Department of Theoretical Computer Science
Faculty of Computer and Information Science
University of Ljubljana

A Laboratory for Algorithms and Data Structures

Borut Robič

Prof. Dr Borut Robič, head of the laboratory.

Among those problems which are solvable, there are still some so difficult that for all practical purposes they cannot be solved in their full generality on a computer. Such problems are called "intractable", because they have been proved to require at least exponential time for their solution. For some others, usually called "NP-complete" problems, the implication is that exponential time is required to solve them; however, if any of these could be solved in less than exponential time, then a great number of problems could also be solved substantially more quickly. The significance of the class of NP-complete problems is that it includes thousands of important natural problems from computer science, mathematics, economics, and other fields. These problems have been examined seriously, yet none of them is known to have a quick (i.e. polynomial) solution.

The fact that theory doesn't bring only good news has pushed researchers in a number of directions, intended to try to alleviate intractability. For example, in practice, we may not need the optimal solution to a problem; a solution that is nearly optimal may be good enough and may be much easier to find. Algorithms of this kind are called approximate. Other problems seem to be more easily solvable by probabilistic algorithms, which use the outcome of a random process, such as flipping a coin. The price paid for this is the possibility of error, but for some problems this possibility can be minimised and safely ignored. Yet another approach is to use computer networks, where there may be many computers, or many processors within a single computer, to work on a problem and compute the solution jointly. Combining all these approaches, we can take inspiration from ants (or some other biological or physical system) in order to design algorithms for solving computationally demanding problems.

In the following, we describe some of our application-directed research.

Location problems and uncertainty

Models in which a number of facilities have to be located in the presence of customers, so as to meet some objective, have been studied for many years. For example, facilities can be warehouses, supermarkets, parking garages or hotels, as well as facilities that are not so expensive to set up, such as computers or communication transmitters. Although these examples are different from each other, they share common features, which are formalised in the so-called location problem. Informally, this is defined as follows: given a space, a distance function that is defined between any two points in that space, a number of existing "customers" that are located in the space under consideration and who have certain "demand" for a product, the task is to locate one or more "facilities" in that space that will satisfy some or all of the customers' demand. There are many variations of this problem. Usually, the facility to be located is small in relation to the space under consideration, but the space can be continuous, discrete, or a network, and there are also several ways of choosing the distance (e.g. Euclidean, Manhattan, etc.). Unfortunately, it can be shown that many of these variations are NP-complete, thus requiring either specialised and highly efficient exact solution techniques or good heuristics. In our laboratory, we develop algorithms and methods for solving location problems and other optimisation problems where there is uncertainty of input data.

Prof. Dr Borut Robič, head of the laboratory: in the Laboratory for Algorithms and Data Structures, the following areas of application-directed research are currently being pursued: combinatorial optimisation (e.g. facility location problems, replication problems in distributed computing, routing problems in networks, mapping and scheduling problems in parallel and distributed systems) and compiler design (e.g. combining top-down and bottom-up parsing). Members of the laboratory also pursue fundamental research in algorithmics, such as modern techniques for coping with intractability (e.g. approximate and randomised computing) as well as other aspects of algorithm design and computational complexity (e.g. algorithms for fast matrix multiplication, bio-inspired and parallel algorithms).

Assist. Dr Jurij Mihelič: uncertainty of input data is often present in practical optimisation problems. For this reason, we are interested in algorithms and methods for solving these problems in a flexible way. This means that the solution has to be constructed in such a way that it can easily be adapted to reflect subsequent change in input data, while at the same time retaining its desirable properties.

Distributed processing

Computers are becoming a commodity, emerging in all aspects of modern life. However, these computers are mostly under-utilised in terms of their computing power, as well as their storage capabilities. On the other hand, science is changing rapidly. Large-scale scientific endeavours are carried out via collaborations on a global scale in which information and computing technology plays a vital role; these are popularly termed "e-Science". The need for enormous amounts of computing power and data has motivated the development of new technology which would con-

nect all the idle computing power, data capabilities and other computer services into one virtual computer. Thus, grid computing has emerged as one of the key computing paradigms enabling the creation and management of Internet-based utility computing infrastructure for the realisation of e-Science. There are many problems which arise when constructing such systems. Some of these problems are the focus of the research in our laboratory.

Assist. Dr Uroš Čibej: one of the most critical problems in data intensive grids is the optimisation of the access to data. In modern distributed systems, large amounts of data are being generated. Instead of keeping the data in one place, several copies can be made on different nodes of the system. In this way, the users can access the closest copy, thus making the transfer of data much faster. However, finding optimal locations for them is an NP-complete problem.

We develop formal models that describe these problems in strictly mathematical language and make them more amenable to analysis. We design new algorithms for solving data-replication problems and use various simulation tools to empirically evaluate them. In collaboration with the University of Melbourne, we developed a component for the GridSim simulator, which makes such evaluations easier.

Another very important problem is the search for specific data. Mechanisms need to be developed which enable users to find the data they require from the incredible number of different files and databases available in a typical grid. An algorithm has been developed in our laboratory that uses a biologically inspired technique called ant-colony optimisation. A user sends a query in a form of a set of agents (ants) which "run" around the nodes of the grid collecting the required information. The algorithm shows good results and we believe it will improve the quality and usability of today's grids.

Compiling

Computer programs are written in programming languages which are designed to ease the programmer's work of describing the computation to be performed. However, before this program can be processed, it must first be translated from a programming language into the computer machine language. Translating a program from one language into the other is called compiling, and a tool which automatically compiles a program is a compiler. Unfortunately, compilers are complex programs. Each is written for a specific programming language and for a specific processor. A compiler needs to handle every detail of the programming language and requires every detail of the processor's architecture to be integrated. Additionally, it is expected that the compiler-generated translation is improved (optimised). A modern compiler consists of two main parts. The compiler's front-end is used for analysis of the program which is being compiled, and for translating it into the compiler's internal representation. This part of the compiler is also responsible for detecting various syntactic and semantic errors in the program, although not all can be found by the compiler. The second part, or a compiler's back-end, is used for generating the translation of the program being compiled. With the development of ever more sophisticated programming languages, more powerful methods for program analysis are needed in compilers. Thus, a number of new parsing algorithms have been developed in the Laboratory.

Many techniques and algorithms that were developed in the compiler field are used in much wider context nowadays. For instance, each time a web browser displays a web page, it must analyse its description and this is done by syntax-analysis methods similar to those used in compilers. Furthermore, a number of algorithms developed for program analysis have been used for gene analysis in bioinformatics.

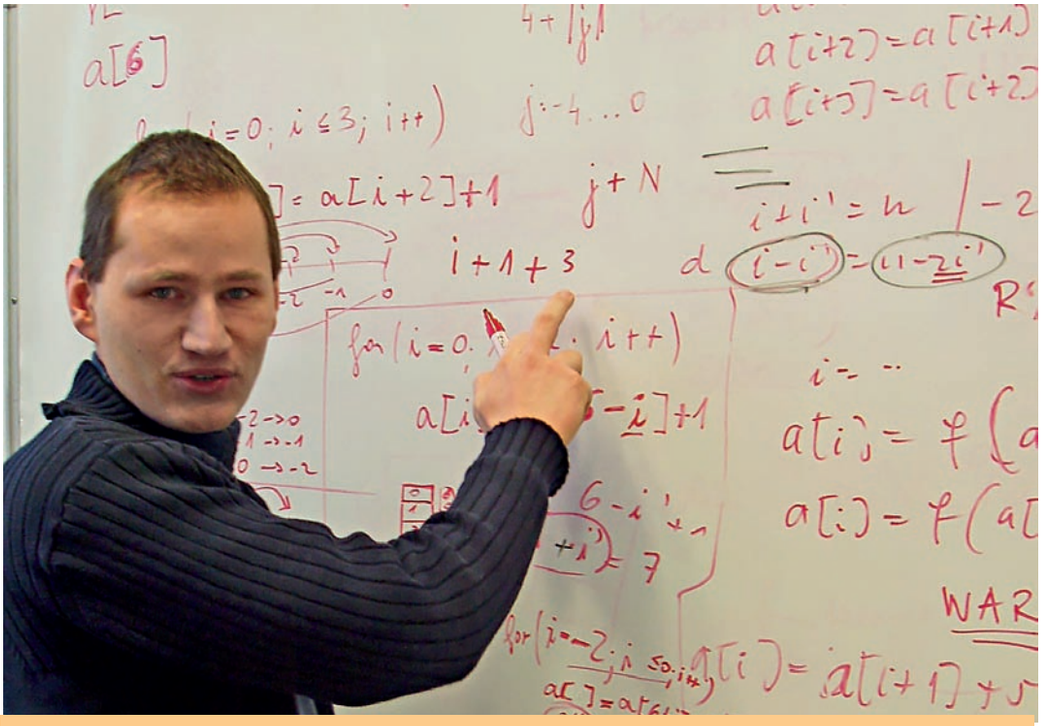
Routing in regular networks

Routing is the process of selecting paths in a network along which to send data or physical traffic. It can be performed for many kinds of networks, including the telephone network, the Internet and transport networks. In our laboratory, we are interested in routing within computer networks. Such a routing directs forwarding of logically addressed data packets from their source computer toward their ultimate destination computer through intermediary nodes. Routing is usually performed by dedicated hardware devices, but it can also be controlled

Doc. Dr Boštjan Slivnik: our new algorithms simplify the construction of the syntactic analyser, the part which represents the backbone of every compiler's front-end, in two ways. Firstly, far fewer restrictions are imposed on the code that the analyser is augmented with, and thus the compiler writer need not apply any "dirty tricks". Secondly, by applying these algorithms, the syntax of the programming language for which the compiler is made can be described more conveniently. At the same time, they provide much better mechanisms for error reporting – something that a programmer, the user of a compiler, badly needs.

by software stored in each node of the network. Routing techniques differ in how resources are allocated, especially when collisions occur. If two packets collide, one of them "wins" (i.e. gets the desired resource) while the other "loses". This raises the question of what to do with the losing packet? There are several possibilities, one of which is to send the packet in an alternative direction.

One of the research areas of the laboratory is the design of algorithms which construct alternative directions, and hence alternative routing paths, either statically (before the routing starts) or dynamically (when the collisions appear). Dynamic routing algorithms are capable of on-the-fly decisions on where to send a losing colliding packet to stay on one of the shortest paths to its destination. Dynamic routing algorithms are more flexible in avoiding collisions and routing problems that arise from congestion or node/link faults.



Doc. Dr Tomaž Dobravec: our simulations have shown that in regular networks, such as circulant graphs, tori, and hypercubes, dynamic routing algorithms outperform their static counterparts in about 11% of all cases. This is because dynamic routing offers a larger set of routing possibilities during each routing step.