E-mail: {seese | schlottmann}@aifb.uni-karlsruhe.de

Schlottmann, F. and ***

**GILLARDON AG financial software¹
Alte Wilhelmstraße 4
D-75015 Bretten
Germany

1. ABSTRACT

The analysis of many complex problems and complex dynamical systems shows that there are systematic dependencies between high complexity and the existence of large grids in the underlying structures. This complexity criterion is formulated in a precise way and analysed in different areas of application, particularly for selected industrial and management problems. We discuss how our criterion can be applied to make complex problems more tractable by exploring structural parameters to control the complexity of problems and systems in complexity engineering. In some areas the criterion is provable in a strong mathematical sense, whereas in others it is confirmed by numerous examples, without finding a counterexample. The areas of application cover: complexity theory, design of efficient algorithms, dynamic systems, chaos theory, neural networks, auctions, capital markets, design of VLSI-circuits, software engineering and risk management.

2. INTRODUCTION

The growing complexity of many real world problems is one of the biggest challenges of our time. Besides the high complexity even of single products or systems and the corresponding complexity of the technologies to develop these components, the problems usually get more difficult e. g. by the globalisation of companies and competition, the quick and ubiquitous flow of information through the growing World Wide Web (transport systems, communication systems, etc.), the appearance of electronic commerce, the speed up of time to market (i. e. from the first ideas to the final products) as a necessity to survive in business, the pure size and the number of components and development steps for new products.

Moreover, it seems that managers and politicians confronted with real life situations and/or systems of growing complexity fail too often, and many problems are too difficult to expect correct or at least adequate decisions, see e. g. Reason (1987), Tschoegl (2000). The problem is that the reason for this is not a lack of intelligence – usually managers are highly educated and are supported by sophisticated tools and qualified staff.

One of the main problems in complexity research is quite fundamental – it is the lack of a precise definition or a useful criterion that helps us to decide whether a problem, situation, or system has to be called complex. Usually such complex problems, situations, or systems are defined by giving a collection of properties characterising them as complex. The typically used properties for describing complex entities are:

inestimable, incalculable, confusing, highly connected, having their own momentum, non-transparent, devious, depending on probabilities, unstable, depend on non-linear processes, having a very large number of parts connected

¹ Partial support from GILLARDON AG financial software is gratefully acknowledged. The views expressed in this paper are results of independent research and do not necessarily reflect the views of GILLARDON AG financial software.

together in a particular pattern, being difficult to understand or to explain because there are many different aspects or people involved.

For instance, Carkhuff et al. (2000) define complexity as

the inability to relate to or represent the interdependent process and processing relationships within and between systems.

It is not the goal of this paper to discuss all possible definitions of different aspects of complexity - especially for computer science many formal definitions that are trying to cover the aspect of the interaction between machines or agents with bounded resources in a problem solving process can be found in the literature (see e. g. Papadimitriou (1994)). Instead, we will present a structural criterion which seems to be responsible for high complexity of many theoretical and practical problems. The basic question is whether there exists a reason for high complexity of problems and the behaviour of systems which can be explained by an examination of the input structures to the problem, the internal structure of the regarded system or the necessary communication structure of the methods that solve the regarded problems algorithmically. To get a possible answer to this question, we have identified a uniform criterion within the huge diversity of theoretical and real life complex problems. The goal of this paper is to present our ideas, arguments and examples together with some proposals how to make the criterion applicable to complexity engineering.

The paper is organised as follows. In the next section we will develop the basic idea of the criterion by discussing NP-hard problems and tiling problems. The theoretical foundation for the structural ideas is presented in section 3, where the necessary definitions of graphs, grids and graph minors together with related main results are presented. In section 4 we build a bridge to the control of dynamic systems. Section 5 presents a collection of some examples from different areas of application: auctions, capital markets, development of VLSI hardware and software as well as operational risk management. The paper ends with concluding remarks and references.

2. COMPUTATIONAL COMPLEXITY AND LARGE GRIDS

When we are thinking about computational complexity, NP-hardness is one of the most famous notions. Many algorithmic problems in real world applications are NP-hard (see Garey & Johnson (1979)) and their NP-hardness is the preferred argument that is given if we are not able to find an efficient solution to a problem and propose a heuristic approach instead of an exact algorithm. By definition, a problem is NP-hard if each problem in the complexity class NP, i. e. all problems solvable by a non-deterministic algorithm in polynomial time, are reducible to it in polynomial time. A problem is NP-complete if it is NP-hard and inside the class NP. Hence, the NP-complete problems are the hardest problems among all those that can be solved by a non-deterministic algorithm in polynomial time and until now there has not been any solution for such a problem by a deterministic polynomial time algorithm (polynomial in the size of the input).

To get a feeling how structure influences the complexity of a problem we first look at computational complexity of decision problems for graphs. Usually a property P of graphs is given here and the question is for an arbitrarily given graph G, whether G has the property P or not. Examples of such properties are:

PLANARITY

instance: graph G

question: Does G have an embedding without edge-crossings in the Euclidean plane?

HAMILTONIAN CIRCUIT

instance: graph G

question: Does G contain a Hamiltonian circuit, i. e. a subgraph of G which is a circuit containing each vertex of G?

While the left problem can be solved in linear time for all graphs, the right one is a standard example of a NP-complete problem, for which no polynomial time solution has been found until now. In Papadimitriou (1994) a profound introduction to computational complexity of decision problems is given. Many problems are investigated with respect to their complexity, e. g. in Garey & Johnson (1979), Downey & Fellows (1999) and Brandstaedt et al. (2000), and many of them are interesting for real world industrial and engineering applications. There are different attempts to make a complex decision problem tractable by restricting the class of problem instances to graphs with a special structure, e. g. to planar graphs or trees instead of the class of all graphs (see section 3 for definitions). It is a surprising observation that almost all NP-complete problems remain NP-complete for almost all restrictions of the input, with the exception of structures closely related to trees (see figure 1 below).

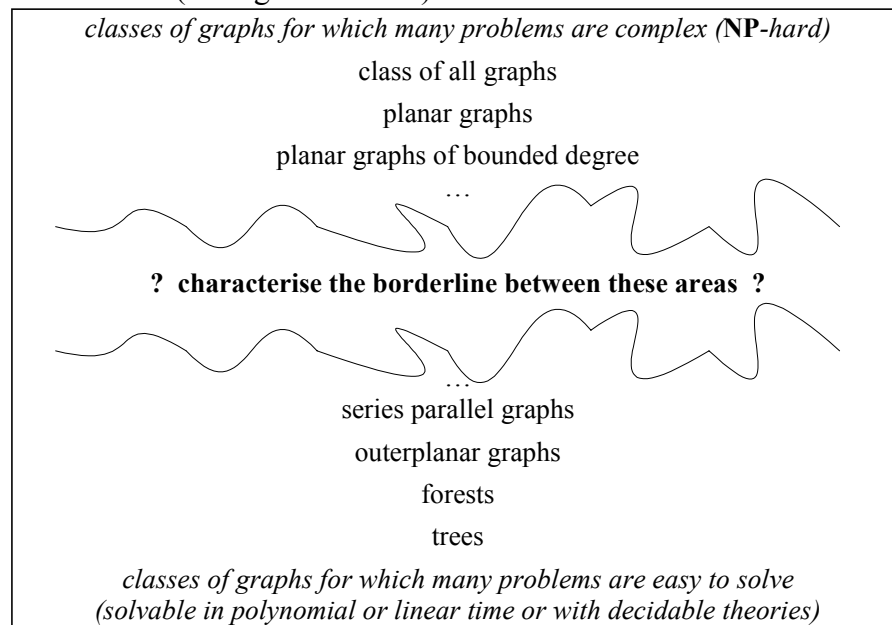


Figure 1: borderline between complex and non-complex graph problems

For trees and graphs with a structure closely related to trees (up to a certain parameter - see e. g. Downey & Fellows (1999)) most algorithmic problems are solvable in polynomial or even linear time. So the natural question is to search for a characterisation of the borderline between these different kinds of behaviour by finding a structural reason for high and for low complexity. Surprisingly, the attempts to achieve this were partially successful (see section 3) and led to the criterion presented in this paper.

To shape the idea of our criterion, it is necessary to look at the proof that a given problem, say P, is NP-hard. The usual way to prove this is to choose a problem Q, which is known to be NP-hard and show that Q can be reduced to the given problem P. The following problem often serves as ‘master’ reduction problem from the problem class NP.

TILING

instance: $D := \{t_0, \dots, t_k\}$ set of square tile types together with two relations $H, V \subseteq D \times D$ (the horizontal and vertical compatibility relations, respectively) and a natural number n

question: Is there a $n \times n$ tiling, i. e. a function $f: \{1, \dots, n\} \times \{1, \dots, n\} \rightarrow D$ such that

(a) $f(1,1) = t_0$, and

(b) for all i, j : $(f(i,j), f(i+1,j)) \in H$, and $(f(i,j), f(i,j+1)) \in V$?

There are many variants and applications of the tiling problem in complexity, decidability, picture recognition and physics, also for other shapes and general covering problems. Figure 2 shows a visual representation of a set of tiles and a tiling for $n=3$.

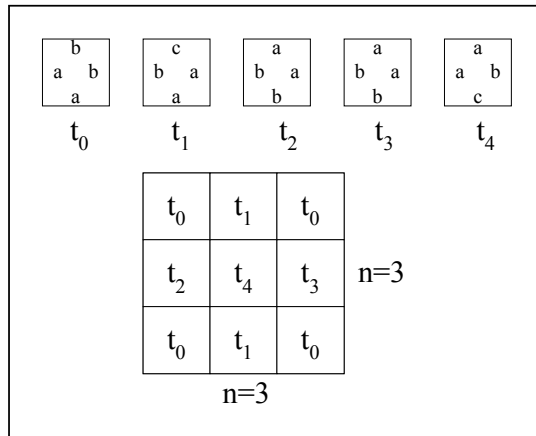


Figure 2: tiling problem example for $n=3$

We will call a decision problem *undecidable* if there is no algorithm to solve it, otherwise it is *decidable*. For our applications the following result is important.

(1) Theorem (see e. g. Papadimitriou (1994)): TILING is NP-complete if n is given in unary representation, it becomes NEXP-complete if n is given in binary representation, the problem becomes undecidable if it is asked whether there exists an $n \times n$ -tiling for all $n > 0$, even when the origin constraint, condition (a), is omitted.

Assume now that we are regarding a decision problem P for a class K of input structures for which we are not able to find a polynomial time solution. In this case it is often conjectured that the problem is NP-hard. If we need a proof for this conjecture the only thing to do is to find a polynomial time reduction of the TILING problem to the original problem, i. e. we have to find an algorithm F which transforms each tiling problem (D, V, H, n) in polynomial time into an input structure $F((D, V, H, n)) = G \in K$ such that there exists an $n \times n$ -tiling of (D, V, H) if and only if $F((D, V, H, n))$ has property P . Usually, this is accomplished by showing that some elements $G \in K$ contain (in a definable way) a large grid structure representing the positions of the tiles in the $n \times n$ -square, that the local structure of these elements G permits the coding of the tiles and permits a ‘flow of information’ along the edges of the grid in such a way that it can be verified whether two neighbouring tiles fit together (horizontally or vertically).

The analysis of this proof leads us to the possibilities to reduce the complexity of a problem P on a class K of structures either by trying to avoid the possibility to find or define large grids inside the input structures, or by avoiding the possibility to code the tiles, or by avoiding the flow of information between parts of the structure coding different tiles. This can be achieved by not allowing input structures which contain large grids (in a definable way – specified later in section 3) or by restricting the local structure of the input in such a way that the structures look locally the same (i. e. are very regular or locally isomorphic), or simply by restricting the flow of information. Of course the coding has to be done in such a way that the ‘decoding’ is reflected in a certain way by the regarded property P .

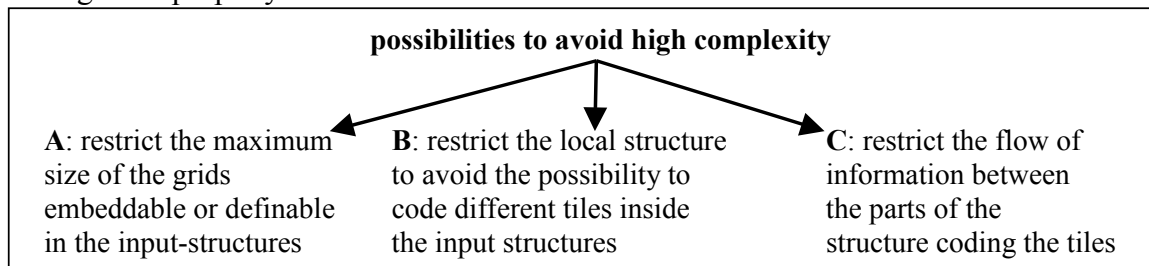


Figure 3: possibilities to avoid high complexity

In the following sections we will make this notions more precise and we will show that the criterion underlying figure 3 is not only of relevance in computational complexity, but it is also important for complexity investigations of dynamic systems and in several other areas of real world applications.

3. SOME IMPORTANT RESULTS FROM TOPOLOGICAL GRAPH THEORY

To give a formal definition of our criterion we use standard mathematical and graph theoretic terminology that can be found in any standard text book. Usually \mathbb{N} denotes the set of natural numbers and i, j, k, l, m, n are used for elements of \mathbb{N} . The cardinality of a set X , i. e. the number of its elements, is denoted as $|X|$. To make the article also readable for those not familiar with graph theoretic concepts we present some of the necessary terminology here and refer the reader to the details in the literature, e. g. West (1996). A *simple graph* $G := (V, E)$ with n vertices and m edges consists of a vertex set $V := V(G) = \{v_1, \dots, v_n\}$ and an edge set $E := E(G) = \{e_1, \dots, e_m\}$, where each edge is an unordered pair of vertices. We write uv for the edge (u, v) and say that u and v are *adjacent* when $uv \in E(G)$. Remember that in this interpretation $uv = vu$. In this case the vertices u and v are denoted the *endpoints* of the edge $e = uv$. Define $n(G) := n$ and $m(G) := m$. The *degree* of a vertex a in a graph G is the number of vertices adjacent to it. A *path* is a finite graph with exactly two vertices of degree 1 and without vertices of degree ≥ 3 . A graph G is said to be *connected* if for each pair of vertices a and b there is a subgraph H of G which is a path and contains a and b . Usually, the graphs in this paper are assumed to be finite. Exceptions of this rule will be clear from the context or will be explicitly mentioned. Two graphs H and G are said to be *isomorphic* if there is a 1-1-function f from $V(H)$ onto $V(G)$ such that for all pairs of vertices (u, v) of $V(H)$ uv is adjacent in H if and only if $f(u)f(v)$ are adjacent in G . A graph H is said to be a *subgraph* of a graph G , denoted as $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. If $e := uv$ is an edge of G , then *contracting* e means replacing both endpoints of e by a new single vertex a , and choosing a new graph $G.e := (V(G.e), E(G.e))$, where $V(G.e) := (V(G) \setminus \{u, v\}) \cup \{a\}$ and $E(G.e) := E(G) \setminus (\{ux : x \in V(G)\} \cup \{xv : x \in V(G)\}) \cup \{ax : x \in V(G)\} \cup \{xa : x \in V(G)\}$. A graph H is a *minor* of a graph G if H results from a subgraph of G by contracting some of its edges. We do not distinguish graphs from their isomorphic copies here.

The following structures are essential ingredients for the high complexity of many problems. Let $n > 0$ be an arbitrary natural number. The n -*grid* Q_n is the graph defined by $V(Q_n) := \{(i, j) : 0 \leq i < n \text{ and } 0 \leq j < n\}$ and $E(Q_n) := \{(i, j), (k, l) : |i - k| + |j - l| = 1, 0 \leq i, j, k, l < n\}$. Define the *infinite grid* $Q_\infty := (V(Q_\infty), E(Q_\infty))$ by $V(Q_\infty) := \{(i, j) : i \in \mathbb{N}, j \in \mathbb{N}\}$ and $E(Q_\infty) := \{(i, j), (k, l) : |i - k| + |j - l| = 1 \text{ and } i, j, k, l \in \mathbb{N}\}$. It is easy to see that for each planar graph H (for a definition of planarity see section 2), there is an n such that H is a minor of the n -grid Q_n (see Robertson & Seymour 1984). A surprising observation is that graphs which avoid n -grids as minors (for a fixed n) have a structure which is up to a certain parameter – the tree width – closely related to trees. The result is one of the structural components of our criterion discussed in the previous section. This structural result is the essential reason that a certain set of algorithmic problems can be solved efficiently.

To make the ideas more precise we need the notion of tree decomposition. A *forest* is a graph which does not contain a cycle. A *tree* is a connected forest. A *tree decomposition* of a graph G is a pair (T, X) , where T is a tree and $X := (X_t : t \in V(T))$ is a family of subsets of $V(G)$ with the following properties:

$(X_t : t \in V(T)) = V(G)$, for every edge e of G there exists $t \in V(T)$ such that e has both ends in X_t , and for $t, t', t'' \in V(T)$: if t' is on a path of T between t and t'' , then $X_t \cap X_{t'} \subseteq X_{t'}$.

The *width* $w((T,X))$ of the tree decomposition (T,X) is $\max(|X_{t-1}| : t \in V(T))$. The graph G has *tree width* $tw(G) := m$ if m is the minimum k such that G has a tree-decomposition of width k .

The following landmark result concerning grids, minors and tree width shows that graphs without large grid minors are essentially ‘tree structured’.

(2) Theorem (see Robertson & Seymour (1996)): For every planar graph H there is a natural number n_H such that every (even possibly infinite) graph G without H as minor has tree width $\leq n_H$.

A class K of graphs has *universally bounded tree width* if there is a natural number m such that the tree width of each graph in K is bounded by m , i. e. $tw(G) \leq m$ for each $G \in K$.

There is a world of results proving that many algorithmic problems for many classes of graphs of universally bounded tree width can be solved in a time being polynomial or even linear in their input size. Moreover, there are many uniform approaches defining an algebraic, a logic or a hybrid calculus which allow us to describe algorithmic problems and to construct efficient algorithmic solutions for such problems, sometimes even in an automatic way (see e. g. Arnborg et al. (1991), Courcelle et al. (2000)).

Together these results give a good theoretical explanation why so many problems for input structures without large grids can be solved efficiently:

- Graphs without large grids have a close similarity to trees.
- For such graphs, many global problems can be solved by algorithms working locally in a way similar to dynamic programming, hence they can be solved in linear time.

As a consequence, we can conclude that almost all algorithmic problems for graphs without large grids (i. e. of bounded tree width) can be solved in linear time. These problems contain almost all NP-hard problems for graphs and networks.

Now assume that grids of arbitrary size can be contained in the class of input structures. Is it possible to give a good characterisation of the structures and problems for which we can expect efficient algorithms in this case? The results in this area are not as elaborate as in the preceding considerations. Hence, we will show an example from classic efficient algorithms underlining our arguments about the connections between structure and complexity.

A standard problem of computer science is the problem of sorting a set of n input elements corresponding to a given key. Such a general sorting problem can be solved in $O(n \log n)$ time. The sorting problem asks for the existence of a linear ordering which sorts the elements according to the same order defined by the given ordering of their keys. In this setting the problem is not obviously a question about graphs of bounded tree width since the graph underlying a linear ordering is a complete² graph (simply transform each directed edge of the ordering into an undirected edge). A complete graph of n vertices has the tree width $n-1$ and contains the $\lfloor \sqrt{n} \rfloor \times \lfloor \sqrt{n} \rfloor$ grid as subgraph.

Looking more carefully at the structures underlying the ordering problem, it is observable that the input structures can be represented as ordered trees, i. e. trees whose branches are linear orderings³, and the output structures are closely related to graphs of path width⁴ 2, i. e. they result from a graph of path width 2 by substituting two of its paths by a linear ordering. But in an ordered tree the successor relation, whose transitive closure is the

² A graph is *complete* if each pair of its vertices is connected by an edge. A complete graph with n vertices is denoted (up to isomorphism) as K_n .

³ In this special case only one branch is used – the given linear ordering.

⁴ *Path width* is defined in the same way as tree width, the only difference is that the underlying graph of the tree decomposition has to be a path.

ordering of the tree, can be used to define the partial ordering of the ordered tree. This can be used as basis of a transformation of the ordering problem into a tree problem. It can be shown in a related way that many algorithmic problems with known efficient solutions, e. g. breadth-first search, depth-first search, planarity, minimum spanning tree, topological sort, max-flow, can be solved by algorithms using a flow of information whose structure is closely related to trees or paths (see Seese & Schlottmann (2001) for more details).

4. STRUCTURE OF DYNAMIC SYSTEMS AND COMPLEXITY

Dynamic systems are very important for many real world applications, particularly in industrial production, manufacturing processes, control engineering and everyday business life contexts. During the last 10–15 years there has been a lot of success in the study of dynamic systems. The literature on these subjects is very large and growing, and it is impossible for us to give an exhaustive presentation. In this section, we will have a look at dynamic systems with respect to the question how structure influences complexity.

On a first look it seems that dynamic systems are typically much easier than complex decision problems, since for dynamic systems we are usually interested in the iteration of a ‘simple’ function f , i. e. we start from a value x_0 and define $x_{t+1}:=f(x_t)$ for all $t=0,1,2,\dots$, so we are interested in the trajectory generated by the iteration of the function f starting from a given value. Possibly, f can have additional parameters. Typical problems in this area are:

STATE CAN BE DRIVEN TO THE ORIGIN, where a system $x_{t+1}:=f(x_t,u_t)$ and a fixed initial state x_0 are given and the question is whether there exist some $T\geq 1$ and controls u_t , $t=0,\dots,T-1$, such that $x_T=0$,

NULLCONTROLLABILITY, asking whether all states x_0 can be driven to the origin,

TRAJECTORY GOES TO THE ORIGIN, asking for a system $x_{t+1}:=f(x_t,u_t)$ and a fixed initial state x_0 whether there exist some $T\geq 1$ such that $x_T=0$, and

ALL TRAJECTORIES GO TO THE ORIGIN, asking whether for every x_0 there exists a T such that $x_T=0$.

For linear systems all these problems can be solved in polynomial time (see Sontag (1995)). For arbitrary systems this question is stated too general and usually too difficult to answer, e. g. the null-controllability problem for general non-linear systems includes the problem of deciding whether a given arbitrary non-linear equation $\Phi(u) = 0$ has a solution (see Blondel & Tsitsiklis (2000) for more details). Even if the system inhibits only a single scalar non-linearity the problem to decide about stability is difficult, as illustrated by the example of Neural Networks whose activation function is a saturated linear function. For this case, Siegelmann and Sontag (1995) showed that **TRAJECTORY GOES TO THE ORIGIN** is undecidable by simulating Turing machines.

What is it that makes such systems complex in case of non-linear functions and why they remain easy for linear functions? Is there a connection to our criterion on grids? These problems are of arithmetic kind in a certain sense and graphs do not appear on a first look. Of course, these problems are proved to be NP-hard or undecidable via the usual proof techniques, hence tilings and Turing Machines are reduced to this problems, and so grids occur on a second view. But the grids can also be found by a more direct consideration. To demonstrate this, let us focus now on a slightly different, but nevertheless famous problem, the decision problem for the Mandelbrot set, firstly asked by R. Penrose. It is defined by the recursion $x=x^2+c$ over the complex numbers. The loop of the recursion terminates with output 1 if $|x|>2$. Blum & Smale (1993) used a special computation model over real numbers from Blum et al. (1989) to show that this set is undecidable.

Another example of an iteration leading to undecidable sets is the Newton's method used to solve mathematical problems in many industrial and engineering applications. It is a method to search for approximate zeros ζ of functions f , i. e. values ζ with $f(\zeta)=0$. The corresponding decision problem is whether the Newton method converges for a given starting point. Using the BSS-model of computation over \mathbb{R} , the following theorem was proved:

(3) Theorem (see e. g. Blum et al. (1998)): The Mandelbrot set and the set of points that converge under Newton's method are undecidable over \mathbb{R} .

Algorithmically, the decision problems from (3) are quite simple, the algorithms are just realisations of one loop. So what is so difficult about computing one loop containing only very simple arithmetic operations? In case of the Mandelbrot set it is just one multiplication and one addition of complex numbers. By analysing a single iteration step we can see that if x and c have the form $x=a+bi$ and $c=d+ei$, then the result of the next step of the iteration is of the form $x=x^2+c=(2ab+e)i+a^2-b^2+d$, where a, b, d and e are real numbers and i is the complex unit. It is obvious that the arithmetic properties of this operation are quite simple. But where is the hidden complexity?

The first reason is the size of the objects handled in this algorithm – the numbers. Let us assume that we work with real numbers having finite binary representations (see below) as inputs and start with an input c of size n . More exactly, we could assume that the length of the binary representation of both components d and e have a length of at most n . After the first step, the length of the representation of the result has already doubled (exactly it is at least $2n-1$; the size of the represented number grows nearly to $2^{(2^n)}$ after n steps). If we use an arithmetic of an arbitrary high precision (i. e. we allow registers of arbitrary large finite size) the representation of the intermediate result has at least a size of $2^t(n-1)+1$ after t steps of running the loop. Hence, even writing down the intermediate results is difficult since they are growing exponentially large. However, it is not necessary to know the complete intermediate result for the final decision, since it is sufficient to check whether the equivalent condition $a^2+b^2>4$ is true for the actual value of $x=a+bi$. To see more details, let us represent the real part and the imaginary part of a complex number by a sequence of bits, respectively, where we separate the bits representing the integer parts from the bits representing the fractional parts by a dot, e. g. the real part by $(u_s u_{s-1} \dots u_1 \cdot u_{-1} u_{-2} \dots u_{-(r-1)} u_{-r} \dots)$ and the imaginary by $(v_t v_{t-1} \dots v_1 \cdot v_{-1} v_{-2} \dots v_{-(r-1)} v_{-r} \dots)$.

Moreover, let us assume that we compute using finite complex numbers only, i. e. numbers whose representation of the real as well as of the imaginary part can be assumed to be finite. To analyse the complexity from a structural point of view, assume that we are interested in the subproblem to decide about the outcome of the underlying recursion after T steps for an arbitrarily given natural number T and an arbitrary finite input x . To decide about the result, it is sufficient to know whether there exists an integer t with $1 \leq t \leq T$ for which there are more than two bits set to 1 in the representation on the left side of the point in the binary representation of a^2+b^2 , where $x=a+bi$ is the result of the computation at step t .

Now define a *communication graph* for an algorithm. The idea is just to look exactly what possibly happens with the content of the variables on a bit level at each time step. So a communication graph consists of vertices that represent all possible contents of the variables on a bit level at each step of time, and there are edges between such 'bit positions' if they possibly influence each other (see Seese & Schlottmann (2001) for details and Hromkovic (1997) for related ideas). We are particularly interested in the size of embeddable grids, i. e. the tree width, and in regularities of the communication graphs.

A simple observation is that the Mandelbrot set problem becomes trivial if the size of the input and the size of the intermediate results are bounded by a fixed n . In this case, the tree width of the corresponding communication graph is bounded and the problem becomes obviously decidable. Particularly, the decidability is trivial since for each fixed bound of the number of bits in the variables the algorithm degenerates to a finite automaton.

(4) Lemma: The tree width of the communication graph of the above algorithm for the Mandelbrot set cannot be bounded.

This is caused by the growing number of necessary bits for the representation of the intermediate results. It is known that there is no uniform bound for the length of the recursion for arbitrary inputs. The rest follows from an analysis of the communication necessary to multiply numbers having large representations. It is not difficult to see that the communication graph defined above contains a bipartite graph $K_{m,m}$ whose size m is growing with the size of the calculated numbers. Since it is a well known fact that the tree width of such graphs is growing with m , the tree width is not bounded.

Hence, we have an indication for well known dynamic systems that complex problems are connected with large grid sizes. With respect to our philosophy, it is useful to observe that the communication graphs are quite regular. However, the irregularity that seems to be necessary in a certain sense for high complexity comes from the different inputs here which cause an irregular flow of information along the regular communication graph. This again gives evidence of a possible connection between high complexity of dynamical systems on one side and large grids, irregularities and an unbounded information flow on the other. Nevertheless, this is not a strict proof that high complexity for dynamic systems and large grids are causally related, it is only an indication that there could be a connection. Related questions can be asked for other dynamic systems, which are of special interest for business applications (see e. g. Sterman (2000)). Complexity of many real world systems and problems is mainly caused by an interaction of structural complexity and the dynamics of the systems and processes. The above mentioned Mandelbrot set is an example for an algorithmic problem based on a simple loop but the prediction of its behaviour is nevertheless very complex. Therefore, it is obviously more complex to try to predict the behaviour of more complicated systems in many real world application contexts, e. g. in manufacturing or other business applications. We will provide some consequences and solutions for further applications in the next section.

5. FURTHER APPLICATIONS

5.1. Auctions and capital markets

Contrary to fixed pricing, where a market participant endogenously sets a price for a product or service that has to be accepted or refused by the responding market participant, in dynamic pricing the price is dynamically determined by bids of the market participants. We concentrate on multi item auctions where interdependent auctioned items are regarded and where bidders have got preferences over combinations of items. There are several applications, e. g. allocation of bandwidth, transportation or manufacturing tasks, as well as contracts between construction companies or electricity markets.

A multi item auction can be modelled as a combinatorial auction, i. e. we start with a universe O of traded objects and the buyers supply the auctioneer with a set A of bids. The i -th bid is a subset A_i of O and a price p_i that the buyer i is willing to pay for all the objects in A_i . Our auctioneer has to choose a collection of bids $B \subseteq A$ that yields the best possible total price while being consistent, in the sense that no two sets A_i and A_j from A

overlap. The auctioneer's decision which of the conflicting bids to accept is related to graph theoretic concepts. A *bid graph* G consists of a set of vertices of G , which are the bids and a set of edges, where an edge is placed between any two bids if they share an object. Each vertex gets a weight equal to the value of the bid it represents. In this notation the auctioneer's goal is to find the most valuable consistent set of bids. But this problem is equivalent to the maximum weight independent set problem, which is NP-hard and cannot be approximated to within a ratio $O(n^{1-\epsilon})$ for an n -node graph and any positive ϵ , unless $P=NP$ (see Akcoglu et al. (2001)). The problem remains NP-complete for cubic planar graphs and becomes solvable in polynomial time for many classes of structures of bounded tree width (see Garey & Johnson (1979)). Hence, we have again a problem caused by large grids and an example for a solution by restriction of the grid size.

Moreover, Aspnes et al. (2000) showed that in a double auction capital market, i. e. a market where stock buyers and stock sellers compete in a two-sided auction, the possibility of predicting future prices depends heavily on the computational complexity of market participants' trading strategies. If there are a large number of traders but they employ a relatively small number of simple strategies, then there is an algorithm for predicting future price movements with high accuracy in polynomial time. However, if the number of trading strategies is large, market prediction becomes computationally more complex than NP-complete problems (Aspnes et al. (2000) define a corresponding strong complexity class). Grids can be found here because of the common reduction techniques used in the proofs of these facts.

5.2. Hard- and software engineering

Many problems in the process of engineering microelectronic circuits are NP-hard. Among those are several problems of placement, global and fine routing and test (see e. g. Lengauer (1990)) and many of the articles in the proceedings of the annual Design Automation Conferences). Especially in VLSI design grids appear in a natural way, often just visible by a visual inspection of the connections of the layout of the circuits. Whether we consider designs of circuits with special functionality depending on a given specification, the division into smaller subcircuits to minimise connections, the logic design, the electrical design, placement, routing, global and fine routing, channel routing, compactification, test of the functionality – grid-like structures appear almost everywhere in a natural way. It is a general and in many cases provable observation that problems become more simple in the more regular memory units and generally in the more regular structured parts of the circuits, as standard cell circuits or gate arrays show. Hence many successful applications in the area of VLSI circuits follow implicitly our approach. The so-called software crisis that has been evolving since the 1960s is still a major problem in all fields of systems and applications. Although much effort has been achieved e. g. by modularization, object-oriented analysis and design, and the development of modern object-oriented languages during the 1990s there are still too many failing software products and projects as well as too many delays in software release dates world wide. It seems that the improvements in development techniques and tools, software engineering concepts and so on are dominated by the growth of requirements on most of today's software products (modern user interface, high speed, real time operation, multiple users at the same time, networking ability) resulting in steadily growing sizes of development teams that are creating complex products, while working in a distributed team at different locations. In many cases, this is still an unmanageable problem. It is clear that no single concept has cured this problem until now. But the above mentioned concepts like the object-oriented approach have contributed major

improvements towards managing complex software development and solving at least some aspects of the software crisis. Moreover, there is a development of software complexity measures (for an overview see e. g. Zuse (1991)) that provide a toolbox for the analysis of functions, loops etc. in a software programme. Such a complexity measure being equivalent to the concept of tree width is the notion of ($\leq k$)-complex listings defined in Thorup (1997). This concept was used to analyse the complexity of listings by looking at the control flow graphs of the programmes which can be derived directly (in linear time) from the parse trees of the listings. Using this concept, it could be proved that Goto-free Algol and Pascal programmes have (≤ 3)-complex control-flow graphs, all Modula-2 programmes have (≤ 5)-complex control-flow graphs and Goto-free C programmes have (≤ 6)-complex control flow graphs.

These results are interesting for small programmes or small parts of whole software systems. Now we consider programming on a large scale, i. e. we allow many (an arbitrary number of) modules, many procedures, many functions or many different classes. Especially in the object-orientated approach it is easy to construct control-flow graphs of arbitrary complexity by creating programmes having arbitrary complex class diagrams. Here, our criterion of complexity is an additional concept for the analysis of software complexity, and it provides guidelines supporting the design of complex software systems that are consistent with practitioners' experiences.

To illustrate this fact, imagine e. g. that are at least two alternatively proposed designs for an object-oriented software system to be developed. It must be emphasised here that in most problems of real world software development there is no single design alternative. We can analyse the structure of the different proposals by applying our grid complexity criterion from figure 3. The first way to analyse the complexity of a system is by looking at the size of the embedded grids in the class diagram (path A of our criterion). From the view of grid complexity, a design with fewer dependencies and less communication needs between classes is better as it results in smaller embedded grids. It is obvious that smaller grids particularly lead to over-proportional smaller consequences of wrong specifications, design and coding errors in conjunction with easier overall testing, and improved reusability of components. Especially the management of the system as a whole (e. g. system stability, managing the release time of the whole system) is improved by keeping the embedded grids small.

Furthermore, a complex class diagram (and complex code, too) can be understood well if the local structures share a similar pattern. Partially, this can be easily accomplished by imposing certain naming conventions on the classes and their contents (e. g. a standard for naming variables and functions), by a standard way of organising the local source code files throughout the development team etc. All measures of this kind are consistent with our proposed path B. Practical experience shows that this is possible without destroying creativity among developers as the benefits are dominant compared to the disadvantage that creating a 'quick and dirty' solution just from scratch will not be possible after establishing the standards. Beyond that, the approach of path B implies that there should be a certain structure among the local connections between classes, i. e. the local interactions between different parts of the software should preferably follow a similar pattern or use the same interface or protocol, respectively. This is a helpful hint when large structures (grids, see path A) cannot be avoided. But in general, this is difficult to achieve because today's software design and development process often follows a bottom-up approach that is required by the object-oriented paradigm and, of course, by the need for efficient work of many developers on the same project at the same time. And in real world projects, the necessary reuse of old components because of time pressure puts further restrictions on this point.

The implication of path C is to judge software complexity by analysing the local information flow between pairs of components/objects at the runtime of the system. Less local information flow and/or fewer function calls between runtime objects lead to *ceteris paribus* less complex software systems because the runtime behaviour of components does not extensively depend from the runtime behaviour of other components, e. g. failures inside single components remain within the same component. As a consequence, understanding the runtime behaviour becomes easier, debugging and isolated testing of single components is simplified, one gets smaller components for reuse and so on. This is somewhat similar to our conclusions from the implications of path A (see above) but now we look particularly at the micro-level runtime behaviour of the system and not on the macro-level of the overall design stage.

As a conclusion, the results from our complexity criterion can serve as guidelines to avoid or manage complexity in this area of application. Many practical experiences from software projects are consistent with the implications from our complexity criterion. Of course, in a real world project there will be no idealised solution by applying one of the paths A,B,C or even all of them. But keeping an eye on the grids and the local structures as well as on the local information flow as shown above will help to deliver results of good quality in time.

5.3. Operational risk

Operational risk (abbreviated by OpRisk) for financial or industrial companies (or other organisations) is a synonym for unexpected losses of profits or cash flows caused by failures of management or internal controls, changing markets, products, services, technologies, human error and/or fraud, failures of information systems or by unmanageable events and complex operations (see e. g. Marshall (2001) for details). It is currently one of the most important and most difficult management problems for many companies throughout the world.

The management of OpRisk requires solutions for many NP-hard problems, e. g. capital market prediction, financial risk management, optimisation problems, scheduling, operations management, facility management, total quality management, reliability engineering, statistical process control, control of non-linear dynamical systems and complex organisations. Large grids and irregular flow of information cause high complexity in many of these problems as indicated in the above sections, so our criterion is of implicit relevance concerning this point.

Furthermore, there are strong dependencies between staff, process elements, information systems, external markets and other organisational elements. These structures, their connections and their dynamics lead to a more complex overall management problem. In a highly connected organisation a failure of a single element or a wrong decision can already result in huge losses. Therefore, we want to point out that our complexity criterion is explicitly applicable here. An OpRisk manager has to keep an eye on the grid size of the organisation (smaller embedded grids are preferable), on the regularities of the organisational elements (more regular patterns are more manageable) and on the dependencies between organisational elements (less pairwise dependencies lead to lower overall risk). Of course, these general considerations cannot be perfectly applied to each real world organisational element. However, they provide an explicit modelling and understanding of complexity in OpRisk management. Since there is still a strong need for explicit measures of organisational complexity to allow quantitative evaluation of OpRisk, our grid criterion, particularly the well-defined tree width of embedded grids in the organisational structure, is of high relevance in this context. It is a better structural measure of OpRisk for an organisation than simple macro-level numbers like the

company's total profit, turnover etc. currently used as proxies for quantitative OpRisk measures. So our considerations about the tiling problem and the three derived paths to manage complexity are also very useful in this real world application context.

6. CONCLUSION

We have provided a structural criterion of complexity in this paper which is important in different theoretical and application-oriented contexts. Starting from computational complexity we have analysed topological graph theory problems to observe how the presence of large grids inside the input structures influences the complexity. A main conclusion is that graphs without large grids are closely related to trees, and restricting the input for computational problems to such tree-like structures leads to non-complex problems. Therefore, we have found a first way to make complex problems more tractable by restricting their structure. For problems where the grid sizes cannot be bounded we have identified two other possibilities to avoid high complexity: Either the local structure inside the large grids has to be very regular, or the flow of information between the grid elements has to be restricted. Otherwise, if a problem is not restricted in at least one of these three structural criteria the problem will be complex (i. e. NP-complete).

After our considerations about graphs and related problems we have analysed the relationship between the structure of dynamic systems and their complexity. It has been shown that dynamic systems containing recursive iterations of a single non-linear function can lead to complex problems, even if the recursive iterations are based on quite simple loops. This is important e. g. for many industrial production processes and control engineering problems. We found indications that our complexity criterion is useful to manage complexity in these areas, too, since our analysis of the iteration steps in simple dynamic systems has led to the discovery of large grids inside the communication graphs of the regarded variables' contents. This is a basis for analysing more complicated dynamic systems by examining the interaction between their components.

We have considered the presence of large grids and the resulting complexity of problems in further applications like dynamic pricing in auctions or capital markets, hard- and software engineering and risk management. Particularly for software engineering, we have discussed the use of our complexity criterion for real-world projects. The implications of our considerations are consistent with experiences from real software projects. In the very recent field of Operational risk management there is a need for structural complexity measures and complexity management guidelines, so our criterion is of high relevance in this context, too.

Besides the results and indications that we have discussed, there are still many open questions. For instance, there is a need for further empirical validation of our concepts in real-world applications. The proposed grid-oriented criterion can lead to tools which support complexity engineering of real life problems if the corresponding parameters influencing the complexity can be controlled, e. g. by regarding designs of smaller grid sizes or with more regular patterns, or by using organisational structures which avoid high complexity in our sense. Finally, we think that we have contributed an idea that makes a difference in understanding of complexity.

BIBLIOGRAPHY

- Arnborg, S., Lagergren, J. & Seese, D. 1991, 'Easy problems for tree-decomposable graphs', *Journal of Algorithms* 12, pp. 308-340.

- Akcoglu, K., Aspnes, J., DasGupta, B., Kao, M. 2001, 'Opportunity cost algorithms for combinatorial auctions', <http://www.cs.yale.edu/homes/aspnes/auctions-abstract.html>.
- Aspnes, J., Fischer, D., Fischer, M., Kao, M. & Kumar, A. 2000, 'Towards understanding the predictability of stock markets from the perspective of computational complexity', preprint to appear in *Proc. of 12th annual ACM SIAM Symposium of Discrete Algorithms*.
- Blondel V. & Tsitsiklis, J. 2000, 'The boundedness of all products of a pair of matrices is undecidable', *Systems & Control Letters* 41, pp. 135-140.
- Blum, L., Shub M. & Smale, S. 1989, 'On the theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines', *Bulletin of the Amer. Math. Soc.* 21, pp. 1-46.
- Blum, L. & Smale, S. 1993, 'The Goedel incompleteness theorem and decidability over a ring', in *From Topology to Computation: Proc. of the Smalfest*, eds M. Hirsch, J. Marsden, M. Shub, Springer, New York, pp. 321-339.
- Blum, L., Cucker, F., Shub, M. & Smale, S. 1998, *Complexity and real computation*, Springer, New York.
- Brandstaedt, A., Bank Le, V. & Spinrad, J. 2000, 'Graph classes: A survey', *SIAM Monographs on Discrete Mathematics and Applications*.
- Carkhuff, R., Carkhuff, C., Cook, A., O'Brien, W., Tisdale, D. & Rayson, P. 2000, 'Complexity, Simplicity and Collaboration', in *Complexity and Complex Systems in Industry*, Conference Proceedings, eds I. Mc Carthy & T. Rakotobe-Joel, University of Warwick, pp. 86-91.
- Courcelle, B., Makowsky, J. & Rotics, U. 2000, 'Linear time solvable optimization problems on graphs of bounded clique width', *Theory Comput. Systems* 33, pp. 125-150.
- Downey, R. & Fellows, M. 1999, *Parameterized Complexity Theory*, Springer, Berlin.
- Garey, M. & Johnson, D. 1979, *Computers and Intractability*, W. H. Freeman and Company, New York.
- Hromkovic, J. 1997, *Communication complexity and parallel computing*, Springer, Berlin.
- Lengauer, T. 1990, *Combinatorial algorithms for integrated circuit layout*, John Wiley & Sons, Chicester.
- Marshall, C. 2001, *Measuring and Managing Operational Risk in Financial Institutions*, John Wiley & Sons, Singapore.
- Papamitriou, C. 1994, *Computational Complexity*, Addison-Wesley, Reading.
- Reason, J. 1987, 'The Chernobyl Errors', *Bulletin of the British Psychological Society* 40, pp. 201-206.
- Robertson, N. & Seymour, P. 1984, 'Graph minors III. Planar tree-width.', *J. Combin. Theory Ser. B.* 36, pp. 49-64.
- Robertson, N. & Seymour, P. 1986, 'Graph minors V. Excluding a planar graph.', *J. Combin. Theory Ser. B.* 41, pp. 92-114.
- Seese, D. & Schlottmann, F. 2001, 'Structural reasons for high complexity: A survey on results and problems', working paper, University Karlsruhe, 2001.
- Siegelmann, H. & Sontag, E. 1995, 'On the computational power of neural nets', *Journal of Computer and System Sciences* 50, pp. 132-150.
- Sontag, E. 1995, 'From linear to non-linear: some complexity comparisons', *Proc. IEEE Conf. Decision and Control*, New Orleans, pp. 2916-2920.
- Serman, J. 2000, *Business dynamics*, McGraw-Hill, Boston.

- Thorup, M. 1997, 'Structured Programs have Small Tree-Width and Good Register Allocation', in *Graph-Theoretic Concepts in Computer Science*, ed R. Moehring, Springer, Heidelberg, pp. 318-332.
- Tschoegl, A. 2000, 'The Key to Risk Management: Management', in *Risk Management*, eds M. Frenkel, U. Hommel, Springer, Heidelberg, pp. 103-120.
- West, D. 1996, *Introduction to Graph Theory*, Prentice-Hall, Englewood Cliffs.
- Zuse, H. 1991, *Software complexity*, DeGruyter, Berlin.