LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
Department of Information Technology

Evolutionary computing in search-based software engineering

The topic of master's thesis has been confirmed in the Departmental Council of Department of Information Technology on 3rd December 2003.

Supervisors: D.Sc. (Econ.), Professor Jouni Lampinen and D.Sc. (Econ.), Lecturer Timo Mantere.

Lappeenranta 23rd March 2004

Leo Rela
Korpraalinkuja 3 As. 1
53800 LAPPEENRANTA
Finland
Tel. +358 40 51 31 297
leo.rela@iki.fi

Abstract

Lappeenranta University of Technology
Department of Information Technology
Leo Rela

Evolutionary computing in search-based software engineering

Master's thesis
2004

125 pages, 30 figures and 2 tables.
Supervisors: Professor D.Sc. (Econ.) Jouni Lampinen and Lecturer, D.Sc. (Econ.) Timo Mantere.

Keywords: Evolutionary algorithms, genetic algorithm, search-based software-engineering, software production.

This master's thesis aims to study and represent from literature how evolutionary algorithms are used to solve different search and optimisation problems in the area of software engineering. Evolutionary algorithms are methods, which imitate the natural evolution process. An artificial evolution process evaluates fitness of each individual, which are solution candidates. The next population of candidate solutions is formed by using the good properties of the current population by applying different mutation and crossover operations.

Different kinds of evolutionary algorithm applications related to software engineering were searched in the literature. Applications were classified and represented. Also the necessary basics about evolutionary algorithms were presented.

It was concluded, that majority of evolutionary algorithm applications related to software engineering were about software design or testing. For example, there were applications about classifying software production data, project scheduling, static task scheduling related to parallel computing, allocating modules to subsystems, N-version programming, test data generation and generating an integration test order. Many applications were experimental testing rather than ready for real production use. There were also some Computer Aided Software Engineering tools based on evolutionary algorithms.

Tiivistelmä

Lappeenrannan teknillinen yliopisto
Tietotekniikan osasto
Leo Rela

Evoluutiolaskennan sovellutukset ohjelmistotekniikassa

Diplomityö
2004

125 sivua, 30 kuvaa ja 2 taulukkoa.
Tarkastajat: Professori, KTT Jouni Lampinen ja Lehtori, KTT Timo Mantere

Hakusanat: Evoluutioalgoritmit, geneettinen algoritmi, ohjelmistotekniikka, ohjelmistotuotanto

Tämän diplomityön tavoitteena on kartoittaa ja esitellä kirjallisuudessa raportoituja evoluutioalgoritmien sovellutuksia ohjelmistotekniikan alueelle liittyvien haku- ja optimointiongelmien ratkaisuun. Evoluutioalgoritmit ovat erilaisia luonnon evoluutioprosessia jäljitteleviä menetelmiä, joissa keinotekoinen evoluutioprosessi arvioi ratkaisun hyvyyttä mittaamalla tavoitefunktiolla populaation yksilöt eli ratkaistavan ongelman eri ratkaisuehdokkaat. Seuraavan sukupolven populaatio yriteratkaisuja muodostetaan yleensä käyttäen nykyisen sukupolven hyvien yksilöiden ominaisuuksia soveltaen erilaisia mutaatio- ja risteytysoperaatioita.

Työssä etsittiin, luokiteltiin ja esiteltiin erilaisia kirjallisuudessa julkaistuja evoluutioalgoritmien ohjelmistotekniikkaan liittyviä käyttötapoja ja sovellutuksia sekä käytiin läpi tarpeellisin osin itse evoluutioalgoritmien teoriaa käsitteineen.

Työssä havaittiin, että suurin osa evoluutioalgoritmien sovellutuksista ohjelmistotekniikassa liittyy ohjelmistojen suunnittelu- ja testausvaiheisiin. Sovellutuksia oli mm. ohjelmistotuotantoon liittyvän datan luokittelu, projektin aikataulutus, rinnakkaislaskentaan liittyvä staattinen tehtävien jako, moduulien jako alijärjestelmiin, N-versio ohjelmointi, testitapauksien generointi ja integrointitestausjärjestyksen luonti. Suurin osa sovellutuksista oli luonteeltaan enemmänkin kokeellisia kuin soveltuvia suoraan todelliseen käyttöön. Toisaalta valmiita evoluutioalgoritmeihin perustuvia tietokoneavusteisen ohjelmistosuunnittelun välineitä on olemassa.

Contents

Symbols and Abbreviations

ANN	Artificial Neural Network
BAN	Burrows, Abadi and Needham logic (BAN, A logic of authentication)
CASE	Computer Aided Software Engineering
C#	Csharp, an object-oriented language
CDG	Control-Dependence Graph
CDC	Condition-Decision Coverage
CFG	Control-Flow Graph
CG	Conjugate Gradient method
CI	Computational Intelligence
COCOMO	Constructive Cost Model
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DCS	Distributed Computing System
DSH	Duplication Scheduling Heuristic
EA	Evolutionary Algorithm
EP	Evolutionary Programming
EMG	Extended Module Graph
ES	Evolution Strategy
FP	Function Point
GA	Genetic Algorithm
GGGP	Grammar Guided Genetic Programming
GP	Genetic Programming
LOC	Lines of Code
MDG	Module Dependency Graph
ML	Machine Learning
MMRE	Mean Magnitude Relative Error (mean of absolute percentage errors)

MQ	Modularization Quality function
MSE	Mean Square Error
NP	Non-deterministic polynomial
NVP	N-Variant Programming, N-Version Programming
OGA	Ordered-Deme Genetic Algorithm
OO	Object-Oriented
PARAdeg	The Parallel Degree
PGA	Parallel Genetic Algorithm, Partitioned Genetic Algorithm
PN	Program Net
RAD	Rapid Application Development
STGP	Stongly Typed GP
TPG	Task Precedence Graph, Test Precedence Graph
UML	Unified Modeling Language

1 Introduction

This Chapter represents briefly the overall structure of this document, research objectives, and the background concepts related to this research: software engineering, computational intellicenge and search-based software engineering.

1.1 Research objectives

The objective in this master's thesis is to study and represent wide selection of applications from literature, where evolutionary computing methods (evolutionary algorithms, EA) are used to solve different search and optimisation problems in the area of software engineering. This thesis will concentrate only on the applications related to software production rather than cases, where evolutionary algorithms are used only in the software. It was to be expected beforehand that the majority of evolutionary algorithm applications related to software engineering were about software testing.

Publications reporting the use of EA methods to solve software engineering problems are classified depending on their position in software development process and their application area. The objective is to find promising application areas for EA, to find areas where EA will be probably used in the practical softare engineering and to discuss the possibilities of more research.

1.2 Structure of this document

The Chapter 1 is an introduction and describes the background concepts and research objectives for this master's thesis. The Chapter 2 provides the basic information and theory about evolutionary algorithms, which are needed to understand the contents of this thesis.

This thesis is reporting, how evolutionary algorithms are used for solving software engineering problems. In Chapter 3, is it described how those applications found in the literature are being represented and classified to classes corresponding four major phases of the software development process. The applications itself are described in Chapters 4, 5, 6, and 7.

The Chapter 8 is the summary and describes the number and type of publications publications related to the subject of this thesis. In addition, the overall findings are also described here. In Chapter 9 there are the final conclusions of this research.

1.3 Software engineering

IEEE standard 610.12-1990 defines the term "*Software engineering*" as:

"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software." [IEE90]

In this thesis that term means work or single process that results in a complete, tested software product that behaves much as possible like the original model that was presented in requirements analysis. Therefore focus is in *software development* or *software production*.

The history of software engineering is relatively short when compared to other engineering sciences. The software technology has developed rapidly and the size of a usual software product can be doubled in a few years. Thus, there exists a rapidly growing demand for new software having increased complexity, but still the productivity of the software development work grows slow, as it is said to be only about four percents per year. [HM98, PP98]

Software engineering methods are usually products of interaction between science and engineering. New problems are usually solved with *ad hoc* solutions, which will continue their life as a folklore. Folklore gets a systematic form and soon we will have models and theories, which can be applied commonly to real world problems [Sha90]. After that, we have also new problems.

1.4 Computational intelligence in software engineering

In the past, the research of intelligent and adaptive systems has organized to the research area that is called *computational intelligence* and *soft computing*. The idea is to replace the lack of explicit models and human knowledge with numerical models that are computed to fit the solution of the problem. [Ala98] Most famous techniques on the area are neural networks, fuzzy logic, and evolutionary algorithms.

Search methods based on techniques such as evolutionary algorithms, simulated annealing, and tabu search are also referred as *metaheuristic* search and optimisation methods. According Portland Pattern Repository [PPR02], heuristic approach to problem is empirical method that often solves problem but cannot proof that given solution is the best possible. *Metaheuristic* methods are used to find good heuristics for different problems.

Portland Pattern Repository [PPR02] gives following example about the metaheuristic approach to the problem:

"What parameter do I use to get good results when applying heuristic method X to problem Y?"
"How do I adjust the parameters of heuristic X so I get better results on problem Y?"
"Which is better, heuristic X or heuristic Y?"

Different computational intelligence methods separately and in various combinations have been used to solve software engineering problems. [PP98]

1.5 Search-based software engineering

Harman and Jones [HJ01b] say that:

"Search-based software engineering is a reformulation of software engineering as a search problem, in which the solution to a problem is found by sampling a large search space of possible solutions."

According to Harman and Jones [HJ01a, HJ01b], metaheuristic search techniques, such as evolutionary algorithms, are applicable to many software engineering related optimisation and search problems because there are usually need to balance competing constraints and cope with inconsistency without any precise rules to compute the best solution. There are usually too many solutions without a perfect one, but still good solutions can be recognised from the bad ones.

Search methods with metaheuristic nature such an evolutionary algorithms, tabu search, and simulated annealing are used as sampling techniques. The most used search techniques in search-based software engineering are evolutionary algorithms including techniques such a genetic algorihms (GA) [Gol89], genetic programming (GP) [Koz92], evolution strategy (ES) [Rec73, Sch81], and evolution programming (EP) [FOW66]. Overview on various evolutionary algorithms techniques is represented in Chapter 2.

One challenge of search-based software engineering is to reformulate software engineering problems as search problems. Different methods based on evolutionary algorithms have been successfully applied to software engineering problems. For example, GP to automatic programming, GA to project scheduling, and test data generation.

There exists a research network called *Software Engineering using Metaheuristic Innovative Algorithms* (SEMINAL) [SEM02]. SEMINAL have the objective to demonstrate applicability of miscellaneous metaheuristic techniques to software engineering problems. This thesis will concentrate only to the use of evolutionary algorithms in software engineering.

2 Overview on evolutionary algorithms

This Chapter provides basic information and theory about evolutionary algorithms needed to understand the contents of this work. Three most widely used techniques - Genetic Algorithm (GA), Genetic Programming (GP), Evolution Strategy (ES), and Evolutionary Programming (EP) - are described briefly.

2.1 Evolutionary algorithms

Evolutionary algorithms are soft computing techniques inspired by Charles Darwins' "survival of the fittest" theory. The general idea is to have a population of solution candidates (individuals) for the problem. A problem specified evaluation function is used to measure the fitness of each individual. The first population can be initialized with random parameters, but usually all other populations after that contain individuals that derive "good" parameter values from their parent populations. Different kinds of crossover, mutation and stochastic operations are performed when creating new populations.

Evolutionary algorithms can be effective for finding global optimum of complex and non-continuous problems that are too difficult to solve using derivative-based methods. When using large and n-dimensional search space, evolutionary algorithm cannot always find the best possible solution in reasonable amount of time. However, solutions can still be "good enough" when compared to use of the brute force approach, which means the evaluation of all possible solutions. When the search space is large, brute force cannot be used because of so called combinatorical explosion that means too large a number of solution candidates to evaluate.

Evolutionary algorithms have been applied to many practical problems successfully. For example, mechanical shape optimisation [AL97], communications network design [KC93, KC97], finding potential customers from marketing database [Sun92], and optimisation related on electrical power systems [Yin93].

Next Sections (2.2, 2.4, 2.3, 2.5) present the most commonly applied techniques based on evolutionary algoritms. However, for each technique, there is a large number of different approaches and modifications and only the basics are described here.

2.2 Genetic algorithm

Genetic algorithms (GA), originally developed by John Holland [Hol75] are search algorithms based on the natural selection and natural genetics. The original goal for the research was to explain the adaptation of natural systems and to design artificial system that retains mechanism of natural systems. [Gol89]

The simulated evolution process in the genetic algorithm is done with a population of individuals represented by chromosomes. In practice, chromosomes are individual's parameters encoded to the string, bit or byte presentation.

Because often the first population does not have the final or "good enough" solution, there is need for keeping an artificial diversity in the population. Diversity can be maintained by using the crossover and mutation operations. The mutation is performed by applying a random change to the individual's chromosomes. A mutation can be complete or affect only few genes. In Figure 1 there is an example, where the mutation was performed for one gene. When using binary encoding, one practical way to perform mutation is to flip one bit in a chromosome. For example, a chromosome 1011 after the mutation can be 1001.

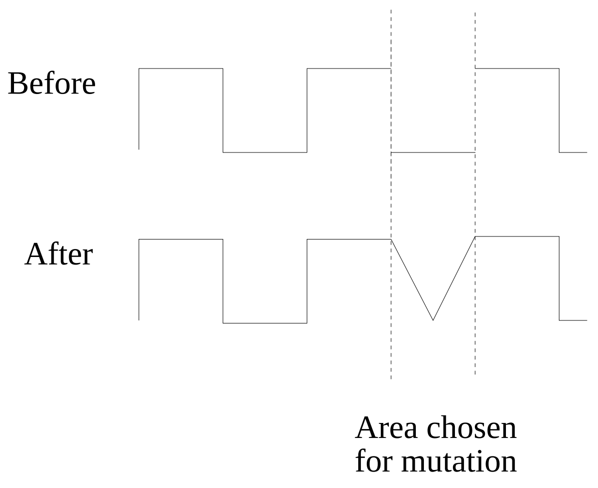

Figure 1: Mutation in Genetic Algorithm [Rya00]

The crossover in the natural evolutionary process means that child will inherit its properties (genes) from its parents. In genetic algorithms, the crossover operation is needed to mix and inherit good gene combinations from the current population to the new population. In Figure 3 there is an example of a crossover, where childs inherit their genes from the both parents.

Implementation of the genetic algorithm usually does following cycle [HB01]:

1. Evaluate the fitness value for all the individuals in current population.
2. Create new population by using crossover, mutation and reproduction operations.
3. Discard the old population and continue iteration.

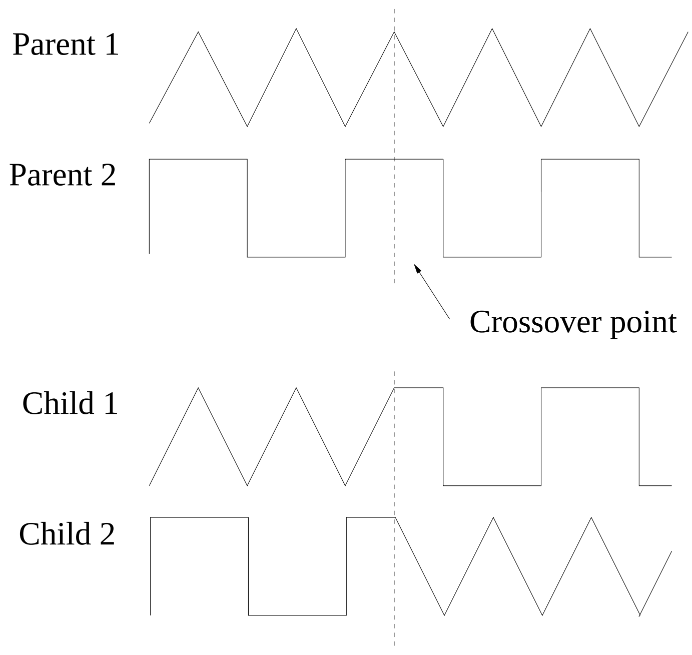

Figure 2: Example of crossover in the Genetic Algorithm [Rya00]

2.3 Genetic programming

Genetic programming (GP), introduced by John Koza [Koz92], has the objective to let a computer solve problems without being explicitly programmed to do so. Genetic programming can be seen as an extension of the conventional genetic algorithm. While the GA will generate a string that represents the solution, the GP generates a computer program as the solution. [Neg02]

High-level programming language called LISP was chosen as the main programming language for GP. Individuals in the population are variable length strings and each string will encode a candidate solution. Each solution is a parse tree that represents a computer program. Like GA, GP uses a fitness function to evaluate solutions. [HB01, Neg02]

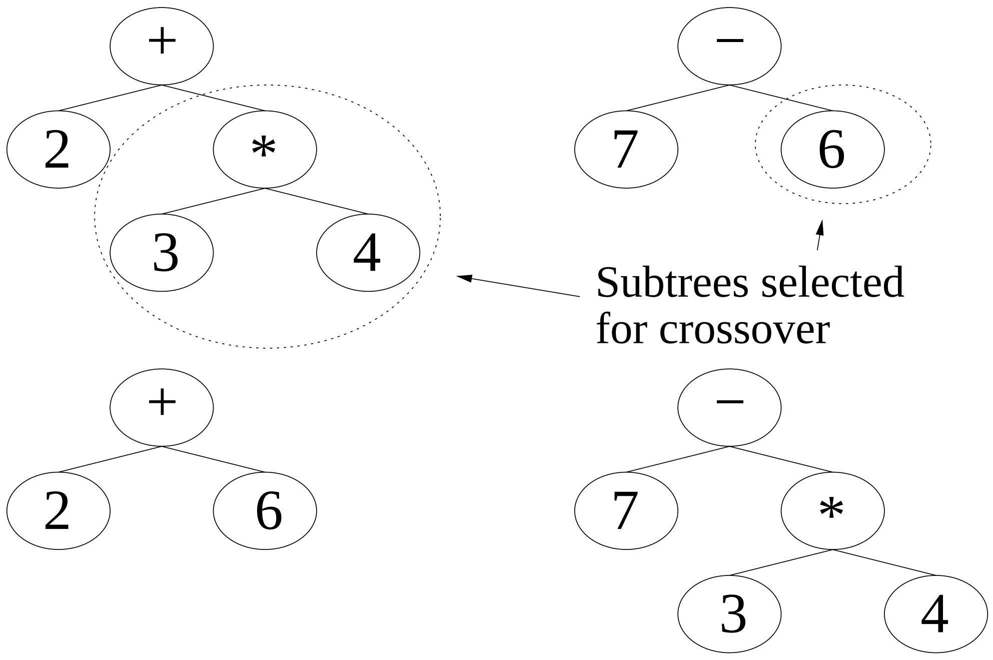

Figure 3: Crossover for two parent parse trees [Rya00]

GP uses the crossover operation, which will select two programs and produces two offspring programs. Offspring programs are recombinations of their parents with randomly chosen parts swapped between them. An example of crossover operation in GP is shown in Figure 3: there are two parse trees representing parent functions $2 + 3 * 4$ and $7 - 6$. The crossover produces two new parse trees with functions $2 + 6$ and $7 - 3 * 4$. [Neg02, HB01]

GP programs are usually composed of elements like fuctions and terminals. Mutation in GP can randomly change a terminal symbol to different terminal symbol or a function to another function. [Neg02]

2.4 Evolution strategy

Evolution strategy (ES) was developed to solve technical and mechanical optimisation problems and was previously only known in the civil engineering area. Evolution strategy can be used when no suitable objective (fitness) function exists or it is too difficult to use and no other suitable optimisation method exists. [HB01]

In the original ES, an individual (parent) in population generates one new individual (offspring) per generation. The process causes small mutations to occur more likely than larger mutations, until the offspring performs better than its parent. For example, the mutation operation can be based on normal probability distribution. Later on, ES was generalized to the form, which imitates the natural evolution process by using recombination, mutation, and selection. [Gol89, HB01]

2.5 Evolutionary programming

Evolutionary Programming (EP), introduced by Fogel *et al.* [FOW66] allows the use of different length individuals in the population. There is no crossover in EP, but instead the individuals selected as parents are subjected to the mutation to produce children. It has been said, that any structure, which can be mutated, can be evolved by using EP. [Rya00]

3 Classification of EA applications for solving software engineering problems

The software development process is usually divided into four major phases which are derived from the linear sequential model (Figure 4), also known as the waterfall model. [Pre01] The phases in chronological order are *analysis*, *design*, *implementation*, and *testing*. In this thesis, those four phases are used as rough classes, into which different EA based applications for solving software engineering problems are classified. Each EA based approach that is used to solve a particular software engineering problem is called as *an EA application* or *a case*. Classes will contain subclasses, which are also called *application areas*. For example, the testing phase contains separated subclasses for the structural testing and the functional testing.

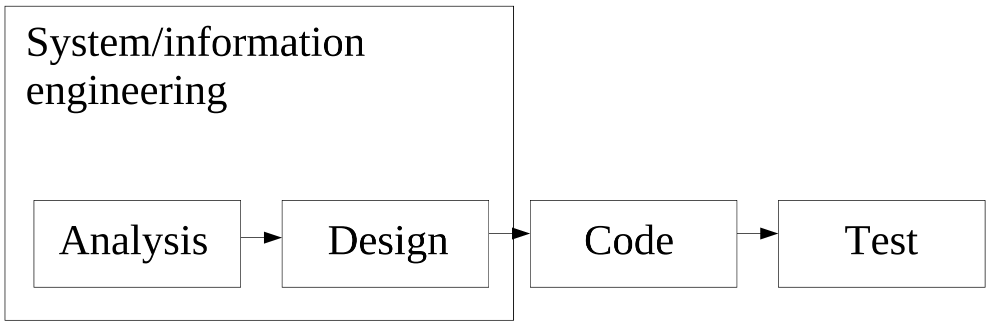

Figure 4: Linear sequential model, also known as waterfall model [Pre01]

The first half of the classes, the analysis and the design, represent the area of *system/information engineering* [Pre01], where the software under development does not exists and all software development tasks are pre-implementation tasks. The rest of the classes, the implementation and the testing are more concentrated on the software under development. The following four Sections (3.1, 3.2, 3.3, 3.4) will briefly describe each class.

3.1 Analysis

In a software development project, the analysis phase contains the feasibility study and the requirements analysis phases. In the feasibility study phase, highly generalised system level requirements are defined. Those requirements are only customer requirements and usually have no technical details about the system. Customer requirements are analysed and software requirements, needed in the requirements analysis phase, are derived from them. The requirements analysis is a gathering process that is intensified and focused specifically on the software itself. To understand the software being implemented later, the information about the required functionality, behaviour, performance, interfaces, and the application domain must be understood. [HM98, Pre01]

In this thesis, the analysis phase corresponds to the first class and is meant to contain the applications related to early software project tasks. Those tasks could be about the project management and design, project effort, and software quality prediction. The analysis class is about software project design rather than the software design, with which the next class is concerned.

3.2 Design

The second class is called design and concentrates on the software design, which is used to translate software requirements to a representation of the software. The software design process focuses on four attributes of a program: the data structure, the architecture, interface presentations, and algorithmic details. [Pre01]

This class will contain the applications, which are used to search for the software structure or the internal functionality. Also applications about the searching of optimal runtime organisation of the software are concerned, *e.g.* the resource and task allocation in the distributed system.

3.3 Implementation

Third class is called implementation and refers to implementation or code generation phase in the linear sequential model. In this phase, the previously produced software design is translated in to machine readable form - *e.g.* the computer program. EA applications which can be used to produce computer programs or support work of the human programmers, are classified in to this class.

Because genetic programming (GP) technique is meant only to generate programs, almost every use of the GP goes into this class. However, it is not appropriate, within the scope of this work, to report every use of the GP in the literature as an EA based application in the implementation phase of the waterfall model. That is why this class contains only cases which are closely related to the well known software engineering problems.

3.4 Testing

The testing phase refers to testing process that focuses on logical internals of implemented software. The objective is to uncover errors and to ensure that the program behaviour is as expected. Test results must coincide with the requirements. [Pre01]

All applications related to software testing are classified in to this class. Applications can be about test case generation or searching for program inputs which will cause failures or too long response times.

4 Analysis

This Chapter represents applications related to the analysis phase of the waterfall model. The related application areas are: prediction of software failures, exploring difficulty of the problem, software project effort prediction, and project management.

4.1 Prediction of software failures

Software failures usually occur after some implementation tasks have already been completed. If failures are found in testing phase, there will be need for extra implementation tasks, which will try to eliminate those failures. The extra work will usually cause development costs to increase. If failures are found after software is in use, the costs could go up even more. Because of this, it is important to find software failures in early phases of development.

Using special reliability techniques in the software development will require more resources, time and money. Because of this, it is important not to use them in every part of the system development. Quality prediction methods could be used to determine parts of the system to, which realibility techniques should be applied [EKCA98].

Evett *et al.* have described in the paper *GP-based software quality prediction* [EKCA98] the model based on GP, which can be used to find modules with high propability of failure and should be implemented with using reliability improvement techniques. The GP system will predict a number of faults the module is likely to produce. Number of faults will be used to rank the modules. Thus, final evaluation of quality will be based on ordinal criteria rather than number of failures.

There exists a software quality data from software development projects. Data set has attributes for unique and total number of operators and operands, lines of source code, lines of executable code, and two values for cyclomatic complexity (*McCabe's cyclomatic complexity* and *extended cyclomatic complexity*, number of linearly-independent paths throught a program [MB94]). Modules are classified as *fault-prone* and *not fault-prone*. [EKCA98]

The data was splitted to training and validation data sets. The GP system was trained by using training data. For every run, the best program was returned as the result. The ability of the trained program to generalise the data was measured by using the result program to predict faults also for validation set. After that, best-of-run programs were ordered accordingly. The usefulness of the model is evaluated by its ability to approximately order modules from the most fault-prone to the least fault-prone. The order was compared to random order of modules. In this case, random order results were really different than GP results. [EKCA98]

Even best GP result programs are not expected to predict failure module perfectly. According to Pareto's Law, 20% modules have 80% of total number of failures (in this case, over 70% of modules has near zero failures) and developers should be interested in those top 20% of modules in order. [EKCA98]

In the paper *Automated Knowledge Acquisition and Application for Software Development Projects* Baisch *et al.* [BL98] represented how to express tailored fuzzy expert system capable of classifying software modules by their propability of having errors. Quality evaluation was interpreted as the transformation of different software metrics to a quality factor, which means correctness or maintainability of the software. Because rules consist of a complex combination of different metrics, Baisch *et al.* propose to use GA to assist human experts to define rule-base for classification.

Every individual in to population represents one fuzzy expert system encoded as a binary string. Fitness function was based on the contingency-table like weighting according to false and correct classification data. Modules were classified to two different classes: Modules with fault amount of 0 to 5 and more than 20 faults. Data was splitted in training and test sets. Extracted expert system was able to classify correctly 87.8% of training data (409 modules) and 82.4% of test data (301 modules). System was also tested with real-world application. When applied to top 10% modules with high amount of faults, the prediction of failures reduced total number of late faults in top-10 down with 14%. [BL98]

Liu *et al.* in the paper *Genetic programming model for software quality classification* [LK01] have used GP to classify software modules. Liu *et al.* have used real world datasets from large software written in C++. The used dataset was about 807 software modules classified as fault-prone or not fault-prone. Dataset has attributes for a number of times the source code was inspected, number of LOC for different production phases and final number of commented code. System was made to run faster than interpreted LISP by having pointers to C functions in nodes of parse trees.

Data was splitted into two sets (training and validation). At first, GP model tries to predict number of faults for each module. Then it classifies modules. Fitness was calculated from number of hits and raw fitness, which comes from pre-defined *cost of misclassification* values. Built GP model has misclassification rate of about 20% and it was compared against regression model, which has a rate of over 30%. In conclusions of the paper, the GP was said to be more accurate in this case. [LK01]

Bouktif *et al.* discuss in the paper *Combining Software Quality Predictive Models: An Evolutionary Approach* [BKS02] the problem caused by different software quality prediction models and that there are not many companies that systematically collect and publish related data. Usually data is confidential and only summaries are published. For a solution to this problem, Bouktif *et al.* propose the use of existing models as independent experts. [BKS02]

In an approach to this problem, it was tried to combine decision trees into one final classifier system with GA. When the input vector was given to the system, decision making starts from the root of the tree and corresponding edges are followed. Questions in nodes can be something like *Is* $x < \alpha$ *?*. [BKS02]

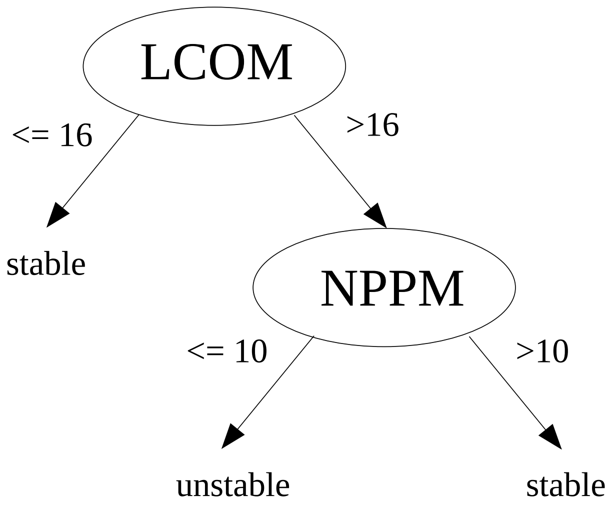

Figure 5: Example of a decision tree for the software stability prediction [BKS02]

Chromosome representation of trees was made with set of boxes with sides parallel to the axes. In Figure 5 there is a simple decision tree for software stability prediction with OO related variables LCOM (lack of cohesion methods) and NPPM (Number of Public and Protected Methods in a class). Same tree in a box representation is in Figure 6. [BKS02]

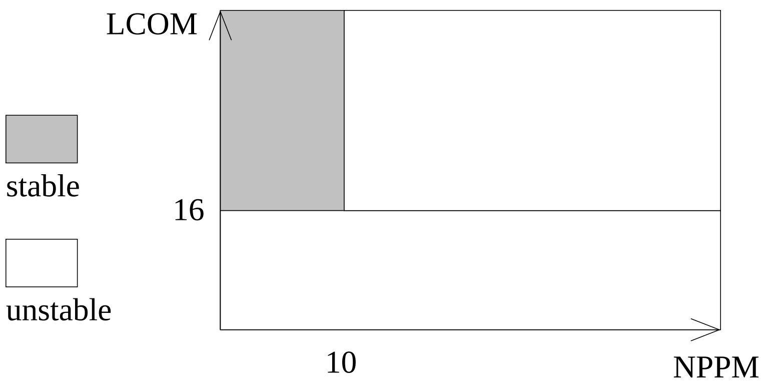

Figure 6: Example of output regions for decision tree for software stability prediction [BKS02]

$$J(f) = \frac{1}{k}\sum_{i=1}^{k}\frac{n_{ii}}{\sum_{j=1}^{k} n_{ij}} \qquad (1)$$

Fitness was calculated by using Youden's *J-index*, which calculates the average correctness per attribute. Function is represented in Equation 1 where n_{ij} is the number of training vectors with real attribute c_i classified as c_j. [BKS02]

According to Bouktif *et al.* [BKS02] The system was tested by predicting the stability of classes written in Java. 22 structural software metrics about coupling, cohesion, inheritance, and complexity were selected for attributes. Data was randomly split into 10 datasets with equal size and then the classifier is trained with 9 sets. The last one dataset was used as test data. This was repeated with all 10 possible combinations. After test experiments, it was concluded, that resulting *meta-expert* model can perform significantly better than individual models.

GA is used to train ANN to predict software quality in the papers *Using the genetic algorithm to build optimal neural networks for fault-prone module detection* [HKAH96] and *Evolutionary neural networks: a robust approach to software reliability problems* [HKAH97].

Other related papers are *Software quality knowledge discovery: a rough set approach* [RPA02] and *Using genetic programming to determine software quality* [EKA99].

4.2 Exploring difficulty of the problem

During early phases of a software development project, developers usually do not have an exact awareness about the problem, which should be solved using the software. Developers increased uncertainty about the real nature of the problem could increase cost and duration of the project. It is commonly said, that bad decisions occuring in early phases of project will lead to problems, which will be expensive to eliminate later. One way to avoid increased development cost is to collect more knowledge about the problem to be solved with the software.

Feldt in the paper *Genetic Programming as an Explorative Tool in Early Software Development Phases* [Fel99] (available also in *Biomimetic Software Engineering Techniques for Dependability* [Fel02]) has presented the idea about using genetic programming to explore the difficulty from input data. The term *software problem exploration using genetic programming* (SPE-GP) is proposed to mean this approach.

The target system is software, which controls arresting of landing aircraft on a runway. This is commonly used in military aircraft carriers, where the length of the runway is limited. There is a cable on the runway, which will catch the incoming aircraft. The controlling system has to apply suitable pressure to drums of tape attached to cable for smooth stop. The system has to stop the aircraft close as possible to target distance without exceeding physical limits (*e.g.* length of the cable or tape and maximum retarding force applied to the pilot and the aircraft) [Fel99, Fel02]

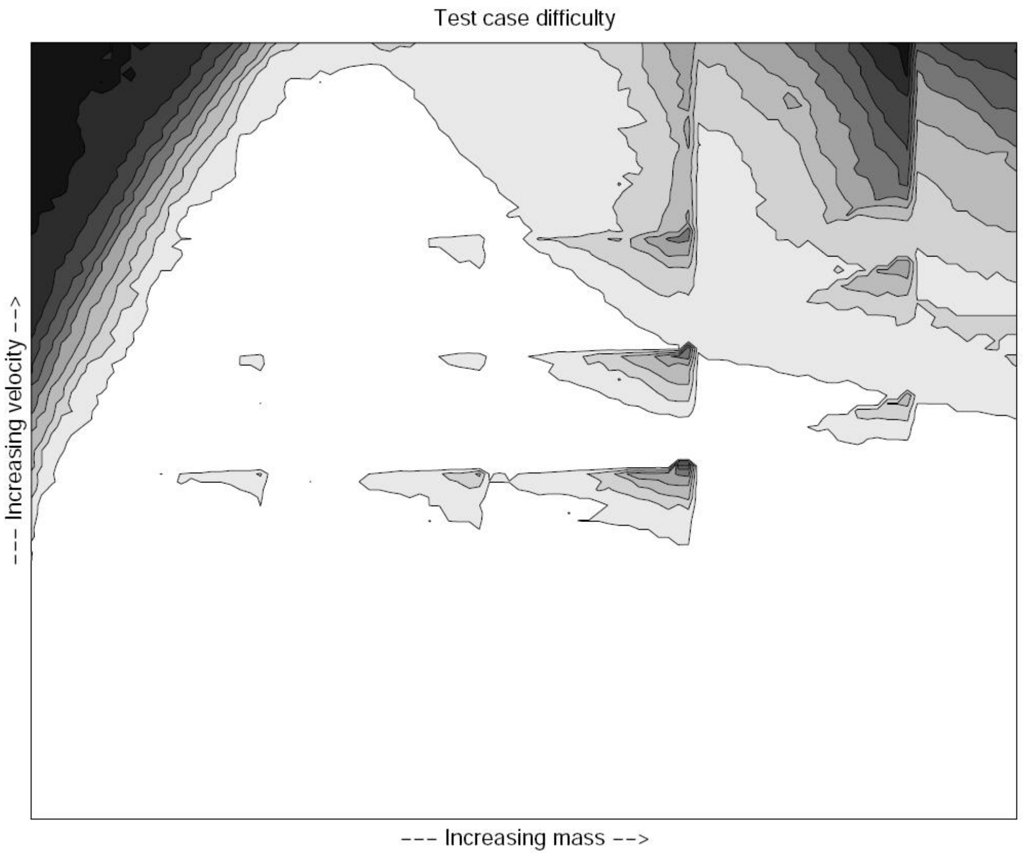

Figure 7: Difficulty of test cases as function of mass and velocity [Fel99]

Values from the simulation are used to assign penalty values on the four fitness criteria. All penalty values are summed to the total fitness of the test case, therefore this is a minimisation problem and zero fitness is the global optimum. Used GP system was built on top of GPSys genetic programming system. After each run, the best program tree was evaluated with 10000 test cases with different aircraft mass and velocity values. If any physical limit was exceeded, failure was recorded. [Fel99, Fel02]

Difficulty of the test case is a proportion of programs with failures. Test case difficulty with different masses and velocities is shown in Figure 7 where dark area means higher difficulty. In original requirements, there was defined maximum hook force for certain points with specified mass and velocity. Because of this, there is separated areas with higher difficulty in the middle of the Figure 7. In addition, there was discussion on the paper, which critised the usage of the low-dimensional input space and lack of research about effect of altered requirements to difficulty. Feldt says also, that SPE-GP technique includes the risk, that result data will be GP-specific rather than human-specific. [Fel99, Fel02]

4.3 Software project effort prediction

In the early days of computing, the cost of the software itself was only a small part of the total cost of an information system. Nowadays software is usually the most complex part of the system and developing it may be expensive when compared to other costs related. [Pre01]

Cost and effort estimation for a software development is not very exact science. Larger number of variables affects the total cost of software [Pre01]. Because of this, many computational intelligence methods have been applied to estimate cost.

In the paper *On the problem on the software cost function* [Dol01] Dolado has tried to use classical regression and GP for publicly available data sets to find software cost functions. Both methods were used because classical regression makes assumptions about the distribution of data and GP can explore data without assumptions. There were 12 different sets with software product size (*e.g.,* LOC or FP values) as independent variable. As result, significant good predictions attained solely by the product size was not found. Dolado has also compared GP, ANN and regression methods for software cost estimation in the paper *Limits to the Methods in Software Cost Estimation* [Dol99].

Burgess *et al.* in the paper *Can genetic programming improve software effort estimation? A comparative evaluation* [BL01] have trained GP system to predict effort of software projects and compared results to usage of other methods. Used data was taken from 81 projects from a Canadian software house and had attributes about developers/managers experience, development environment, year of completion, and 5 other attributes about software size and complexity. The dependent attribute was effort (person-hours). The data was splitted into training and *query* (test) sets with ratio 63/18.

Different methods were used to predict efforts: random, linear LSR, 2/5 nearest neighbours, ANN and GP. Errors were measured with different functions. ANN has superior performance and GP seems to be able to provide accurate estimates. Authors believe, that after their results, further investigation is needed. In conclusions it was said, that ANNs and GPs give good accuracy but they will need more effort to setup (training). [BL01]

Shan *et al.* have used almost same approach in the paper *Software Project Effort Estimation Using Genetic Programming* [SMLE02]. The Grammar Guided GP (GGGP) was used to fit a model. In GGGP, the objective is to find good production rules of grammar, which are used to produce the actual program.

```
effort = size * (if application in{dss,missing} then 5.34 else 1.68)
         + 18.5 * team_size * log(size) + 92.7 * log(size)
```

Figure 8: The example of GP program [SMLE02]

According to Shan *et al.* [SMLE02], Data set of 423 software projects was splitted to training and test sets (211/212). For each project, there was high number of different attributes: 32. There were numerical and non-numerical attributes, *e.g.* team size, size (in function points), business area type, usage of object-oriented techniques and usage of different project management methods. In the Figure 8 there is an example of GP program.

As a result, it was observed that in best programs there were no used non-numerical attributes in best-run programs, only numerical. Shan Y. *et al.* suggests, that non-numerical attributes are too complex to be discovered by the GP or not closely related to the project effort. [SMLE02]

Neural networks trained with GA is also used to estimate efforts in the papers *Neuro-genetic prediction of software development efforts* [Shu00].

Other related papers are *An evolutionary approach to estimating software development projects* [ARRRT01] and *A validation of the Component-Based Method for Software Size Estimation* [Dol00].

4.4 Project management

Project manager has time, resources, and a goal for the project. The goal is reached when all the project tasks are completed. Each task needs resources (like employees) and time to be completed. There is a limited number of resources for use and project must be completed before the pre-defined date.

Usually resource usage and task completion order can be planned by using timeline chart. In Figure 9 there is an example of timeline chart (also known as Gantt chart) for the project of six tasks and three employees. Horisontal bars are tasks and lines between them are dependencies: *e.g.*, tasks 2 and 3 cannot be started before task 1 is completed.

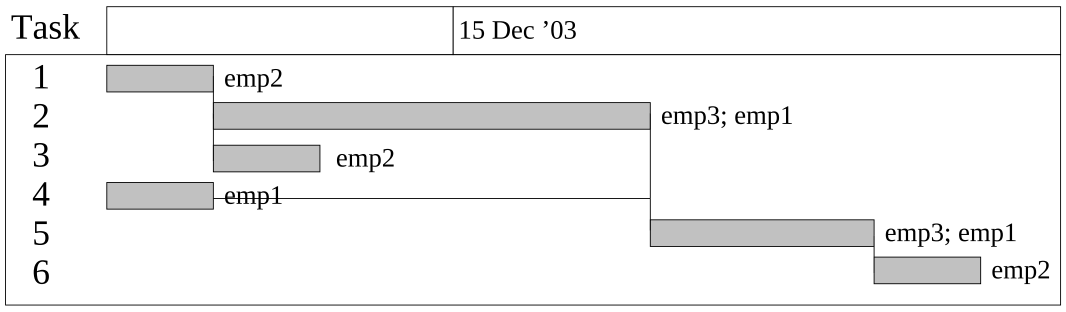

Figure 9: Gantt chart for the project with three employees and six tasks.

Chang *et al.* [CCNC98] have developed a formal model called *Software Project Management Net* (SPMNet) to model software development projects. Model has an automatic resource allocating and a scheduling based on GA. Projects and tasks needed to complete are representated by using Task Precedence Graph (TPG). In Figure 10 there is an example of TPG with ten tasks. Each task has values for man month (MM) and skill required (SR).

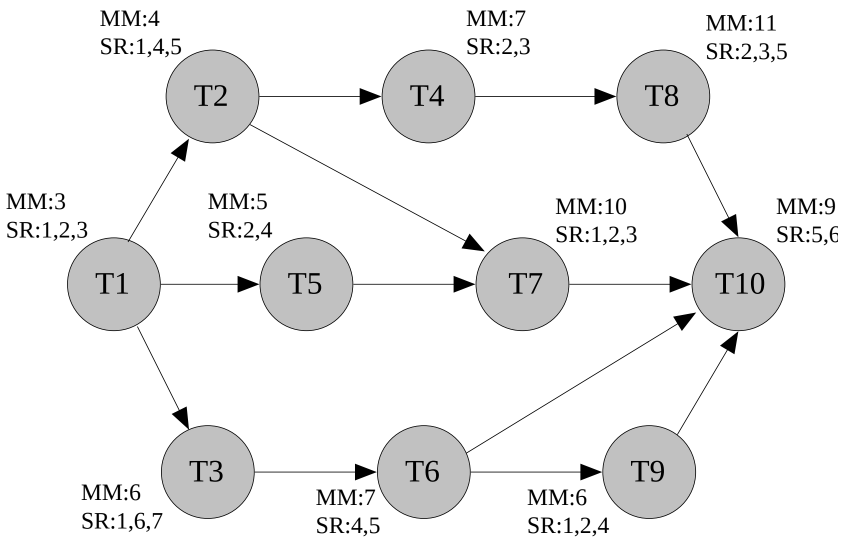

Figure 10: Example of Task Precedence Graph. (MM: man month, SR: skill required) [CCNC98]

The GA system takes TPG and employee/skill database as an input and outputs a near-optimal schedule. The project schedule itself is used as a chromosome and it is a string, which tells the order how employeers will participate to tasks. GA-based scheduling was compared against exhaustive search (which generates a set of all possible solutions and returns the best one) and GA was able to find optimal solution in reasonably shorter time. [CCNC98]

Later in the paper *Genetic Algorithms for Project Management* [CCZ01] Chang *et al.* have represented improved GA-based project scheduling method. New method uses 2D array, which allows many to many and one to many relations between tasks and employees. A C++ library of GA components called *GAlib* was used. The approach have similarities to previous research [CCNC98], where GA takes TPG and employee database (skills and salary) as a input. Individuals are 2D arrays with employeers enumerated along the rows and tasks along the collumns. The population diversity is increased by using several operations, like *flip*, *destruction*, and *swap* in mutation. In Figure 11 there is an example of used 2D array crossover. Also the support for multiple projects and partial commitment (*e.g.*, employees A and B can take part to the task with ratios 0.3 and 0.7) is added. [CCZ01]

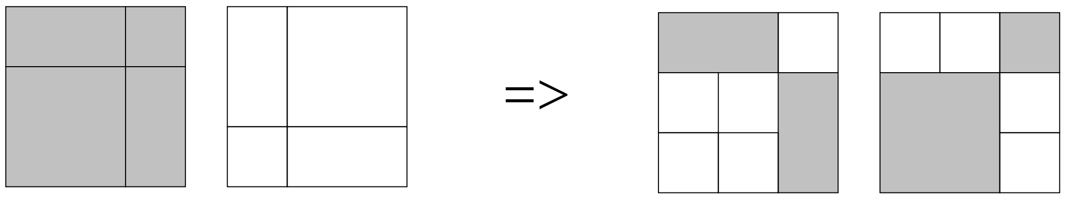

Figure 11: Example of the crossover of two 2D arrays [CCNC98]

The objective function is composition of four objectives:

- *Validity of job assignments (Validity).* If all skill requirements for the tasks are satisfied then $Validity = 1$, otherwise 0.
- *Minimum level of overtime (Overload).* Amount of over time for all employees.
- *Minimum cost (CostMoney).* Labor costs of the project. Calculated by using labor rates of each resource and the hours used for the tasks.
- *Minimum of time span (CostTime).* Maximum time span, in which the project must be completed.

The used composite objective function needed to maximise is:

$$fitness = Validity * (W_1/OverLoad + W_2/CostMoney + W_3/CostTime) \quad (2)$$

The objective function (2) allows usage of weigts w_n for component objectives. In this case, the $Validity$ component with value 0 will cause large penalty for the whole function. GA system was tested with test problems and was superior when compared to exhaustive search. Also the ability to set weights for the components of the objective function was said to be very usefull. [CCZ01]

Chang *et al.* are going to continue research on this area. They will suggests integration of some cost model (like *COCOMO*) with the constrains for better relationship with time to complete the task and number of people involved in it.

In the paper *Projecting Risks in a Software Project through Kepner-Tregoe Program and Schedule Re-Planning for Avoiding the Risks.* Komiya *et al.* [KH00] have proposed the method for avoiding risks in software projects through *Kepner-Tregoe* program. Kepner-Tregoe is logical method for problem solving and decision making. The used method itself was project task scheduling with the GA.

5 Design

This Chapter represents applications related to the design phase of the waterfall model. The related application areas are: multiprocessor scheduling, task and resource allocation in distributed systems, hardware/software co-design in embedded systems, protocol construction, and architecture design.

5.1 Multiprocessor scheduling

A multiprocessor system is a set of two or more processors, which communicate with each other. A parallel program is set of tasks to be executed under a number of constrains. In static multiprocessor scheduling, tasks can are scheduled to the processors before the execution or during runtime. If tasks, which had been scheduled for different processors, communicate during their execution, it will cause message passing and waiting, which will slow down the execution. Usually the goal in static multiprocessor scheduling is to minimise the total expected runtime of tasks. [CFR99, JPP00]

A dynamic multiprocessor scheduling will occur on runtime and is widely used in modern operating systems. All scheduling, dynamic or static, is also referred as *load balancing*.

A directed acyclic graph (DAG) can be used to represent computer program. Nodes are tasks and arcs between them are data dependencies. Weights in arcs means cost caused by message passing and weights in nodes representing computational cost. [DAA95]

Dhodhi *et al.* say that efficient assignment and scheduling of parallel program tasks are important in the effective utilisation of multiprocessor system [DAA95]. In the paper *A Multiprocessor Scheduling Scheme Using Problem-Space Genetic Algorithms* [DAA95] they represent a GA based technique, which can be used to increase resource utilisation and therefore to reduce completion time of the parallel program.

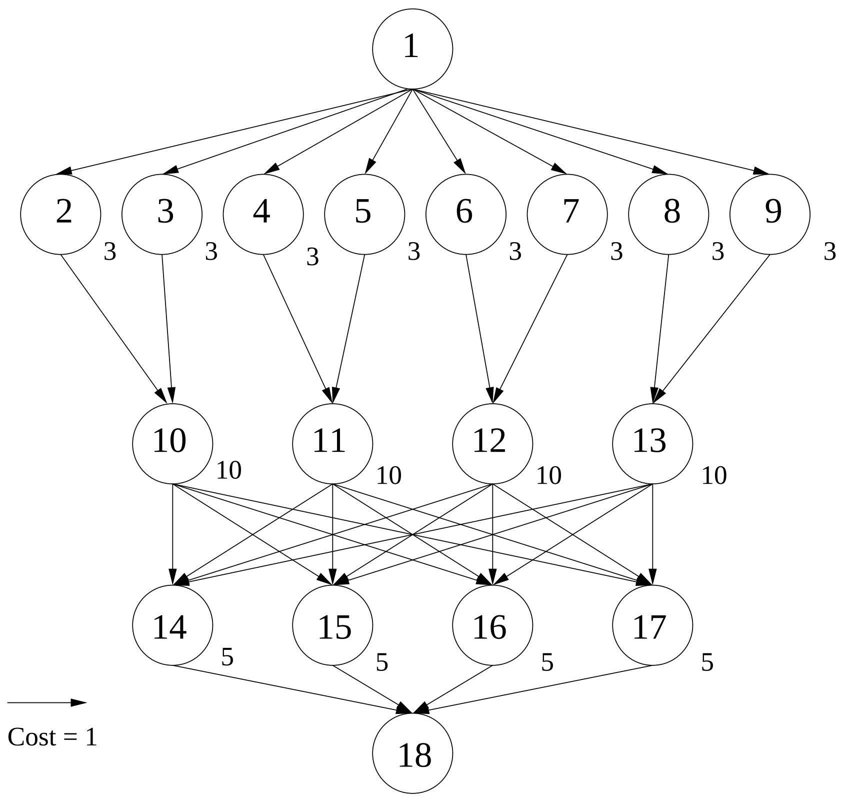

Figure 12: Directed acyclic graph (DAG). Nodes are tasks and arcs are data dependencies between them. [RL90]

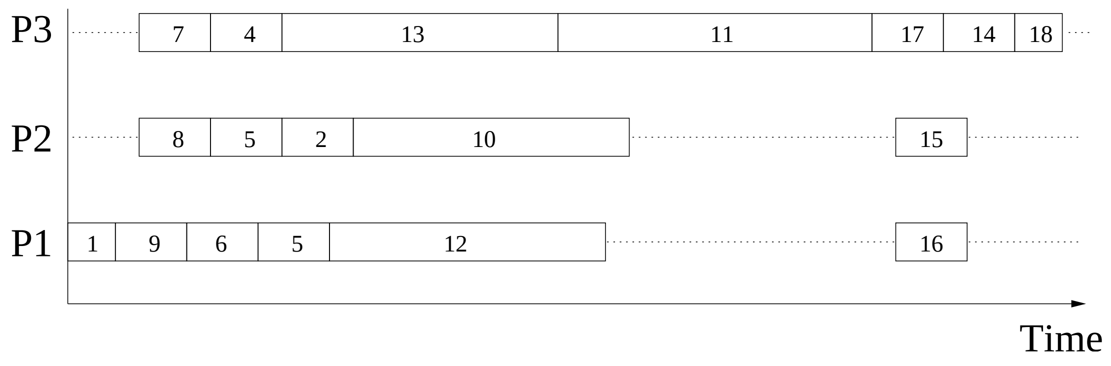

Figure 13: Task allocation for three processors (Pn) [DAA95]

According to Dhodhi *et al.* [DAA95], GA was used to search good assignment of tasks of the program to different processors. DAG (Figure 12) is used to represent a program. The task execution can start after all the data it needs have been received from previous tasks for, which task has dependency. In this case, the communication cost is always zero between tasks assigned to the same processor. If tasks are in different processors, the weight of the arc between their nodes means communication cost. In good task assignment, every processor has a task to execute and total execution time of program is minimal.

Individual (Figure 14) is a list, which index corresponds with DAG node numbers. The list itself contains task priority values. The tasks without previous dependencies are placed to the new task list. Then a task with highest priority is allocated to the available processor on, which the start time of the task with dependencies is the earliest. This is repeated until there are no tasks to allocate or no idle processor. A task cannot be allocated to processor until tasks it depends are completed and communication from them has occured. Figure 13 shows complete task allocation for three processors (Pn). [DAA95]

Priority	20	19	19	19	19	19	19	19	19	16	16	16	16	6	6	6	6	1
Node	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Figure 14: GA individual, with list having node (task) priorities [DAA95]

Fitness value to be minimised is calculated by taking MAX value from the set of completion time of processors. The completion time includes computational, communication and waiting times. [DAA95]

Experimental results for this technique were compared against examples from literature and results were very promising. It was concluded, that proposed technique was able to reduce completition time and increase processor utilisation. [DAA95]

In the paper *Scheduling using Genetic Algorithms* Ursula Fissgus [Fis00] has used GA to find good execution orders for tasks. DAG was replaced by *extended module graph* (EMG), which has separated structural and data dependencies between modules. In EMG, the structural dependency between modules A and B means that module A must be executed before B. Data dependency means, that module A will produce data to module B as a parameter. [Fis00]

Each individual represents an execution order of tasks and also versions of tasks. Each task has one or more versions implemented using different functions. Fitness is measured by using total execution time of the program. In conclusions part of the paper, it was said that simultaneous exploitation of task and data parallelism and ability to choose between different implementations of tasks can lead to faster programs. [Fis00]

Qi-Wei Ge in the paper *PARAdeg-processor scheduling for acyclic SWITCH-less program nets* [Ge99a] has used GA for multiprocessor scheduling problem for data-flow program nets (PN). Data flow program is run by the idea of data-driven processing, which can be used to avoiding memory access bottlenecks. PN differs from DAG, *e.g.* by allowing to visit in one node several times.

Tsuchiya *et al.* in the paper *Genetics-based multiprocessor scheduling using task duplication* [TOK98] represents a GA approach to multiprocessor scheduling problem, which uses task duplication. If tasks are executed on different processors, a message passing between the processors will cause delay. Task duplication means scheduling with copies of the task in different processors, which can eventually reduce delay caused by communication.

According to Tsuchiya *et al.* [TOK98], the individual is set of lists. Each list represents a processor and had tasks assigned to it. Order of tasks in list is also execution order. An example of representation is in Figure 15. Each individual must satisfy following three conditions [TOK98]:

1. Every task is allocated to at least one processor.
2. No task is allocated to one processor more than once.
3. For any task, none of its predecessors are executed after the task on the same processor.

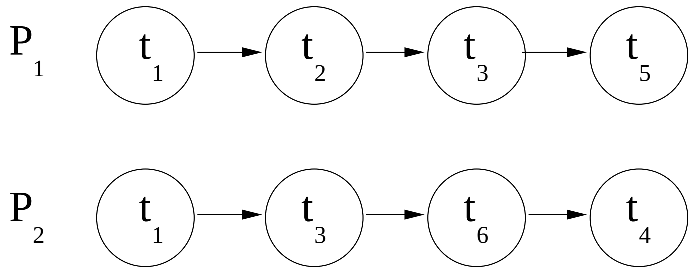

Figure 15: Tasks scheduled for two processors (P_n) with duplicated tasks t_1 and t_3 [TOK98]

Fitness value f is defined in Equation 3 where L_{max} is the maximum length from all schedules in population and $length(S)$ is the length of the currently evaluated schedule. [TOK98]

$$f = \left(\frac{L_{max} - length(S)}{10} + 1 \right)^2 \tag{3}$$

Method was compared against Duplication Scheduling Heuristic (DSH) and as result it was observed that when the communication delays are small, the GA was able to find better schedules than DSH. Otherwise, both methods had almost the same performance. [TOK98]

Jung *et al.* had used ordered-deme GA (OGA) for multiprocessor scheduling in the paper *An Ordered-Deme Genetic Algorithm for Multiprocessor Scheduling* [JPP00]. In OGA, a globally sorted population is divided to the sorted array of subpopulations.

The average fitness of each subpopulation is maintained in a stepwise manner and converted to one value. This method can improve search capability because search takes place independently in each subpopulation. Scheduling with OGA was compared to GA and was performed better in 13.4% of all the task graphs.

Yi-Hsuan *et al.* in the paper *A Modified Genetic Algorithm for Task Scheduling in Multiprocessor Systems* [LC03] has used partitioned genetic algorithm (PGA) for multiprocessor scheduling. PGA integrates the idea of *divide and conquer* to partition the entire problem space into subgroups, solve them individually, and merge them to the solution of the problem.

At first, task graph is partitioned in to subgroups, which are scheduled using GA. The schedule itself is represented as list of tasks. After subgroups are scheduled, they are combined to the final solution. A single schedule is represented as a list with execution order of tasks and number of processor to which task is scheduled. [LC03]

T_0	T_2	T_4	T_1	T_3	T_{11}	T_9	T_7	T_5	T_{10}	T_8	T_6
1	0	2	1	1	2	0	1	2	0	1	2

Figure 16: A single schedule with task numbers (T_n) and number of processor to, which the task is allocated [LC03]

According to Yi-Hsuan *et al.* [LC03], in experiments, it was noticed that PGA was usually able to find better solutions with lesser time than the original GA.

In the paper *Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors using A Parallel Genetic Algorithm*, Yu-Kwong and Ahmad [KA97] represents a Parallel Genetic Scheduling algorithm (PGS), which can be used for multiprocessor DAG scheduling. The PGS itself is a parallel algorithm, which can cause scheduling to be faster. In experimental study, PGS has found optimal solution for half of the cases.

There is more research about using the GA in (static) multiprocessor scheduling: *An implementation of the linear scheduling algorithm in multiprocessor systems using genetic algorithms* [BC00], *Genetic algorithm approach towards scheduling DAG on multiprocessor* [YQN01] and *A two-processor scheduling method for a class of program nets with unity node firing time* [Ge99b], *Pareto-based soft real-time task scheduling in multiprocessor systems* [OBWK00] *Fast scheduling and partitioning algorithm in the multiprocessor system with redudant communication resources* [Las01], and *Scheduling Multiprocessor Tasks with Genetic Algorithm* [CFR99]. In addition, a cellular automata [Ser98] was also used to design multiprocessor task schedules.

5.2 Task and resource allocation in distributed systems

In distributed systems, all the information and functionality is usually allocated to different nodes of the system. Usually one node needs access to its own resources but also resources in other nodes. Communication with other nodes causes delay in execution and can decrease overall reliability of the system.

Sanjay Ahuja in *A genetic algorithm perspective to distributed systems design* [Ahu00] has developed a GA based framework for allocating data files to the nodes of the distributed computing system (DCS). DCS is represented as graph where nodes are DCS nodes and each node can have programs and copies of data files. Arcs between the nodes are connection links, which can have reliabilities and capacities. For every program, it is known how much traffic is required to be transferred during execution.

The GA takes DCS graph, values for link reliables and capacities, program allocation in nodes, files needed for each program, and traffic required to be transferred when the program is executed. The goal is to maximise $ADPT(P_k)$, which means Average Distributed Program Throughput of program P_k. The function is defined in Equation 4 where α is the traffic delivered across the system and α_{Pi} is traffic requirement of the system for program Pi. [Ahu00]

$$ADPT(P_k) = \frac{\alpha}{\alpha_{Pi}} \tag{4}$$

The file allocation (individuals) is coded to binary numbers. In DCS with four nodes, the single file to allocate had $N_1 N_2 N_3 N_4$ where binary number for N_j tells if file F_i is present in node N_j. GA run is terminated when the average and maximum $ADPT(P_k)$ equals, number of generations reach the maximum or all individuals in the population are same. The GA was compared to exhaustive search and it was concluded that GA was usually able to find optimal solutions using less computational time. [Ahu00]

Grajcar has used *genetic list scheduling* algorithm, based on list scheduling and GA, to assigning tasks to processors in coupled and heterogeneous system. Research is represented in the paper *Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System*. [Gra99]

Kumar *et al.* in the paper *Genetic algorithm based approach for file allocation on distributed systems* [KPG95] have done earlier research on GA based file allocation in DCS where approach was very similar to [Ahu00].

Ahuja and Kumar have used GA to allocate programs and data into nodes of distributed systems in *A genetic algorithm approach for performance based reliability enhancement of distributed systems* [AK94].

Jaewon Oh *et al.* have represented a GA based model for allocation of objects into distributed system in the paper *A formal model for allocation of objects into heterogeneous distributed environments* [OCW00].

Kim and Hong in the paper *A task allocation using a genetic algorithm in multicomputer systems* [KH93] had used GA to allocate program modules for processors in multicomputer systems. Method and results are very similar to other research where GA is used for task scheduling in multiprocessor systems (Section 5.1 Multiprocessor scheduling).

There are also other papers about task scheduling (in multiprocessor systems), which are related mainly to multicomputer/distributed systems: *Object-oriented simulation and GA* [Ram01], *Genetic algorithm based data and program partitioning* [SNYF00], *Genetic scheduling algorithms in distributed computing systems* [WYKH97], *Efficient allocation of program modules on multicomputers* [BA94], *Building a retargetable local instruction scheduler*, and *Improved Static Multiprocessor Scheduling using Cyclic Task Graphs: A Genetic Approach* [SM98].

5.3 Hardware/software co-design in embedded systems

There is need for real-time embedded system, which operate reliably and predictably. To achieve these requirements, usually hardware and software designers will co-operate. This integrated design approach is called *hardware/software co-design*. [SC94]

Korousic-Seljak and Cooling have proposed a GA based co-design technique in the paper *Optimization of multiprocessor real-time embedded system structures* [SC94]. When a multiprocessor real-time embedded system is being designed, this technique can be used to search optimal node structure and task allocation.

The system is a set of computing nodes. Nodes will perform tasks and are connected to each other with I/O devices. The system kernel has scheduler, which is responsible for allocating ready-to-run tasks to processors. The GA is used find schedule where all tasks are allocated and completed before their deadline time is expired. If there is no schedule where no deadlines are expired, some tasks must be rejected. In real-time system design, decisions must be taken between the criticality and timeliness. [SC94]

	t_1	t_2	t_j	t_n
P_1	0	0	$\cdots$	1
P_2	1	0		0
P_i			$\vdots$	
P_m	0	1		0

Figure 17: An encoded individual with processors P_n and tasks t_n [SC94]

The individual (Figure 17) is encoded to an array where first dimension corrensponds to the processor number P_n and second to ready tasks t_n. Numbers 0 and 1 indicates if the task is allocated to the processor or not. In this approach, there was no dependencies or communication between the tasks. [SC94]

Grajcar in the paper *Conditional scheduling for embedded systems using genetic list* [Gra00b] has used GA based *genetic list scheduling algorithm* to search for good task schedules for the real-time system. Task execution is assumed to be conditional, which means that executed tasks and their start times depends on previous task executions. The idea of using genetic list scheduling algorithm on multiprocessor task scheduling was represented in the paper *Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System* [Gra99], which was introduced in Section 5.2.

Semeraro has used GA to find near-optimal system configurations for real-time operating system in poster paper *Evolutionary approach to real-time analysis* [Sem98].

Other articles about using GA in embedded systems hardware/software co-design are *Optimization of multiprocessor real-time embedded system structures* [YG02a], *System level software/hardware partitioning by genetic algorithm* [YG02b], *Heuristics to optimize the speed-up of parallel programs* [Agu96], and *A hierarchical genetic algorithm for hardware software co-synthesis with a stochastic approach* [CRC02].

5.4 Protocol construction

Generally a protocol stack has a fixed number of layers. Every layer is implementation of some protocol functionality. Construction of static end-to-end protocol to meet all communication requirements could be difficult and this complexity could reduce protocol performance. Dynamically configurable protocol stacks have used to construct protocols, which will meet its requirements with minimal overhead. [Gra00a]

Grace has used GP for building communication protocols in his master's thesis *Applying Genetic Programming to Protocol Construction* [Gra00a]. Idea was to decompose protocols to micro-protocols and then with GP to compose them to form a complete protocol, which is light as possible, and satisfies the requirements. Used GP system was GPsys and protocols were constructed by using JavaGroups, a Java-based protocol framework toolkit.

Because of the used toolkit, each protocol stack was represented in string format:

```
<prop1>(arg1=val1):<prop2>(arg1=val1;arg2=val2):<prop n>
```

In the string, there are numbered properties (`prop1`, `prop2`...) separated with colons, each property represents one protocol layer. Arguments can be passed to the each layer: `arg1=val1` means, that value (`val1`) is assigned as argument number 1 (`arg1`). In Figure 18 there is protocol stack `UDP:NAKACK:UNICAST:FLUSH:GMS` represented as GP program tree, where each node represents one layer and only operator is colon (`:`), which separates layers. [Gra00a]

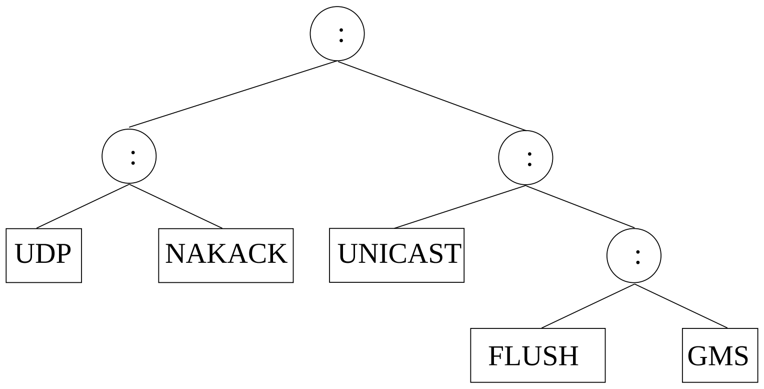

Figure 18: Protocol stack `UDP:NAKACK:UNICAST:FLUSH:GMS` represented as GP program tree [Gra00a]

Fitness for a single protocol layer was measured with three tests: testing semantic viability, how stack meets the communication requirements and quality of service provided by the protocol. After experiments, it was concluded, that the system was able to generate correct protocol stacks in hours and results were highly depending on the size of the population. Future work was proposed to improve performance of GP system and to allow system to be tailored each particular search. [Gra00a]

In the paper *Protocols are programs too: the meta-heuristic search for security protocols* [CJ01] Clark and Jacob have used GA and simulated annealing to generate correct and efficient Burrows, Abadi and Needham (BAN) protocols.

5.5 Architecture design

Software design is usually splitted in to two parts: *Architecture design* and *module design*. In architecture design, system is being clustered to modules and subsystems and their interfaces are defined. In module design, each module and its internal functionality is being designed. In architecture design, it is usually tried to develop system's parts as independent as possible and keep number of connections and dependencies between subsystems small as possible. [HM98]

Doval *et al.* in the paper *Automatic clustering of software systems using a genetic algorithm* [DMM99] represents *software clustering GA*, which can be used to good partition of the module dependency graph (MDG). MDG is a directed graph which is used to describe the modules of the system and their relationships. In Figure 19 there are an example of simple MDG. Figure 20 shows a partitioned MDG and in Figure 21 there is MDG with best possible partitioning: there is many intra-connections inside of the subsystems and only one inter-connection. [DMM99]

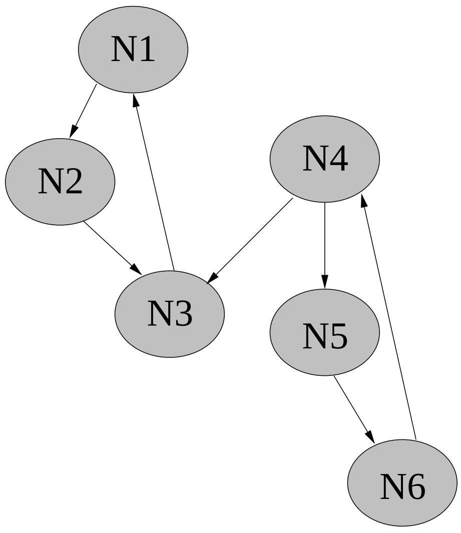

Figure 19: A simple module dependency graph [DMM99]

According Doval *et al.* [DMM99], individuals in the population are strings with character index pointing to identifier of module and character itself is identifier of the cluster to, which the current module belongs. For example, the string *1 1 2* means MDG with three modules; first two modules belongs to the first cluster and third module to the second cluster.

The *modularization quality function* (MQ) is used to measure the fitness of MDGs. The module intra-connectivity means density of dependencies between the modules of the single cluster (subsystem). Intra-connectivity A_i for cluster i is defined by the Equation 5 where N_i is number of components and μ_i is the number of module dependencies inside of the cluster. Therefore N_i^2 is a maximum number of possible dependencies. [DMM99]

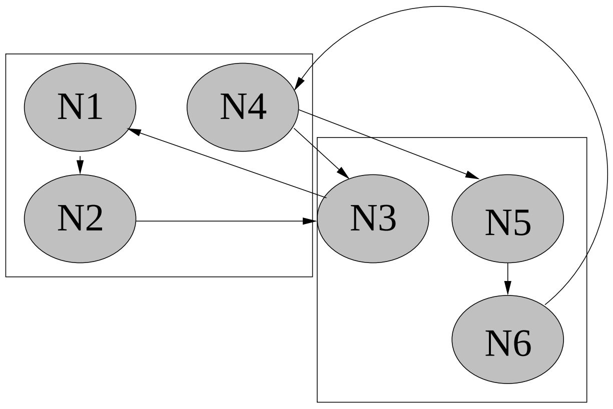

Figure 20: A partitioned module dependency graph [DMM99]

MQ (Equation 6) represents a tradeoff between inter- and intra-connectivity by subtracting average inter-connectivity from the average intra-connectivity. In MQ, k is number of clusters and $E_{i,j}$ is inter-connectivity between clusters i and j. [DMM99]

The method has been tested with the source code of Mini-Tunis operating system. Mini-tunis is well-designed and documented software with 20 modules. With population of 10 individuals and 200 generations, the best MDG was very similar to the original partition of Mini-Tunis. Method has some problems with the library and interface modules and also global library modules were located to the cluster where the most of the callers of them were located. Earlier research on this was represented in the paper *Using Automatic Clustering to Produce High-Level System Organizations of Source Code* [MMR+98].

$$A_i = \frac{\mu_i}{N_i^2} \tag{5}$$

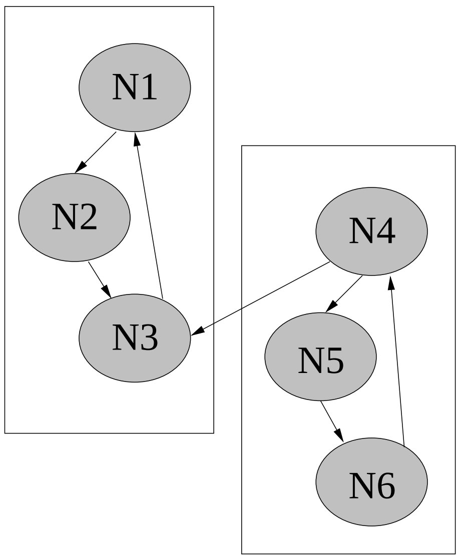

Figure 21: A best possible partition of module dependency graph [DMM99]

$$MQ = \begin{cases} \frac{\sum_{i=j}^{k} A_i}{k} - \frac{\sum_{i,j=1}^{k} E_{i,j}}{\frac{k(k-1)}{2}} & k > 1 \\ A_i & k = 1 \end{cases} \tag{6}$$

In the paper *A New Representation and Crossover Operator for Search-Based Optimization of Software Modularization* Harman *et al.* [HHP02] have developed a new representation for a software modularisation but also introduces new GA crossover method, which is suitable for the software modularisation problem.

An individual is represented by having components and enumerated modules in lookup table. A module contains components and every component contains functions and variables. A functions can have a dependency to another function or variable. The problem is to find a graph with maximum number of dependencies inside each subgraph and minimum number of dependencies between subgraphs. [HHP02]

The component number one is always in module number one and all components in the same module as component number n are always in module number two. This sorting is continued by selecting lowest unallocated component as the defining element of the module. This will allow to reduce size of the search space and avoiding many-to-one mapping in individual representation. [HHP02]

Wadekar and Gokhale in the paper *Exploring Cost and Reliability Tradeoffs in Architectural Alternatives using a Genetic Algorithm* [WG99] have explored cost and reliability tradeoffs of different software architecture alternatives with GA. This tradeoff can be important when resources are highly limited and the minimum level of required reliability is defined.

According to Wadekar and Gokhale [WG99], the expected reliability for software system was calculated from reliabilities of individual modules and expected number of times the execution visits in modules. The cost of the whole software is calculated by multiplying the reliability and cost of the each component. Each GA individual is a list containing elements, which corresponds to the modules of the software. Each element has reliability and cost values. A fitness function is defined as:

$$fitness = \frac{-K/lnR}{C\gamma} \tag{7}$$

In function 7, γ is factor, which can be used to linearise the variation of cost, K is large constant $\neq 0$, R is software reliability and C is whole software cost (sum of all module costs). Three case studies were performed and it was concluded, that the method can be used to found satisfied software reliability with minimised cost when using in-house developed and *off the shelf* components. Optimal or near-optimal solution was always found. [WG99]

6 Implementation

This Chapter represents applications related to the implementation phase of the waterfall model. The related application areas are: automatic programming, N-version programming, search for compiler optimisations and re-engineering.

6.1 Automatic programming

Evolutionary algorithms, especially GP can be used to generate computer programs to fulfill the given software specification. Because original GP technique is meant only to generate programs, almost every use of GP goes to the area of automatic programming. Because in practice it is not possible to report every automatic programming case from the literature where GP is used, this Section will concentrate only on cases, which are closely related to well known software engineering problems and tasks.

Bruce in the paper *Automatic generation of object-oriented programs using genetic programming* [Bru96] has developed extensions for GP to allow generation of object-oriented programs. Bruce says that main differences between procedural and OO programming are the use of the inheritance in types (classes) and using global memory during runtime to store states of objects beyond the execution of an individual part of the program.

An indexed memory is used to represent the memory associated with an object. Read/write operations are provided in order to allow construction of class methods. Memory structure and methods can both be inherited from another object. Memory (variables) inheritance can be implemented by using terminal symbols or functions, which allow memory access to class to be inherited. It is possible to implement method inheritance by using functions, which invoke methods from the inherited classes. [Bru96]

Two different fitness functions are used. First compares actual and desired return values of the methods, second compares actual and desired pre-invocation and post-invocation object memory. The space of constructable programs is limited by allowing only programs with required language elements to be constructed. [Bru96]

During experiments, GP programs was generated until there was a program that perfectly matches the training data (desired program). Desired programs were implementation of three data structures; stack, queue, and priority queue. Methods were `Init`, `Empty?`, `Full?`, `AddDataItem`, and `RemoveDataItem`. Experiments were performed with and without the strongly typed GP, using first, second or both fitness functions and by using methods simultaneously or individually (sequential). [Bru96]

According to Bruce [Bru96], the strongly typed GP (STGP) was developed, because the original GP needs all variables, constants, and arguments to be of the same data type. In STGP, data type for each value can be specified beforehand. Initialisation process and genetic operators will generate only syntactically correct program trees [Mon93]. The GP was able to construct correct solutions, but there was some problems. Experiments with strongly typed GP and individually used methods produced best solutions.

Petry and Dunay in the paper *Automatic programming and program maintenance with genetic programming* [PD95] represents an GP approach to automatic programming. According to Petry *et al.*, Turing machines are generated with the GP to solve simple problems. After the problem is solved, the solution is encapsulated and it becomes part of the library. GP itself can use programs from library to solve new problems. System keeps library valid and efficient. If old problem is solved more efficiently, the program in library is updated.

6.2 N-version programming

N-version programming (NVP), introducted by Avizienis in 1977 [AC77] means the independent generation of two or more functionally equivalent program versions from same requirements. Multiple versions of the same program can be used to improve fault-tolerance in case, where only some versions fail independently.

Feldt in the paper *Generating Multiple Diverse Software Versions with Genetic Programming* [Fel98b] (and *Generating Diverse Software Versions with Genetic Programming: an Experimental Study* [Fel98a]) introduces the approach for generating multiple software versions from same specifications using GP. The idea was to let some parameters passed to the GP system to be vary, because of was believed that changing GP run parameters will cause GP to converge on different areas of search space. Following parameters can be an examples of varying parameters: Maximum depths for program and mutation trees and list of allowed functions and terminals.

According to Feldt [Fel98b], phases of the proposed method are following:

1. The design and implement fitness function by using software specification.
2. Decide, which GP system parameters will be varied.
3. Decide how parameters will vary.
4. Choose the parameter value combinations to use in GP runs.
5. Let GP system to run for each combination of parameter values.
6. Measure fitness for each generated GP program. Calculate the diversity.
7. Select the program combinations with lowest failure probability to the software fault tolerance structure.

The diversity for GP programs is calculated from the different parameter values passed to GP system for each run. The diversity values between GP programs can be used, when there is need, *e.g.*, to select GP programs to the program pool of the NVP system. [Fel98b]

To test this, an experimental environment based on aircraft arresting problem (which is described in Subsection 4.2) was developed and pool of controller software versions was generated. 435 versions of software was created and tested. It was found, that the probability for coincident software failures is decreased when diversity between software versions is increased. [Fel98b]

6.3 Search for compiler optimisations

A compiler is commonly used to compile high level source program, written by human programmer, to low level machine-readable target program. A modern compiler can perform different kinds of optimisations to the program, but it is not usually clear, what optimisation(s) should be applied with a particular source program.

Cooper *et al.* in the paper *Optimizing for Reduced Code Space using Genetic Algorithms* [CSS99] have used GA to minimise size of compiled computer program using different optimisations.

Used compiler compiles C and FORTRAN source code to assembly-level intermediate language called ILOC. There were 10 different optimisations, each are able to modify given ILOC code in some way. The objective is to find ordered sequence of optimisations to apply, which will transform ILOC code to the form where the final executable is as small as possible. The optimised ILOC code is compiled to C code, which is compiled to executable and executed. [CSS99]

Optimisation methods were mapped to letters, and the GA individual is a string with letters. For example, c means *cprop* method and s means *coalesce*. One possible optimisation sequence, beginning with *cprop* could be $cnottcdtvooc$. [CSS99]

Experiments were performed with different C and FORTRAN programs and best optimisation sequences for the different programs were found. The sizes of executables were significantly reduced after best optimisation sequences were found and applied. When looking the best individuals, it was found that there was some optimisations, which were part of best individuals for each program. The fixed sequence based on their observations were formed, and it was applied to programs causing 40% smaller and 26% faster executables. [CSS99]

Nisbet represents the *genetic algorithm parallelisation system* compiler framework in the paper *GAPS: A Compiler Framework for Genetic Algorithm (GA) Optimised Parallelisation* [Nis98]

6.4 Re-engineering

It is usual, that a particular software has served the needs for years and it was several times corrected, adapted, and enhanced. During that time, developers usually does not apply good software engineering practices because of other matters. This will lead to the situation, where software is unstable. It works, but almost every time a change is attempted, unexpected effects will occur. [Pre01]

Conor Ryan in his Book *Automatic re-engineering of software using genetic programming* [Rya00] has described the need for automatic software re-engineering tools and has applied GP to different re-engineering tasks. The book will concentrate mainly to program auto-parallelisation problem, which involves re-writing programs to execute on a parallel system (*e.g.*, in multiprocessor computer).

Byung Jeong Lee *et al.* in the paper *Implementation of reusable class library based on CORBA using genetic algorithm* [LMW99] have used GA to clustering components to the reusable library. The clustering tries to find an optimal component grouping considering the number of clusters and different kinds of similarity values.

In the paper *Genetic algorithm based restructuring of object-oriented designs using metrics* [LW02], Byungjeong-Lee and Chisu-Wu had used GA to restructure OO design. Fitness was measured using cohesion and coupling metrics.

Lutz in the paper *Recovering high-level structure of software systems using description length principle* [Lut02] has used GA to find good decompositions of software systems represented as graphs.

7 Testing

This Chapter represents applications related to the testing phase of the waterfall model. The related application areas are: functional testing, structural testing, integration test design, testing based on mutation analysis, and searching for response time extremes. There is also Section for miscellaneous applications.

7.1 Functional (black box) testing

Black box testing is also known as *functional testing*. The idea is to use test cases which are created by using software requirements without knowledge about the internal structure of the software. [HM98]

Schultz *et al.* in the paper *Test and Evaluation by Genetic Algorithms* [SGDJ93] have used GA to find fault scenario combinations, which will cause failures in autonomous-vehicle controller software. Target system is a software controlling aircraft landing.

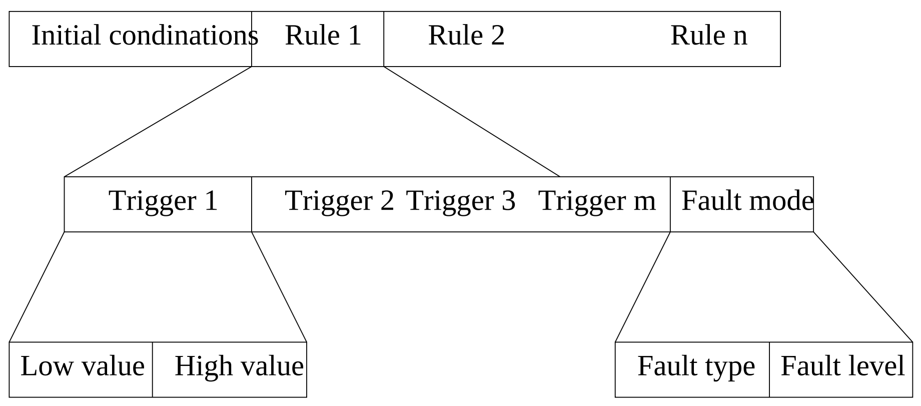

Figure 22: Representation of the fault scenario [SGDJ93]

Fault scenario is a description of faults, that are possible to occur in software. Fault scenario (Figure 22) has initial conditions and set of fault rules. Variables like wind speed, wind direction and velocity are initialised in initial conditions part. A single fault rule has two parts, set of triggers and fault mode. Each trigger measures some part of the current states of vehicle and environment. Each trigger has also low and high values, *e.g. 13 <= position[y] <= 832* is trigger, which is satisfied if Y-position matches the given value range. If all triggers in the rule are satisfied, the fault mode is instantiated. A fault mode has two parts: fault type, which describes the part of system where the failure has occured and the fault level, which measures the severity of the failure. [SGDJ93]

The GA is used to search for interesting failure scenarions, which will cause failures in vehicle simulator. The objective is to find a large number of interesting failure scenarios rather than finding the "best" one. A single individual is a fault scenario. Diversity of initial population is increased by initialising value ranges to be wider. The GA was stopped, when failure scenarios had reached predefined level and therefore the final population is highly diverse set of fault scenarios. In addition, other methods were discussed. [SGDJ93]

Current fault activity for fault scenario is calculated from activated rules using assigned fault level (Equation 8). Total fault activity for whole mission is the sum of current fault activity values divided by time (Equation. 9). The fitness function (eq. 11) uses a score value, which is given for each landing based on its quality (1 means crash and 10 means perfect landing) (Equation 10). If crash occurs without failures, the fitness function will return 1, which is maximal possible value. [SGDJ93]

$$current\ fault\ activity = \prod_{activity\ level} ((|fault\ level| * 9.0) + 1.0) \tag{8}$$

$$fault\ activity = \frac{\sum_{time} current\ fault\ activity}{time} \tag{9}$$

$$score = \begin{cases} 1 & \text{if crash landing} \\ 2 & \text{if abort} \\ 3 \rightarrow 10 & \text{if safe landing} \end{cases} \tag{10}$$

$$eval = 1/(fault\ activity * score) \tag{11}$$

After experiments, some fault scenarios generated by GA were shown to the designer of the controller software. The designer noticed that there is some improvements needed in some part of the software. In particular, a set of failed failure scenarios have been seen as classes of weaknesses, which allows developers to improve reliability of software. [SGDJ93]

In the paper *Testing control software using a genetic algorithm* [Hun95] Hunt has used the GA to test a car cruise control system software. Aim of the system is to maintain the speed of the the car by moving the throttle as necessary. The system is aware about speed of the car and also positions of the brake and clutch switces. A driver can set the required speed, switch whole system or only the cruise controlling on/off. The software was developed specifically for to this research only.

According to Hunt [Hun95], each individual is a binary string with 14 numbers. First seven are *On/Off Switch, Activate/Deactivate Switch, Speed Not_set/Set, Brake Switch On/Off*, *Traction Control Output On/Off*, *Clutch Switch On/Off* and *Throttle Position Increment/Decrement*. Last seven binary numbers represents a 7 bit value (0-127) of current speed.

Fitness of each individual is evaluated by giving six first bits from the individual as an attributes to the system. As an output, the system gives throttle position (increment/decrement), which is compared to desired outputs from software specifications. If values match, then a fitness value 2 is assigned to the individual. Otherwise the fitness value will be a difference between desired and actual outputs. [Hun95]

In this research the attempt is to find large number of interesting faults rather than just one fatal. When the final generation has been produced, the GA system will output all individuals with fitness value above 2 as an list of interesting failures. After experiments results were compared against random search, it was concluded, that used GA testing is effective and can aid human tester to understand about the needed set of tests. [Hun95]

Kasik and George have used GA to generate novice user events for testing a GUI-based application in the paper *Toward automatic generation of novice user test scripts* [KG96]. The method was based on notion, that novice users often use applications in very unpredictable way and they learn the application by performing different tasks. By emulating behaviour of a novice user requires the method to remember previous successes. The GA, rather than pure random search can be used to achieve this.

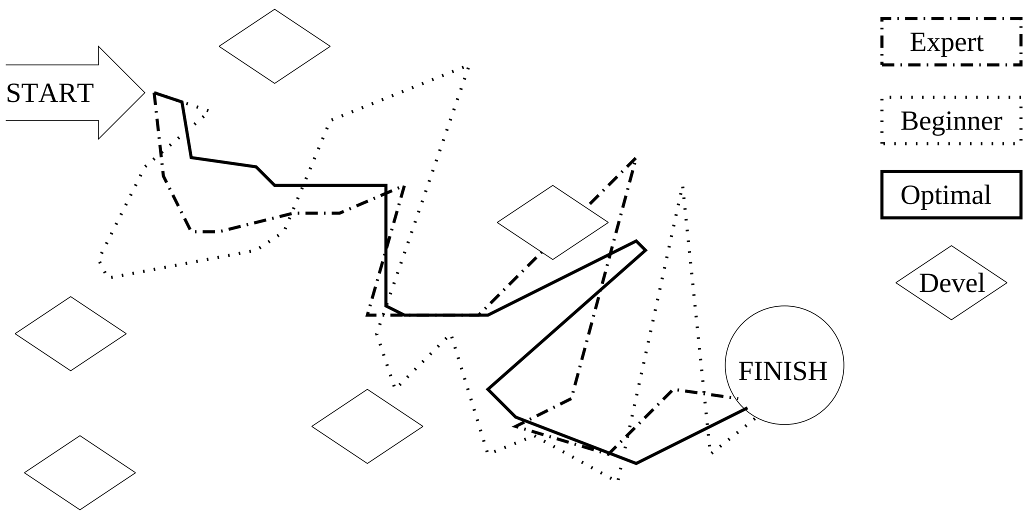

Figure 23: Different users with paths throught the application [KG96]

In Figure 23 there is three different possible paths to travel during system tests:

- An optimal path lets the user to perform task with shortest time and fewest steps.
- The expert user knows enought about program states, and are able to work effiently.
- The novice user wanders in a unpredictable manner. [KG96]

The widget is an object in X Window System, which associates window and related functionality. The *widget tree* is tree structure which, corresponds to the way how widgets in the X application are related to each other. For example, window can have two buttons and both buttons can have labels. [KG96]

Each individual is evaluated by restarting the application and importing related widget tree to the GA system. An active widget is searched from the tree using using random search. Depending on the active widget type, the next gene from the individual is given to the application as an input. [KG96]

The fitness calculation is based on presumption that novice user learns application with a controlled exploration. A novice user starts with one functionality, and after experiments with different inputs, user will move to second functionality. The fitness is calculated by initialising all user events, except one, to have weight 0. A gene will receive a positive score each time, when it causes input for a widget that have same window name as the last active window name. This causes individuals in a population to stay longer on the same window. [KG96]

According to Kasik and George [KG96], the developed method can be used to find application failures before the beta tests or production. Method was said to work best with a companion the automated test tools and expert test scripts.

Patton *et al.* in the paper *A Genetic Algorithm Approach to Focused Software Usage Testing* [PWW93] have used GA to search and identify regions, which causes failures from software input data. The represented technique allows to isolate failure clusters, which helps developers to focus on failures that are most severe and likely to occur for the user.

Other related papers are *Applying genetic algorithms to software testing* [XES$^+$92], *The automatic generation of test data using genetic algorithms* [Wat95], *Testing software using order-based genetic algorithms* [BM96], *Parameter estimation of hyper-geometric distribution software realiability growth model by genetic algorithm* [MT95], *The Automatic Generation of Test Data Using Genetic Algorithms* [Sth95], *Suitability of evolutionary algorithms for evolutionary testing* [WBS02], *Research on automatic testing of a kind of software by using generalized genetic algorithm* [YZC99], and *Breeding Software Test Cases with Genetic Algorithms* [BFJ$^+$03].

7.2 Structural (white box) testing

White box testing is also called as structural testing or glass box testing. Used test cases are created by using information about structure of the implemented program. [HM98] There is usually large number of different paths through the program, *e.g.* 100 lines of the C language with two nested loops can have about 10^{14} different program paths. It is not usually possible to systematically test every possible program path [Pre01].

Smith and Fogarty in the paper *Evolving software test data-GA's learn self expression* [SF96] have generated software test input data using GA. The objective was to produce data, which maximises the test coverage and minimises size of the test data. The triangle problem was used as a test problem for the method. The test problem program takes three integers as inputs (triangle sides a, b and c) and outputs either "scalene", "isosceles", "equilateral" or "not a triangle".

In the paper it was discussed, that there are two suitable metrics to measure the coverage for test inputs. Metrics are the Statement Coverage (the percentage of program statements to be executed during the execution of the tests) and the Branch Coverage (percentage of possible branches through the program taken). [SF96]

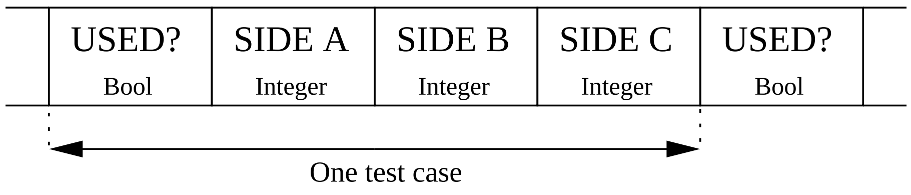

Figure 24: A single test case for the triangle problem [SF96]

According to Smith and Fogarty [SF96], the test case is represented as a fixed size string containing integer values. In Figure 24 there is an example of string representation of GA individual. In the string, each test case begins with boolean flag, which tells, if following integers should be ignored or taken as a part of actual test data. This allows to represent variable length string using fixed length string. The size of the test data was tried to minimise using following function (Equation 12), where set_size is the size of the test data and $maximum\_set\_size$ is maximum allowed size:

$$fitness = Coverage(\%) + \left(1.0 - \frac{set\_size}{maximum\_set\_size}\right) \tag{12}$$

The method was tested with branch coverage metrics using a different GA parameters and it was easy to achieve 100% branch coverage for test problem. Also other test coverage metrics were used with different results. [SF96]

Pargas *et al.* in the paper *Test-Data Generation Using Genetic Algorithms* [PHP99] have used GA to generate software test data. Control-flow Graph (CFG) is used to represent how program is executed. In graph, all nodes are statements and edges corrensponds to the flow of control between statements. For example, an edge between two statements means, that an execution of program can flow from the statement to another statement. Control-Dependence Graph (CDG) is used to represent dependencies between the program statements.

In Figure 25 there is a program example (on left), CFG, and CDG for it. According to Pargas *et al.* [PHP99], the numbered nodes corresponds statements on the program. T (true) and F (false) are marked for arcs with conditional statement. For example, in Figure 25 the statement 2 is `if` (i < j) and if the statement is TRUE, the execution goes to arc with T.

The CFG in Figure 25 is interpreted as follows:

- Node 6 postdominates all other nodes except exit.
- Node 2 postdominates nodes 1 and entry.
- Node 1 postdominates entry node.
- Node 4 is control dependent on 3T.
- Node 5 is control dependent on 3F.
- Nodes 1, 2 and 6 are control dependent on entry node.

```
        Program Example
        integer i, j, k
1.      read i, j, k
2.      if (i < j)
3.         if (j < k)
4.            i = k
           else
5.            k = i
           endif
        endif
6.      print i, j, k
        end Example
```

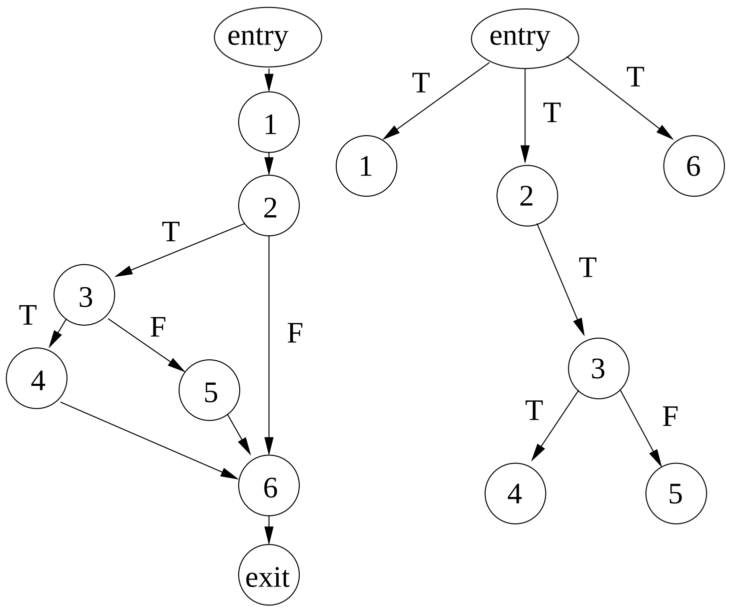

Figure 25: An example program with control flow-graph (CFG) (left) and control-dependence graph (CDG) (right) [PHP99]

A path in CDG from the root node to another node is called a control-dependence predicate path. For example, the set of predicates entryT, 2T, and 3T forms a control-dependence predicate path for statement 4 (Figure 25). [PHP99]

A function GenerateData was used to generate test data. GenerateData takes the program, CDG, and test requirements as an input and with GA it begins to search test data, which will satisfy the requirements. An individual is single test data, which is a set of input values for the program. [PHP99]

Fitness of each individual is evaluated by executing the program with current test data inputs and recording the predicates in the program that executes with input test data. List of executed predicates is compared to list of predicates found on the control-dependence predicate path for the node corresponding the target test requirements. The fitness value for current test data comes from the number of predicates, which are common for the control-dependence predicate path of target. The higher number of covered predicates means higher fitness value for the test data. [PHP99]

In the Table 1 there is population of four test cases for program in Figure 25. Test cases t1 and t2 are covering all statements except statement 5, therefore they have higher fitness than test cases t3 and t4.

Table 1: Test cases with statement traces for program in Figure 25 [PHP99]

Test case	Input data i, j, k	Statement trace
t1	1, 6, 9	1, 2, 3, 4, 6
t2	0, 1, 4	1, 2, 3, 4, 6
t3	5, 0, 1	1, 2, 6
t4	2, 2, 3	1, 2, 6

When testing programs with little complexity, the used method and random search had found full statement coverage immediately. When testing programs with some complexity, the method outperformed the random search. The method was said to be powerful approach in the area of automatic test-data generation. [PHP99]

Michael *et al.* in the paper *Genetic Algorithms for Dynamic Test Data Generation* [MMSW97] have used GA in automatic test data generation. Objective was to generate test data, which satisfieds condition-decision coverage (CDC) metrics. CDC is satisfied, when each condition in program is at least once true and false. In addition, each control branch from the program must be executed at least once. GA based method was compared against the results with random test data. With some test programs, like bubble sort and euclidean greatest common divisor, results were identical. With rest of the test programs, GA showed better performance than random test data.

In the paper *Generating software test data by evolution* [MMS01] Michael *et al.* introduces a GA based test generation system called GADGET (Genetic Algorithm Data Generation Tool), which can be used to test large C and C++ programs. Like the previous research [MMSW97] GADGET also uses CDC metrics to evaluate the test data.

In the paper *Identification of Potentially Infeasible Program Paths by Monitoring the Search for Test Data* Bueno and Jino [BJ00] have proposed a GA based technique to generate test data and to detect infeasible program paths.

Computer programs are represented as CFG and a single individual is input data for the program. Function (Equation 13) is used to calculate the fitness Ft. NC is path similarity metrics, which means a number of coincident nodes between the executed path and the desired path. EP is the predicate function value deviation (error) between the executed path and desired path. MEP is the maximum predicate function value from the set of individuals, where each individual has executed the same nodes of the desired path. [BJ00]

The search has objective to find solution (input test data) with the maximal number of correctly executed nodes and minimal value of the predicate function for the reached predicates. [BJ00]

$$Ft = NC - \left(\frac{EP}{MEP}\right) \tag{13}$$

It is possible, that program path has predicate, which cannot be reached and there is a test data which will try to reach it. The search monitoring with two heuristics was used to detect possible infeasible path. The best individual in population is continuosly monitored, and if fitness does not increase enough, a counter is incremented. Tester can use value from this counter to determine how persistent is the possible lack of search in progress. Technique was tested with different test programs and was able to identify all infeasible paths while generating test input data for programs. [BJ00]

Other related papers are *Automatic Testing From Z specifications* [Xil98], *Automatic test data generation for program paths using genetic algorithms* [BJ02], *Integer- and real-value test generation for path coverage using a genetic algorithm* [MSJ00], *Genetic algorithms and its application in software test data generation* [WJHZ98], *A strategy for using genetic algorithms to automate branch and fault -based testing* [JES98], *The automatic generation of software test data sets using adaptive search techniques* [JSYE95] *CAST with GAs-automatic test data generation via evolutionary computation* [Rop96], *Generating software test data by evolution* [MMS01], *Automatic Software Test Data Generation Using a Genetic Algorithm*

[MGGZ94], *Instrumenting Programs With Flag Variables For Test Data Search By Genetic Algorithms* [Bot02], *Fitness function design to improve evolutionary structural testing* [BSS02], *Genetic Algorithms and the Automatic Generation of Test Data* [RMB$^+$95], and *Breeding Software Test Cases with Genetic Algorithms* [BFJ$^+$03].

7.3 Integration test design

The integration testing is used to test how components and subsystems work together and the idea is to concentrate on interfaces between components. Usually integration testing is performed with module testing (where each module is tested independently) and integration testing advances by collecting more modules to the test system. Missing modules, which are not implemented yet or cannot be used in test platform, are replaced by using *stub modules* which simulates functionality of the real modules. [HM98]

Hanh *et al.* have proposed a new OO integration test process in the paper *Selecting an efficient OO integration testing strategy: an experimental comparison of actual strategies* [HATJ01].

Integration testing is used to test new components with tested components, which are already part of the system. Integration testing in OO systems could be difficult, because OO systems have strong connectivity between their components. For example, it is very common to pass objects of self-defined classes as parameters rather than primitive types from used language. [HATJ01]

Methods based on classical integration strategies are usually based on a graph representation of the system. Methods will move step-by-step through the graph with testing all the components. The paper points, that those classical integration strategies could be useless when developing a OO system. [HATJ01]

Proposed technique can be used to the allocation of testing resources (human testers) to integration test tasks while keeping number of stubs low. A stub for the software component is dummy component, which simulates the behaviour of the actual software component. If component C_1 uses services of component C_2, then C_1 depends on C_2 and cannot be used without it. During integration tests, C_1 can be tested without C_2 by replacing C_2 with dummy stub. The method for testing resource allocation was similar to the one in the integration strategy called Triskell ([TJJM00]). [HATJ01]

Dependencies between OO classes (components) are represented using Test Dependence Graph (TDG), where nodes are classes and directed edges between them are test dependencies. Test dependencies are classified to three levels: *Class-to-class* (Can be retrieved from the early design model), *method-to-class* (if method has class in its signature) and *method-to-method* (If method uses another method). First two levels can be retrieved directly from unified modeling language (UML) class diagram and rest from the source code. [HATJ01]

Building of a test integration order is done step-by-step, *e.g.* integration of class C_1, which needs class C_2, is not achieved until C_1 is tested with real C_2. C_2 dependencies can be satisfied by implement needed stubs. After that, C_1 can be integrated fully. [HATJ01]

In this method, the GA was used only to minimise the effort to implement stubs. Each individual is a single integration order represented as set of letters, each corresponding one class to be integrated, *e.g.* $\{FEHDB\}$. Stubs are assigned to test order if needed and the fitness value for the individual is number of needed stubs. Stubs are also classified to be simple or complex, depending on the functionality of the real module. In order to minimise the effort to implement stubs, especially the need to implement complex stubs must be minimised. [HATJ01]

Method was compared against three integration strategies based on graph algorithms by generating integration test orders for different OO software, like InterViews, a graphics library and Swing, a Java2 graphical user interface library. It was discussed, that results with GA are promising but still deterministic optimised Triskell was given better performance in most of cases. [HATJ01]

Briand *et al.* in the paper *Using Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders* [BFL02] criticize the use of graph algorithms and proposes a GA based method on integration test order problem. Single individual (test order) is a string of characters, each corresponding to one class. Class dependencies are taken from UML class diagram.

Breaking composition and inheritance relationships between classes is completely avoided, because it could lead to very complex stubs. By using two coupling metrics, a complexity is calculated for the remaining dependencies (simple aggregations and usage associations). The value of A is a number of local attributes in class when references to the class appear as method parameters in the other (caller) class or local parameters of the method. [BFL02]

Inherited methods and attributes are ignored in A. The value of M is a number of local methods and constructors in class, which are used by the methods and constructors of other class. Thus, the A measures maximum number of attributes that must be implemented in stub and B measures number of class methods needed to implement or simulate in stub. In the complexity matrix $Cplx(i, j)$ lines and rows are classes and

i has dependency to j. Both complexity values are normalised by using Equation 14 where $Cplx_{max}$ is MAX value from the matrix. [BFL02]

$$\overline{Cplx(i,j)} = \frac{Cplx(i,j)}{Cplx_{max}} \tag{14}$$

According to Briand *et al.* [BFL02], the overall stub complexity $SCplx(i,j)$ between classes i and j is calculated with A and M by using Equation 15, where W_A and W_M are normalised weights for A and B and $W_A + W_M = 1$.

$$SCplx(i,j) = (W_A * \overline{A}(i,j)^2 + W_M * \overline{M}(i,j)^2)^{1/2} \tag{15}$$

The fitness value, to be minimised, for a test order is calculated by using Equation 16 where o is test order and d is set of dependencies between the classes.

$$OCplx(o) = \sum_{k=1}^{d} SCplx(k) \tag{16}$$

A case study was performed to a system with 21 classes and a large number of cyclic dependencies between them with different coupling measure weights. Results from case study were said to be encouraging and GA was performed as well as graph algorithms. [BFL02]

7.4 Testing based on mutation analysis

One possible way to measure the quality of test cases, is based on *mutation analysis* [DLS78]. The idea is based on assumption, that the quality of single test case is related to the comparative number of detected faulty test program mutants. [BFJLT02]

In the paper *Genes and bacteria for automatic test cases optimization in the .NET environment* Baudry *et al.* [BFJLT02] has used GA to generate test cases. The idea was to generate test cases using GA and measure their fitness by using faulty test program mutants generated by mutation analysis. Used platform was .NET and language was C#.

A single individual, $I = [G_1, \ldots, G_n]$, is set of n syntactic tree nodes ($G_x = [N_1, \ldots, N_n]$) for the program. The use of syntactic tree allows system to generate syntactically correct test cases. For example, during mutation and crossover the GA is allowed to replace the class method only with a constructor, a destructor, a field or a property, which all are on the same level in C# syntactic tree.

The quality of single test case I tested with n mutants is calculated by using fitness function F (Equation 17), where S_i is set of detected mutants, $nbMutants$ is total number of mutants, and the cardinal of the union $\cup_{i=1}^{n} S_i$ is the number of detected mutants.

$$F(I) = \frac{card(\cup_{i=1}^{n} S_i)}{nbMutants} * 100; \qquad (17)$$

A case study using the developed method to generate test cases was performed. The encoding of the GA individual was especially designed for the target program of this case study, a simple C# parser. The parser is implemented in C# and takes C#

source as an input and outputs corresponding syntactic tree. Results from GA based technique were compared against a bacteriological, non-EA approach, which simulates the bacteriological adaptation process. The process is based on bacteriological loop, which had phases for mutation, reproduction, memorisation of best individuals, and fitness evaluation. When GA was used, the quality of test cases increased very slow and did not reach a high fitness. The bacteriological based technique had significantly better performance.

Baudry *et al.* in older paper *Testing-for-Trust: the Genetic Selection Model Applied to Component Qualification* [BHT00] have developed a method, based on test qualification and mutation analysis with GA, for helping developers to build trustable OO components.

7.5 Searching for response time extremes

According to Haikala and Märijärvi [HM98], there is two main groups of real-time systems: hard and soft systems. With soft systems, there is no strict response time requirements and occasional delay in response does not cause severe problems. An example of a soft system could be the automatic teller machine. A hard system could be a controller software used in a car or nuclear power plant where too high response times can cause severe damage.

In the paper *GA in program testing* Alander *et al.* [AMTV96] have represented the idea to use GA to find inputs for a system, which causes maximal response times. This can be used to measure, how the system will satisfy possible response time requirements.

In the paper *Searching protection relay response time extremes using genetic algorithm - software quality by optimization* [AMMM98, Man03] Alander *et al.* have used GA to search response time extremes from embedded electrical network protection relay software. A simulator was used instead of the physical embedded system and some hardware drivers had been modified to make possible the use of simulator without a real electrical network environment.

System takes inputs like electrical network status messages and user commands. GA is used to search inputs, which will lead to long response times in system. A response time is the time that system will use to process inputs and produce output message. Each individual is set of inputs and fitness value is the same as the system response time.

The GA based method was compared against random generator method. A statistical significance for results were calculated, and average response time found using GA was from about 15% to 19% longer than response times with pure random inputs. Also the second test was performed. Objective was to determine a phase during internal message passing (between CPU and I/O devices), which causes most of the delays.

Paper *Automatic software testing by genetic algorithm optimization, a case study* [AM99, Man03] contains a case study related on same research and in the paper *Genetic algorithms in software testing - experiments with temporal target functions* [AM00, Man03] Alander *et al.* have done experiments using temporal fitness function and meta-GA with same embedded environment. A meta-GA means the use of GA to search of good GA parameters for better GA performance.

Grochmann and Wegener has used GA to find shortest and longest execution times of different software systems in *Evolutionary Testing of Temporal Correctness* [GW98] and *Testing Temporal Correctness of Real-Time Systems by Means of Genetic Algorithms* [WGJ97]

At first, a Control Flow Graph (CFG) for a simple computer graphics function written in C was builded. By using systematic approach, 49 test cases was created to cover all the branches of the program. Execution times for different executions was measured in processor cycles and they varied from 359 to 1839 cycles. A GA and random search tests were performed to found inputs for maximum and minimum execution times. After 800 test runs, GA has reached maximum of 1839 cycles and also found a new minimum of 355 cycles. A random search used 4600 runs to found same extremes as systematic approach. More comparative tests were performed with different programs, LOC size varying from 107 to 1511 and GA was always founding better extreme times in lesser time than random search. [WGJ97, GW98]

O'Sullivan *et al.* in the paper *Testing Temporal Correctness of Real-Time Systems - a New Approach using Genetic Algorithms and Cluster Analysis* [OVW98] introduces the use of cluster analysis in GA based software testing. *Cluster analysis for genetic algorithm output* is a method developed by Vössner *et al.* [VB96], which can be used to get information about population and determine if GA run should be terminated. O'Sullivan *et al.* summarises the algorithm using following pseudo code: given any two individuals i and j, if the distance between them $d_{ij} < d_{check}$, then they will be members of the same cluster. There are following three cases [OVW98]:

1. If they do not belong to a cluster yet, they form a new cluster.
2. If only one of the two individuals already belongs to a cluster, the other one joins this cluster.
3. If both individuals are members of different clusters, these clusters are combined thus forming a single cluster.

An early termination of the GA with cluster analysis was said to be important because change of finding better solutions is significantly reduced because of converged population and there is also possibility to gain information about local optimum. In the real-time system testing, the information about local optimum would be useful to optimise program code and detect several performance leaks with only one test run.

The experiment was performed using the system from automotive electronics with 70 input parameters and 1511 LOC. Objective was to find inputs using GA, which cause worst-case execution times (processor cycles). Termination criteria was based on cluster analysis.

Seven independent runs were performed and longest execution times varied from 9459 to 12178 processor cycles. The convergence of individuals was analysed by using cluster analysis for the final population of each run.

In comparison against other popular termination criteria, cluster analysis was said to be most promising method. Limiting GA run by using generation number or used time is not suitable because it is not always possible to know that the testing has converged before limit is reached. Using execution time (fitness) as stopping criteria has also same kind of problem.

Wegener *et al.* in the paper *Testing the Temporal Behavior of Real-Time Tasks using Extended Evolutionary Algorithms* [WPS99] has applied *extended evolution testing* to testing response times of motor control system. Results from GA based evolutionary testing (ET) was compared to tests performed by developers (developers' tests, DT). Although developers are familiar with internal structure of their software, ET has found longer executable times than DT. One explanation for this was said to be maybe the use of system calls, compiler optimisations and dependencies to other parts of the system.

Other related papers are *Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints* [MW98], *Evolutionary Algorithms for the Verification of Execution Time Bounds for Real-Time Software* [GJE99], *Verifying Timing Constraints of Real-Time Systems by means of Evolutionary Testing* [WG98], *Testing the results of static worst-case execution time analysis* [PN98], *Measuring Evolutionary Testability of Real-Time Software* [Gro00], and *Evolutionary Testing of Embedded Systems* [SBW01].

7.6 Miscellaneous

There are at least two patents about using evolutionary algorithms to software testing: Whitten with *Method and Computer Program Product for Generating a Computer Program Product Test that Includes an Optimized Set of Computer Program Product Test Cases, and Method for Selecting Same* [Whi95], and Gounares *et al.* *Adaptive Problem Solving Method and Apparatus Utilizing Evolutionary Computation Techniques* [GS01].

Tracey *et al.* have developed an automated test-data generation framework, which can be used to test safety-critical systems. Framework is represented in the paper *A Search Based Automated Test-Data Generation Framework for Safety-Critical Systems* [Tra00].

Pereira and Vergilio in the paper *GPTesT: A Testing Tool Based on Genetic Programming* [EV02] represent a testing tool based on GP.

Mantere and Alander has used GA to test a image processing software in the papers *Automatic Image Generation by Genetic Algorithms for Testing Halftonin Methods* [MA00, Man03], *Testing a Structural Light Vision Software by Genetic Algorithms*

- *Estimating the Worst Case Behavior of Volume Measurement* [MA01, Man03], *Developing and Testing Structural Light Vision Software by Co-Evolutionary Genetic Algorithm* [MA02, Man03], and *Testing Digital Halftoning Software by Generating Test Images and Filters Co-Evolutionarily* [MA03, Man03]

8 Summary

Methods based on evolutionary algorithms could be applied to different software engineering problems and there are many related applications which can be allocated to different phases of the waterfall model. This Chapter summarises findings of this work.

Table 2: The number of classified publications

Class	Subclass (Application area)	The number of publications
Analysis	Prediction of software failures	8
	Software project effort prediction	7
	Project management	3
	Exploring difficulty of the problem	1
		19
Design	Multiprocessor scheduling	17
	Task and resource allocation in distributed systems	9
	Hardware/software co-design in embedded systems	7
	Architecture design	5
	Protocol construction	2
		40
Implementation	Re-engineering	4
	N-version programming	3
	Automatic programming	2
	Search for compiler optimisations	1
		10
Testing	Structural (white box) testing	18
	Searching for response time extremes	13
	Functional (black box) testing	11
	Miscellaneous	8
	Integration test design	2
	Testing based on mutation analysis	2
		54
All	**Total**	**123**

Within this work, there were 123 publications from years 1992 to 2003 in literature where EA was used to solve a particular software engineering problem. Table 2 and Figure 26 illustrates how publications are distributed to different phases (classes) of the waterfall model.

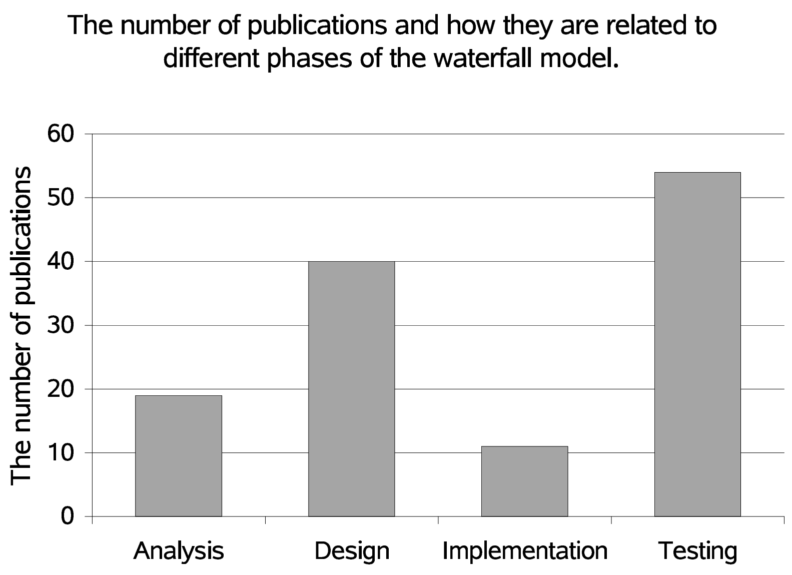

Figure 26: EA usage in different phases of the waterfall model

Most of the applications were related to software design, testing and implementation. The implementation phase was found to be problematic, because genetic programming technique itself is fundamentally targeted only to generate programs and almost every use of it can be classified into implementation phase. Because it is not possible to report every case from literature where GP is used, only cases which are closely related to well known software engineering problems are discussed. After setting this outlining, majority of the publications are related to design or testing phases. About half of the cases (59) were related on pre-implementation phases (system/information engineering area) of the waterfall model. On next four Sections, applications related on different waterfall model phases are summarised.

8.1 Analysis

In analysis phase, majority of applications were about using GA or GP to build predictive models from a software engineering data. Historical data about software quality, module failures, software development project effort or software project cost was splitted in to training and validation sets and the model was trained to predict the outcome of the development work. In two cases, the GA was used only to train neural network to do similar task.

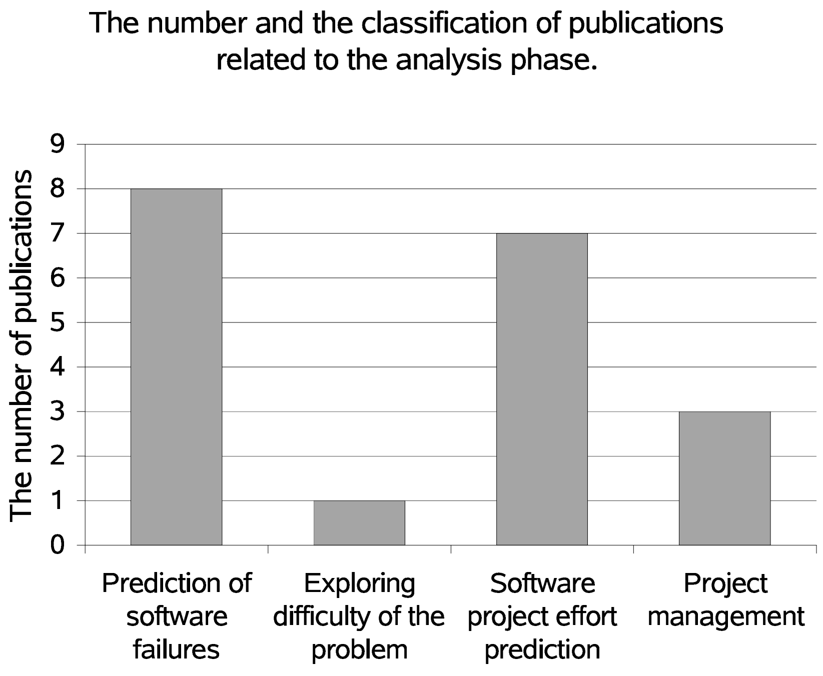

Figure 27: EA usage in the analysis phase

It looks like the practical problems with this approach have more to do with the data itself than used computing methods. According to Bouktif *et al.* [BKS02], the common problem with historical software engineering data is that companies that systematically collect data, does not publish it or publish only summaries. Problems in this application area could be about similar when using using non-EA techniques, like ANN to do a similar task.

The GP was also used to collect more knowledge about the problem to be solved with the software [Fel99] . This was done by exploring the input space of the problem and searching for areas, where programs generated with GP will be more faulty. However, it is still possible that results will not tell how difficult the problem is to be completely solved by human programmers and that was also discussed in the paper.

There were three papers on resource allocation problem related to software projects. The idea was to assign resources (employees) to project tasks and to minimise the total completion time of the project.

8.2 Design

Most of the applications in the design phase are related to static task allocation in multiprocessor systems in order to maximise the processor utilisation and minimise required time to run the program. In addition, there was applications related to task and resource allocation problem in distributed systems, where the domain is a multicomputer system, where also resources, like databases and files are assigned to different nodes. Applications related on hardware/software co-design in embedded systems are also static multiprocessor scheduling problems. Basically, these all applications are very similar to the project scheduling problem, described in the analysis phase. EA could be used for task/resource allocation (scheduling) in multiprocessor and distributed systems when a static scheduling is used. The static scheduling means, that execution/utilisation order is known before the runtime. This fact limits the possible usage.

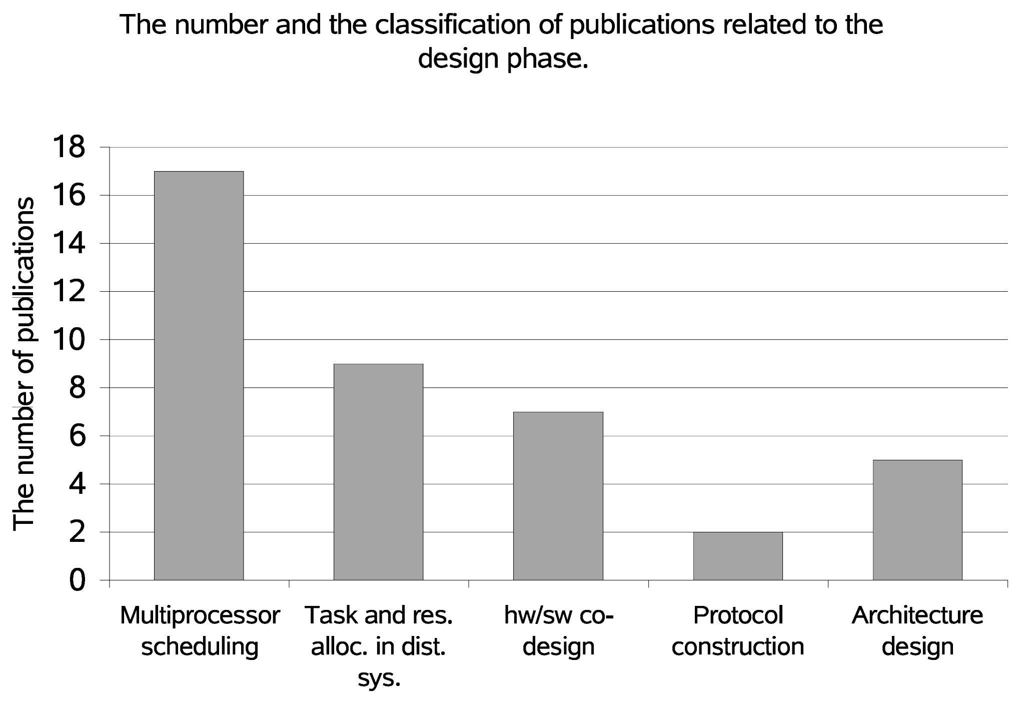

Figure 28: EA usage in the design phase

Architecture design with EA was only a system partitioning: modules have defined connections (dependencies) between them and they are assigned to clusters (subsystems) which must have minimal number of dependencies between them and also a reasonable number of dependencies between internal modules.

8.3 Implementation

It is not appropriate, within the scope of this work to report every use of GP from the literature as an EA based application in the implementation phase of the waterfall model. That is why this class contains only cases which are closely related to well known software engineering problems. In implementation phase, no single application area was clearly more popular than the others. There were few applications related to automatic programming, N-version programming, re-engineering, and searching for compiler optimisations.

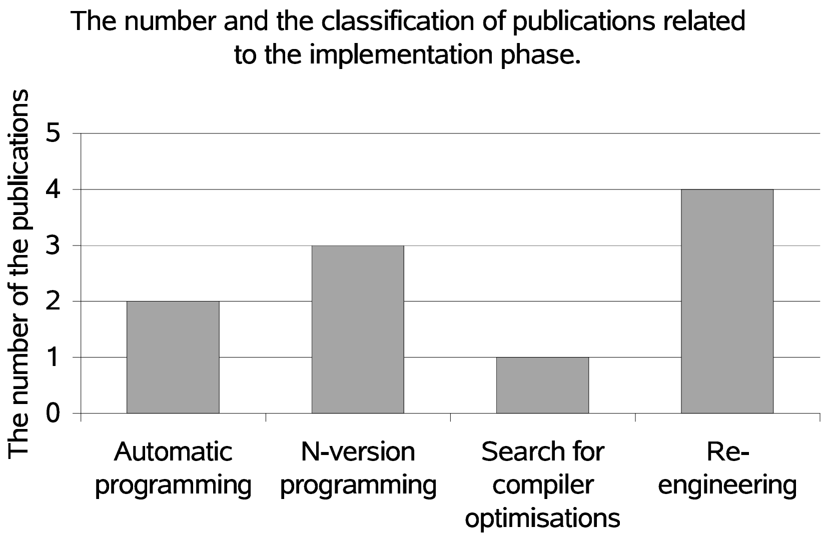

Figure 29: EA usage in the implementation phase

Most of the applications were about software re-engineering and there was no similarities between the approaches. The GP was applied to different re-engineering tasks, mainly to program auto-parallelisation, which involves re-writing programs to execute on a multiprocessor system. In addition, the GA was used to restructure the OO design and clusterise components to the reusable library.

8.4 Testing

Nearly half of the all reported EA applications in this thesis were about software testing. Most of the testing publications were concentrated to the structural (white box) testing, where objective was to find minimum length of test data with maximum coverage for the given program. There are different paths through the program and the coverage metric measures how compeletely given test data input will use programs functionality. Usually this requires that there is a graph representation of the tested program (or some technique that can explore through the source code and generate a graph about the program which can be used with EA based test program). However, applications on this area are more about test data generation rather than just program testing.

There was no really large programs under structural testing, only little test programs and EA was usually able to generate test data which covers the whole program. The EA based test generation was usually compared against other techniques and also with exhaustive search and it outperformed all other techniques.

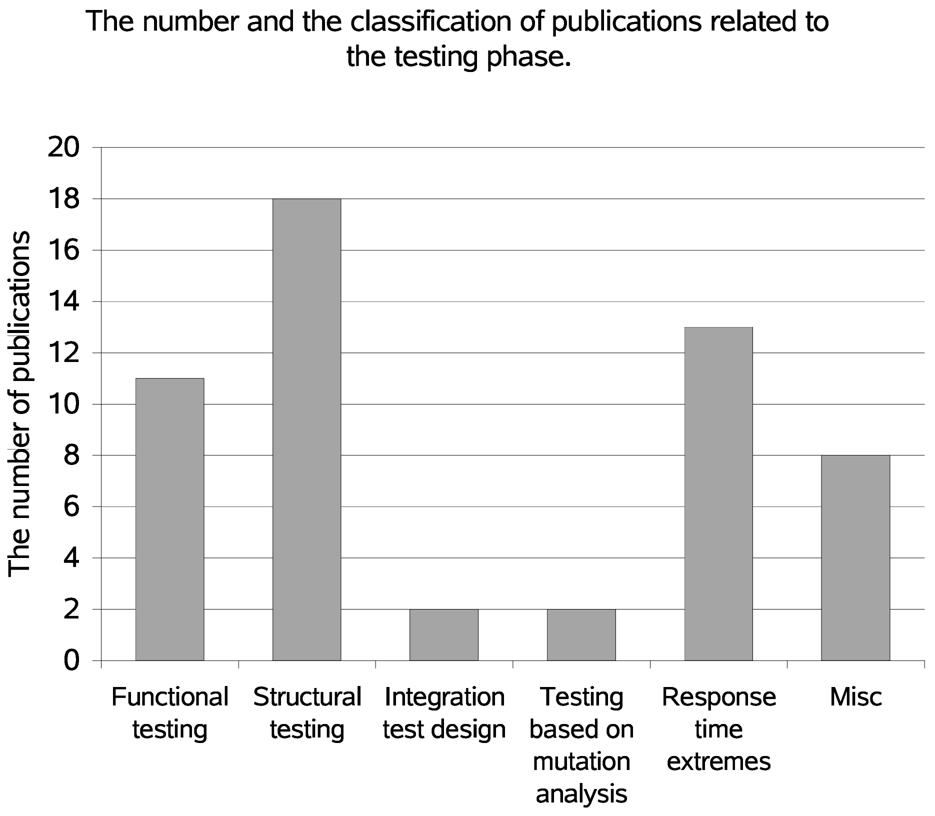

Figure 30: EA usage in the testing phase

In functional, black box testing, test cases are generated by using software requirements without any knowledge about structure of the program. Usually there was objective to find input data which causes failures, *e.g.* autonomous-vehicle controller software was tested using GA generated inputs and by evaluating outputs using software requirements. In this case, the goal is to find several inputs causing failures rather than to find the most severe failure. Developers can use failure scenarios (given input values and information about the failure) when fixing bugs in the program. Failures were usually classified or clusterised when generating failure scenarios.

Software response times are concerned especially when a software is a real-time system. Many embedded systems like different kinds of controllers are also real-time systems, and they can have strict response time requirements. For example, if the vehicle control software reacts to the particular change in the input values too late, it can cause danger.

Testing, which has the objective to find maximum response times from the software is very close to black box and functional testing. This is also the second most popular application area in the testing class after the structural testing. Like in the black box testing, also in this area, failure scenarios which can describe (and maybe generalise) inputs with high response times can be useful to developers who are fixing the software.

The EA based search for finding response time extremes were usually compared against other methods like random search and with test programs, it was able to find higher execution times in reasonably shorter time. There were some cases, where EA based method was used to test a real-world controller software. It looks like this application area is one of the few, where EA search was used to a real software engineering problem and had real advantages.

There were also two applications about the design of integration test orders. The objective was to search order to add components and subsystems to the system under integration test. Because of dependencies between the components, there might be a reason to implement some “dummy” stub modules, which imitates the behaviour of the real module and satisfy a particular dependency during an integration test. In a good integration test order, effort caused by implementation of stub modules is minimal. When testing a large system, this method could be useful, especially if it seems that the needed stubs could be complex to implement.

Some EA applications related to the testing were classified as “miscellaneous”. There is a GP based testing tool called GPTesT, a test-data generation framework for safety-critical systems. Also testing of the image processing software was classified in this class and was introduced only in brief because it is not very close to the common software engineering problems.

8.5 Summarising remarks

Methods based on evolutionary algorithms (EA) could be applied to different software engineering problems and there are many related applications which can be allocated to different parts of the waterfall model. Almost all applications, discussed within this work, were tested only in an experimental environment rather than real situation where a real software is under development. 123 publications about using EA on software engineering problems were found from the literature and majority of them are related to software design (40) and testing (54) phases. There were also some application classified to the analysis phase (19) and the software implementation (limited to 10).

In general, there were some NP-complete problem (which requires time which is exponential in the problem size [Fou04]) to solve, related to software engineering. The problem was transferred to the fitness function and data structure, which can be used to form a set of solution candidates. Then GP or GA based solution was compared against results of other methods like random search or a problem specific heuristic. If the search space for a particular problem was relatively small allowing to find the best solution using an exhaustive search, then EA was usually able to find the same best solution but in shorter time. Reason for this could be that EA methods are favouring good solution candidates when creating new generation and usually good solutions in the search space are closely-spaced. With a larger search space, EA was usually compared against the random search or a problem specific heuristic and almost every time, EA was able to find better solutions in shorter time. The reason why random search was outperformed by EA is in the ability of EA to have *memory* (mutation and crossover) between generations of solution candidates, *i.e.* it is not blind like the random search. There were different kinds of problem specific heuristics, and reason why EA methods were usually better could be in their stochastic nature, which prevents the search to get stuck in the local optimum. However, EA methods have higher computational requirements than those problem specific heuristics.

There were no applications using Evolution Strategy (ES) or Evolutionary Programming (EP). According to Ryan in the book *Automatic Re-Engineering of Software Using Genetic Programming* [Rya00], a renewed interest in EP has been growing recently.

In analysis phase, most popular applications were about using GA or GP to build models from the historic data which will predict software failures, software project efforts or project cost. Also task scheduling for software projects has been done.

The design phase was mainly about task scheduling in multiprocessor or multicomputer systems with GA. Scheduling was static, which means that tasks are assigned to processors or computers before they are executed. In distributed systems, also resources were assigned by depending on their use. There were also applications related to architecture design, where by using the GA, the set of software modules was partitioned to optimal subsystem.

For implementation phase there were only few interesting cases about automatic programming, N-version programming, compiler optimisations, and software re-engineering. The re-engineering applications were related to using GP to program auto-parallelisation and using GA to clusterise implemented software components.

In testing phase, most of the applications were about using GA in the functional testing, the structural testing, and searching for response time extremes. The structural testing has the objective first to generate test data (test case), which covers all possible execution paths through the program. After that, results data can be used to test the program. EA based method for generating test data for structural testing could be useful, when there is really need to make automatic test case generation, because the number of different paths through the program can be very high.

The functional testing and searching for response time extremes both mean generating test inputs for the program and search for faulty or delayed output. Both techniques are very useful and can be used even with real-world testing problems, especially when testing real-time systems, where the behaviour of the system under test is difficult to predict. Because fixing the bugs is one of the most important tasks after software tests, there should be need for more research on the methods, how to generate minimal number of interesting failure scenarios which will cover maximum number of failures in the software. This can help developers to find bugs from the program code. There was also testing related to image processing software, mutation analysis testing and integration test orders. Also a testing tool based on GP and a complete framework for testing safety-critical systems were mentioned.

Many cases were used only in an experimental environment rather than real situation where a real software is under development. Usually it looks like that problems to be solved with EA were searched rather than to search methods for the particular problem. It could be said, that all mentioned EA based methods can be useful in different software engineering tasks, but in practise, there is a tradeoff related to benefits and efforts. For example, usually it is significantly easier to outline the system architecture or the project schedule by using experience and fix it some time later than to transfer the problem to the suitable form and use EA to search for optimal solution. There were also some more or less complete frameworks and EA based computer aided software engineering (CASE) tools, which could be more usable in software production rather than just methods.

9 Conclusions

Within this work, different kinds of applications of the evolutionary algorithms (EA) to solve software engineering problems were found in the literature. Applications from 123 publications were reported and classified into four software development phases, which are derived from the waterfall model. The mentioned four phases are analysis, design, implementation, and testing. Almost all the applications discussed within this work, were using EA based methods called the genetic programming (GP) or the genetic algorithm (GA). It is possible, that this work covers about half, more likely more half than less, of all the EA applications in the area of the software engineering.

EA techniques were used to build predictive models from software engineering data. Different kinds of historic data were splitted in to training and validation sets and models were trained to predict software quality, module failures, project effort, and software costs [BL98, EKCA98, EKA99, Dol00, Shu00, ARRRT01, BL01, Dol01, LK01, BKS02, RPA02, SMLE02]. EA was also used to train neural network to predict software failures [HKAH96, HKAH97]. EA techniques in this area seem to be promising alternative to other approaches, *e.g.* statistical analysis or neural networks. To systematically collect software engineering data is a long-span task and according to Bouktif *et al.* [BKS02], there are very few companies that collect and publish related data. There is a need for more research on this area, but the research should be more general and not concentrate only on the use of EA techniques.

GA was used to find optimal schedules for software development projects, where employees must be assigned to tasks, which have dependencies between each other and different kinds of skill requirements [CCNC98, KH00, CCZ01]. This is a promising application area for EA techniques, and can also be used with non-software development projects. The EA project scheduling can become a commonly used technique if implemented in to project management software tools.

The static task scheduling in multiprocessor systems was one area, to which GA was also applied [DAA95, KA97, TOK98, CFR99, Ge99b, Ge99a, JPP00, Fis00, OBWK00, BC00, YQN01, Las01, LC03]. Tasks of a particular parallel program were assigned to multiple processors in order to minimise the execution time of the program. With GA, tasks and resources (databases and files) were also assigned to nodes of the distributed system [KH93, AK94, BA94, KPG95, WYKH97, SM98, Ahu00, SNYF00, Ram01, OCW00]. In addition, there were also applications related to hardware/software co-design in embedded systems, which were mainly static multiprocessor scheduling [SC94, Agu96, Sem98, Gra99, CRC02, YG02a, YG02b]. Both techniques can be useful if there is really use for the static scheduling, where execution times for tasks are known before the program starts.

In the architecture design, EA techniques were used to clusterise modules to subsystems: modules have dependencies between them and they are assigned to subsystems which must have minimal number of dependencies between them and also a reasonable number of dependencies between internal modules [MMR$^+$98, DMM99, HHP02, MA02]. This kind of problems are only a small part of the architecture design, but the EA based module clusterisation could be useful if implemented to the computer aided software engineering tool.

GP is fundamentally targeted to generate computer programs, and therefore it can have a large number of applications in the software engineering. Because of this, only applications related to well-known software engineering problems are discussed. For example, GP was applied to N-version programming [Fel98b] and the software auto-parallelisation [Rya00]. The automatic programming area is an experimental and there is very little use for it in the practical software production.

The automatic programming, mainly based on the GP, is still an interesting research area and there should be more research on it. The applied research for GP should be done in order to find new practical applications for it. In addition, the basic research for developing GP variants and other approaches to automatic programming should be done.

Most of the applications related to software testing were about the functional testing [XES+92, SGDJ93, Hun95, MT95, Sth95, Wat95, BM96, KG96, YZC99, WBS02, BFJ+03, PWW93], the structural testing [MGGZ94, JSYE95, MT95, Rop96, SF96, Xil98, WJHZ98, PHP99, BJ00, MSJ00, MMS01, BSS02, Bot02, BFJ+03], and searching for response time extremes [VB96, WGJ97, AMMM98, GW98, GJE99, MW98, OVW98, PN98, WG98, AM99, WPS99, AM00, Gro00, SBW01, Man03]. The EA based structural (white box) testing has the objective first to generate test inputs, which covers all possible execution paths through the program. When testing the program using an optimal input test data, all the failures caused by the faulty program code should occur. This could be interesting, because according to Pressman [Pre01], structural testing could in theory lead to 100% correct programs. This approach can be useful in the practical software production, if applied to simple pieces of the program, like individual functions. The problem with EA structural testing could be the fact, that there must be a graph presentation of the internal structure of the program.

EA is used in functional (black box) testing to find test inputs, which will cause failures in the software. The objective is to find several inputs causing failures rather than to find the most severe one. Developers can use failure scenarios (given input values and information about the occured failure) when fixing bugs in the program. Failures are usually classified or clusterised after a set of failure scenarios has been found. Because fixing the bugs is one of the most important tasks after a software failure occurs, there should be a need for more applied research on the methods, how to generate minimal number of interesting failure scenarios which will cover maximum number of failures in the software. The application area of searching for response times extremes was similar to functional testing, but aims to find test inputs which cause too long response times in the software.

At the moment, EA applications related to the functional testing and searching for response times extremes are both possible to use in practical software production, especially when testing a real-time system. The reason why the black box approach is suitable for testing real-time (embedded) systems, is in their different nature. According to Haikala and Märijärvi [HM98], real-time systems usually executes in a loop and continuously read values from the input stream(s) and adjust output value. Inputs are rarely synchronised between or with the system itself. Because of those circumstances, it can be difficult to predict behaviour of the real-time system. Embedded systems are also used very widely and according Thaller [Tha97] almost 90% of all electronic components produced nowadays (1997) are used in embedded systems. Embedded systems are also used with safety-critical systems like nuclear power plants and severe software failures can cause danger to human lives. In addition, there should be more EA based testing tools in order to increase usage of such testing techniques in the real-world software production.

GA was also used to generate optimal integration test orders, where the objective was to search the order to add components to the system [TJJM00, HATJ01, BFL02]. Because of the component dependencies, there might be a reason to implement dummy stub modules, which imitate the behaviour of the real module and satisfy a particular dependency during the integration test. In a good integration test order, effort caused by implementation of stub modules is minimal and when testing a large system, this method could be useful, especially if it seems that the needed stubs could be complex to implement. This technique seems to be ready to be used in practical software production if the system under test is large. In addition, there should be software design tools where such a method is implemented.

In general, there was some NP-complete problem related to software engineering. The problem was transferred to the fitness function and data structure, which can be used to form a set of solution candidates. Then the solution was compared against the results from other (non-EA) methods. If the search space for a particular problem was relatively small allowing to find the best solution using an exhaustive search, then EA was usually able to find the same best solution but in shorter time. Reason for this could be that EA methods are favouring good solution candidates when creating new generation and usually good solutions in the search space are closely-spaced. With a larger search space, EA was usually compared against the random search or a problem specific heuristic and almost every time, EA was able to find better solutions in shorter time. The reason why random search was outperformed by EA is in the ability of EA to have *memory* (mutation and crossover) between generations of solution candidates, *i.e.* it is not blind like the random search. There were different kinds of problem specific heuristics, and reason why EA methods were usually better could be in their stochastic nature, which prevents the search to be stucked in the local optimum. However, EA methods have higher computational requirements than those problem specific heuristics.

Almost all EA applications, discussed within this work, were tested only in an experimental environment rather than in a real situation where a real software is under development. The reason, why such techniques are not used widely to solve software engineering problems in practise, is in the lack of EA based computer aided software engineering (CASE) tools. Currently, EA techniques are not adopted to wide practical use and interest in them is mostly limited to (academic) research. However, according to Alander [Ala98], the number of research publications related to EA in general has been rapidly increasing over the past few years. It could thus be assumed that more practical EA applications in the area of the software engineering will be available in next few years.

Most promising application areas for EA based techniques are the functional testing, structural testing, integration test design, and searching for response time extremes. The EA based task scheduling in multiprosessor and distributed systems is also promising, if there is a need for the static scheduling. EA based project scheduling is also useful in large projects which cannot be scheduled by using common sense. In addition, the software quality, failure, and project effort prediction based on EA are also useful, but in this area, EA is only an alternative for other techniques, like neural networks or the statistical analysis.

The EA applications for project scheduling, the architecture design (clusterising modules to optimal subsystems), and the design of integration testing are ready for use in practical software production, if there will be CASE tools where the techniques are implemented. The software quality and failure prediction and project effort prediction are already used with a large number of different techniques and EA is promising and ready-to-use alternative for them. It is possible, that the EA based functional testing and searching for response time extremes will be used in practical software production, especially when developing mission critical real-time systems.

The most promising research related to reported EA applications will be in the area of software testing, especially in both black box approaches: the functional testing and searching for response time extremes. The prediction of the software quality, failures, cost and project efforts should be subject of research, but it should not to be concentrated only on EA based techniques, because the problems in the area are more related to the data than computing methods. In addition, more research should be done on the Genetic Programming (GP) technique itself.

References

[AC77] Avizienis, A. and Chen, L., On the implementation of n-version programming for software fault tolerance during execution. *Proceedings of IEEE COMPSAC 77*, Nov 1977, pages 149–155.

[Agu96] Aguilar, J., Heuristics to optimize the speed-up of parallel programs. *Parallel Computation, Third International ACPC Conference with Special Emphasis on Parallel Databases and Parallel I/O Proceedings*, Sep 1996, pages 184–183.

[Ahu00] Ahuja, S., A genetic algorithm perspective to distributed systems design. *Proceedings IEEE SoutheastCon 2000*, 2000, pages 83–90.

[AK94] Ahuja, S. and Kumar, A., A genetic algorithm approach for performance based reliability enhancement of distributed systems. *Seventh International Conference on Parallel and Distributed Computing Systems*, Oct 1994, pages 664–669.

[AL97] Alander, J. T. and Lampinen, J., Cam shape optimization by genetic algorithm. *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, Poloni, C. and Quagliarella, D., editors, Trieste, Italy, Nov 1997, John Wiley et Sons, New York, USA, pages 153–174.

[Ala98] Alander, J. T., *Geneettisten algoritmien mahdollisuudet*. Teknologiakatsaus 59/98. TEKES - National Technology Agency of Finland, 1998. ISBN 951-53-1392-9.

[AM99] Alander, J. T. and Mantere, T., Automatic software testing by genetic algorithm optimization, a case study. *SCASE'99 - Soft Computing Applied to Software Engineering*, Ryan, C. and Buckley, J., editors, Limerick, Ireland, Apr 1999, University of Limerick, pages 1–9.

[AM00] Alander, J. T. and Mantere, T., Genetic algorithms in software testing - experiments with temporal target functions. *MENDEL 2000 6th International Conference on Soft Computing*, Osmera, P., editor, Brno, Czech Republic, Jun 2000, Brno University of Technology and PC-DIR, pages 9–14.

[AMMM98] Alander, J. T., Mantere, T., Moghadampour, G. and Matila, J., Searching protection relay response time extremes using genetic algorithm - software quality by optimization. *Electric Power Systems Research*, 3,46(1998), pages 229–233.

[AMTV96] Alander, J., Mantere, T., Turunen, P. and Virolainen, J., Ga in program testing. *Proceedings of the Second Nordic Workshop on Genetic Algorithms and Their Applications 2NWGA*, Alander, J., editor, Vaasa, Finland, Aug 1996, University of Vaasa, pages 205–209.

[ARRRT01] Aguilar-Ruiz, J., Ramos, I., Riguilme, J. and Toro, M., An evolutionary approach to estimating software development projects. *Software Technology*, 43,14(2001), pages 875–882.

[BA94] Barada, H. and Adar, N., Efficient allocation of program modules on multicomputers. *Seventh International Conference on Parallel and Distributed Computing Systems*, Oct 1994, pages 556–560.

[BC00] Bae, S. H. and Choi, S. B., An implementation of the linear scheduling algorithm in multiprocessor systems using genetic algorithms. *Journal of KISS: Computer Systems and Theory*, 27,2(2000).

[BFJ$^+$03] Berndt, D., Fisher, J., Johnson, L., Pinglikar, J. and Watkins, A., Breeding software test cases with genetic algorithms. *36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, Big

Island, Hawaii, USA, Jan 2003, Institute of Electrical and Electronics Engineers.

[BFJLT02] Baudry, B., Fleurey, F., Jezequel, J. and Le-Traon, Y., Genes and bacteria for automatic test cases optimization in the .net environment. *Proceedings - 13th International Symposium on Software Reliability Engineering*, Annapolis, USA, Nov 2002, IEEE Comput. Soc, Los Alamitos, CA, USA, pages 195–206.

[BFL02] Briand, L. C., Feng, J. and Labiche, Y., Using genetic algorithms and coupling measures to devise optimal integration test orders. *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, Ischia, Italy, 2002, ACM Press, pages 43–50

[BHT00] Baudry, B., Hanh, V. L. and Traon, Y. L., Testing-for-trust: the genetic selection model applied to component qualification. *Proceedings 33rd International Conference on Technology of Object Oriented Languages and Systems. TOOLS 33*, Mont-Saint-Michel, France, Jun 2000, IEEE Comput. Soc, Los Alamitos, CA, USA.

[BJ00] Bueno, P. and Jino, M., Identification of potentially infeasible program paths by monitoring the search for test data. *Proceedings ASE-2000. Fifteenth IEEE International Conference on Automated Software Engineering*, Grenoble, France, Sep 2000, pages 209–218.

[BJ02] Bueno, P. and Jino, M., Automatic test data generation for program paths using genetic algorithms. *International Journal of Software Engineering and Knowledge Engineering*, 12,6(2002), pages 691–709.

[BKS02] Bouktif, S., Kegl, B. and Sahraoui, H., Combining software quality predictive models: an evolutionary approach. *International Conference on Software Maintenance*, 2002, pages 1063–6773.

[BL98] Baisch, E. and Liedtke, T., Automated knowledge acquisition and application for software development projects. *Automated Software Engineering, 1998. Proceedings. 13th IEEE International Conference on*, Honolulu, USA, October 1998, pages 306–309.

[BL01] Burgess, C. J. and Lefley, M., Can genetic programming improve software effort estimation? a comparative evaluation. *Information and Software Technology*, 43,14(2001), pages 863–873.

[BM96] Boden, E. and Martino, G., Testing software using order-based genetic algorithms. *Proceedings GB-96 Conference*, J.R. Joza, D.E. Goldberg, D. F. and Riolo, R., editors, Stanford, California, USA, Jul 1996, MIT Press, pages 461–466.

[Bot02] Bottaci, L., Instrumenting programs with flag variables for test data search by genetic algorithms. *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E. and Jonoska, N., editors, New York, USA, July 2002, Morgan Kaufmann Publishers, San Francisco, USA, pages 1337–1342.

[Bru96] Bruce, W., Automatic generation of object-oriented programs using genetic programming. *Genetic Programming, Proceedings of the First Annual Conference*, Jul 1996, pages 267–272.

[BSS02] Baresel, A., Sthamer, H. and Schmidt, M., Fitness function design to improve evolutionary structural testing. *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E. and Jonoska, N., editors, New York, July 2002, Morgan Kaufmann Publishers, San Francisco, USA, pages 1329–1336.

[CCNC98] Chang, C., Chao, C., Nguyen, T. and Christensen, M., Software project management net: a new methodology on software management. *Computer Software and Applications Conference, 1998. COMPSAC '98. Proceedings*, August 1998, pages 534–539.

[CCZ01] Chang, C. K., Christensen, M. J. and Zhang, T., Genetic algorithms for project management. *Annals of Software Engineering*, 11,1(2001), pages 107–139.

[CFR99] Correa, R., Ferreira, A. and Rebreyend, P., Scheduling multiprocessor tasks with genetic algorithms. *IEEE Transactions on Parallel and Distributed Systems*, pages 825–837.

[CJ01] Clark, J. and Jacob, J. L., Protocols are programs too: the meta-heuristic search for security protocols. *Information and Software Technology*, 43,14(2001), pages 891–904.

[CRC02] Chakraverty, S., Ravikumar, C. and Choudhuri, D., A hierarchical genetic algorithm for hardware software co-synthesis with a stochastic approach. *IETE Journal of Research*, 48,5(2002), pages 349–360.

[CSS99] Cooper, K., Schielke, P. and Subramanian, D., Optimizing for reduced code space using genetic algorithms. 34,7(1999).

[DAA95] Dhodhi, M. K., Ahmad, I. and Ahmad, I., A multiprocessor scheduling scheme using problem-space genetic algorithms. *IEEE International Conference on Evolutionary Computing*, 1995, pages 214–219.

[DLS78] DeMillo, R., Lipton, R. and Sayward, F., hints on test data selection : help for the practicing programmer. *IEEE Computer*, 11,4(1978).

[DMM99] Doval, D., Mancoridis, S. and Mitchell, B., Automatic clustering of software systems using a genetic algorithm. *Software Technology and Engineering Practice, 1999. STEP '99*, 1999, pages 73–81.

[Dol99] Dolado, J. J., Limits to the methods in software cost estimation. *Proceedings of the 1st International Workshop on Soft Computing Applied to Software Engineering*, Ryan, C. and Buckley, J., editors, University of Limerick, Ireland, 12-14 April 1999, Limerick University Press, pages 63–68, `http://www.sc.ehu.es/jiwdocoj/docs/dolado-scase99.ps` Cited 12.3.2004

[Dol00] Dolado, J., A validation of the component-based method for software size estimation. *IEEE Trans. Softw. Eng.*, 26,10(2000), pages 1006–1021.

[Dol01] Dolado, J. J., On the problem of the software cost function. *Information and Software Technology*, 43,1(2001), pages 61–72.

[EKA99] Evett, P., Khoshgoftaar, T. and Allen, E., Using genetic programming to determine software quality. *Proceedings of the Twelfth International Florida Artificial Intelligence Research Society Conference*, Kumar, A. N. and Russell, I., editors, Orlando, USA, May 1999, AAAI Press, pages 113–117.

[EKCA98] Evett, M., Khoshgoftar, T., Chien, P. and Allen, E., GP-based software quality prediction. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H. and Riolo, R., editors, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998, Morgan Kaufmann, pages 60–65.

[EV02] Emer, M. C. F. P. and Vergilio, S. R., Gptest: A testing tool based on genetic programming. *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E. and Jonoska, N., editors, New York, July 2002, Morgan Kaufmann Publishers, San Francisco, USA, pages 1343–1347.

[Fel98a] Feldt, R., Generating diverse software versions with genetic programming: an experimental study. *IEE Proceedings - Software Engineering*, 145,6(1998), pages 228–236. `http://www.iee.org.uk/publish/journals/profjrnl/cntnsen.html` Cited 12.3.2004

[Fel98b] Feldt, R., Generating multiple diverse software versions with genetic programming. *Proceedings of the 24th EUROMICRO Conference, Workshop on Dependable Computing Systems*, Vaesteraas, Sweden, 25-27th August 1998, pages 387–396, `http://www.amp.york.ac.uk/external/sweden/sweden.htm` Cited 12.3.2004

[Fel99] Feldt, R., Genetic programming as an explorative tool in early software development phases. *Proceedings of the 1st International Workshop on Soft Computing Applied to Software Engineering*, Ryan, C. and Buckley, J., editors, University of Limerick, Ireland, 12-14 April 1999, Limerick University Press, pages 11–20.

[Fel02] Feldt, R., *Biomimetic Software Engineering Techniques for Dependability*. Ph.D. thesis, Chalmers University of Technology, 2002.

[Fis00] Fissgus, U., Scheduling using genetic algorithms. *International Conference on Distributed Computing Systems*, 2000, pages 662–669.

[Fou04] Foundation, W., Wikipedia, the free encyclopedia, 2004. `http://en.wikipedia.org/wiki/Main_Page` Cited 12.3.2004

[FOW66] Fogel, L., Owens, J. and Walsh, M., *Artificial Intelligence throught Simulated Evolution*. Wiley and Sons, 1966.

[Ge99a] Ge, Q. W., Paradeg-processor scheduling for acyclic switch-less program nets. *Journal of the Franklin Institute*, 7,336(1999), pages 1135–1153.

[Ge99b] Ge, Q., A two-processor scheduling method for a class of program nets with unity node firing time. *IEICE Transactions on Fundamentals of Electronics,-Communications and Computer Sciences*, 11,E82-A(1999), pages 2579–2583.

[GJE99] Groß, H., Jones, B. and Eyres, D., Evolutionary algorithms for the verification of execution time bounds for real-time software. *IEE Workshop on Applicable Modeling, Verification and Analysis Techniques*, London, Great Britain, Jan 1999, `http://www.systematic-testing.com/et_tt.htm`, Cited 12.3.2004

[Gol89] Goldberg, D. E., *Genetic algorithms in search, optimisation, and machine learning*. Addison Wesley Longman, Inc., 1989. ISBN 0-201-15767-5.

[Gra99] Grajcar, M., Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system.

Proceedings of the 36th Design Automation Conference (DAC), 1999, pages 280–285.

[Gra00a] Grace, P., Applying genetic programming to protocol construction. Master's thesis, Computing Department, Lancaster University UK, Sep 2000. `http://www.lancs.ac.uk/postgrad/gracep/gp&protocols.pdf` Cited 12.3.2004.

[Gra00b] Grajcar, M., Conditional scheduling for embedded systems using genetic list scheduling. *Proceedings 13th International Symposium on System Synthesis.*, Sep 2000, pages 123–128.

[Gro00] Groß, H., *Measuring Evolutionary Testability of Real-Time Software*. Ph.D. thesis, University of Glamorgan, Pontypridd, UK, 2000.

[GS01] Gounares, A. and Sikchi, P., Adaptive problem solving method and apparatus utilizing evolutionary computation techniques, 2001. U.S. patent 6,282,527.

[GW98] Grochmann, M. and Wegener, J., Evolutionary testing of temporal correctness. *Proceedings of the 2nd International Software Quality Week Europe (QWE 1998),* Brussels, Belgium, Nov 1998, `http://www.systematic-testing.com/et_tt.htm` Cited 12.3.2004

[HATJ01] Hanh, V. L., Akif, K., Traon, Y. L. and Jezequel, J., Selecting an efficient oo integration testing strategy: an experimental comparison of actual strategies. *ECOOP 2001 Object Oriented Programming. 15th European Conference. Proceedings Lecture Notes in-Computer Science,* Jun 2001, pages 381–401. 18-22 June 2001 Budapest, Hungary.

[HB01] Heitkötter, J. and Beasley, D., The hitch-hiker's guide to evolutionary computing: A list of frequently asked questions (faq), April 2001. `ftp://rtfm.mit.edu/pub/usenet/news.answers/ai-faq/genetic/` Cited 12.3.2004

[HHP02] Harman, M., Hierons, R. and Proctor, M., A new representation and crossover operator for search-based optimization of software modularization. *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E. and Jonoska, N., editors, New York, 9-13 July 2002, Morgan Kaufmann Publishers, pages 1351–1358.

[HJ01a] Harman, M. and Jones, B. F., Search-based software engineering. *Information et Software Technology*, 43,14(2001), pages 833–839.

[HJ01b] Harman, M. and Jones, B. F., The seminal workshop: reformulating software engineering as a metaheuristic search problem. *ACM SIGSOFT Software Engineering Notes*, 26,6(2001), pages 62–66.

[HKAH96] Hochmann, R., Khoshgoftaar, T., Allen, E. and Hudepohl, J., Using the genetic algorithm to build optimal neural networks for fault-prone module detection. *Proceedings of the Seventh International Symposium on Software Reliability Engineering*, White Plains, USA, Nov 1996, IEEE Computer Society Press, pages 152–162.

[HKAH97] Hochmann, R., Khoshgoftaar, T., Allen, E. and Hudepohl, J., Evolutionary neural networks: a robust approach to software reliability problems. *Proceedings of the Eight International Symposium on*

Software Reliability Engineering, Albuquerque, New Mexico, USA, 1997, IEEE Computer Society Press, pages 13–26.

[HM98] Haikala, I. and Märijärvi, J., *Ohjelmistotuotanto*. Suomen Atk-kustannus Oy, fifth edition, 1998.

[Hol75] Holland, J., Adaptation in natural and artificial systems, 1975. University of Michican Press, Ann Arbor.

[Hun95] Hunt, J., Testing control software using a genetic algorithm. *Engineering Applications of Artificial Intelligence*, 8,6(1995), pages 671–680. Elsevier Science Ltd.

[IEE90] IEEE standard glossary of software engineering terminology. Technical Report 610.12-1990, IEEE, New York USA, 1990.

[JES98] Jones, B., Eyres, D. and Sthamer, H., A strategy for using genetic algorithms to automate branch and fault -based testing. *Computer Journal*, 41,2(1998), pages 98–107.

[JPP00] Jung, B. J., Park, K. I. and Park, K. H., An ordered-deme genetic algorithm for multiprocessor scheduling. *IEICE transaction on information and systems*, E83-D,6(2000), pages 1207–1215.

[JSYE95] Jones, B., Sthamer, H., Yang, X. and Eyres, D., The automatic generation of software test data sets using adaptive search techniques. *3rd International Conference on Software Quality Management*, Seville, Spain, 1995, pages 435–444.

[KA97] Kwok, Y.-K. and Ahmad, I., Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *Journal of Parallel and Distributed Computing. Special Issue on Parallel Evolutionary Computing*.

[KC93] Karananithi, N. and Carpenter, T., A ring loading application of genetic algorithms. Technical Report TM-ARH-023337, Bellcore, Bell Communications Research, Morristown, USA, 1993.

[KC97] Karanunithi, N. and Carpenter, T., Sonet ring sizing with genetic algorithms. *Computers et Operations Research*, 24,6(1997), pages 581–591.

[KG96] Kasik, D. and George, H., Toward automatic generation of novice user test scripts. *Proceedings 1996 Conference on Human Factors in Computing Systems CHI96*, Vancouver, Canada, Apr 1996, ACM Press, New York, pages 244–251, `http://www.acm.org/sigchi/chi96/proceedings/papers/Kasik/djk_txt.htm` Cited 12.3.2004

[KH93] Kim, Y. C. and Hong, Y. S., A task allocation using a genetic algorithm in multicomputer systems. *IEEE Region 10 International Conference on Computers, Communications and Automation*, volume 1, Oct 1993, pages 258–261.

[KH00] Komiya, S. and Hazeyama, A., Projecting risks in a software project through kepner-tregoe program and schedule re-planning for avoiding the risks. *"IEICE-Transactions-on-Information-and-Systems"*, 4, pages 627–639.

[Koz92] Koza, J., *Genetic Programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, Massachussets, USA, 1992.

[KPG95] Kumar, A., Pathak, R. and Gupta, Y., Genetic algorithm based approach for file allocation on distributed systems. *Computers et Operations Research*, 22,1(1995).

[Las01] Laskowski, E., Fast scheduling and partitioning algorithm in the multiprocessor system with redundant communication resources. *Proceedings of 4th International Conference on Parallel Processing and Applied Mathematics*, 2001, pages 97–106.

[LC03] Lee, Y. H. and Chen, C., A modified genetic algorithm for task scheduling in multiprocessor systems, 2003. `http://parallel.iis.sinica.edu.tw/cthpc2003/papers/CTHPC2003-18.pdf` Cited 12.3.2004.

[LK01] Liu, Y. and Khoshgoftaar, T., Genetic programming model for software quality classification. *Sixth IEEE International Symposium on High Assurance Systems Engineering*, pages 127–136.

[LMW99] Lee, B. J., Moon, B. R. and Wu, C. S., Implementation of reusable class library based on corba using genetic algorithm. *Journal of KISSC Computing Practices*, 2,5(1999), pages 209–222.

[Lut02] Lutz, R., Recovering high-level structure of software systems using description length principle. *Artificial Intelligence and Cognitive Science. 13th Irish Conference, AICS-2002. Proceedings Lecture Notes in Artificial Intelligence*, O'Neill, M., Sutcliffe, R., Ryan, C., Eaton, M. and Griffith, N., editors, volume 2464, Limerick, Ireland, Sep 2002, Springer-Verlag, pages 61–69.

[LW02] Lee, B. and Wu, C., Genetic algorithm based restructuring of object-oriented designs using metrics. *IEICE-Transactions-on-Information-and-Systems*, 7, pages 1074–1085. Only abstract.

[MA00] Mantere, T. and Alander, J. T., Automatic image generation by genetic algorithms for testing halftonin methods. *Intelligent Systems and*

Advanced Manufacturing: Intelligent Robots and Computer Vision XIX: Algorithms Techniques, and Active Vision, Casasent, D., editor, Boston, USA, Nov 2000, SPIE Optical Engineering Press, pages 297–308.

[MA01] Mantere, T. and Alander, J. T., Testing a structural light vision software by genetic algorithms - estimating the worst case behavior of volume measurement. *Intelligent Systems and Advanced Manufacturing: Intelligent Robots and Computer Vision XX: Algorithms, Techniques, and Active Vision*, Casasent, D. and Hall, E., editors, Newton, USA, Oct 2001, SPIE Optical Engineering Press, pages 466–475.

[MA02] Mantere, T. and Alander, J. T., Developing and testing structural light vision software by co-evolutionary genetic algorithm. *QSSE 2002 The Proceedings of the Second ASERC Workshop on Quantative and Soft Computing based Software Engineering*, Banff, Alberta, Canada, Feb 2002, Alberta Software Engineering Research Consortium (ASERC) and the Department of Electrical and Computer Engineering, University of Alberta, pages 31–37.

[MA03] Mantere, T. and Alander, J. T., Testing digital halftoning software by generating test images and filters co-evolutionarily. *Proceedings of SPIE. Intelligent Robots and Computer Vision XXI: Algorithms, Techniques, and Active Vision*, Casasent, D., Hall, E. and Roning, J., editors, Oct 2003, pages 257–268.

[Man03] Mantere, T., *Automatic Software Testing by Genetic Algorithms*. Ph.D. thesis, University of Vaasa, Finland, 2003.

[MB94] McCabe, T. J. and Butler, C. W., Software complexity. *Crosstalk, Journal of Defense Software Engineering*, 7,12(1994), pages 5–9.

[MGGZ94] Min, P., Goodman, E., Gao, Z. and Zhong, K., Automated software test data generation using a genetic algorithm. Technical Report 6/2/1994, University of Aeronautics and Astronautics, Beijing, China, 1994.

[MMR$^+$98] Mancoridis, S., Mitchell, B., Rorres, C., Chen, Y. and Gansner, E., Using automatic clustering to produce high-level system organizations of source code. *Proceedings 6th International Workshop on Program Comprehension*, 1998, pages 45–52.

[MMS01] Michael, C., McGraw, G. and Schatz, M., Generating software test data by evolution. *IEEE Transactions on Software Engineering*, 27,12(2001), pages 1085–1110.

[MMSW97] Michael, C. C., McGraw, G., Schatz, M. and Walton, C. C., Genetic algorithms for dynamic test data generation. *Automated Software Engineering*, Lake Tahoe, CA, Nov 1997, pages 307–308

[Mon93] Montana, D. J., Strongly typed genetic programming. Technical Report #7866, 7 1993.

[MSJ00] Mansour, N., Salame, M. and Joumaa, R., Integer- and real-value test generation for path coverage using a genetic algorithm. *Proceedings of the IASTED International Conference. Software Engineering and Applications*, Hamza, M., editor, Las Vegas, USA, Nov 2000, IASTED/ACTA Press, Anaheim, CA, USA, pages 49–54.

[MT95] Minohara, T. and Tohma, Y., Parameter estimation of hyper-geometric distribution software realiability growth model by genetic algorithm. *Proceedings of the Sixth International Symposium on Software Reliability Engineering*, Toulouse, France, 1995, IEEE Press, pages 324–329.

[MW98] Mueller, F. and Wegener, J., A comparison of static analysis and evolutionary testing for the verification of timing constraints. *IEEE Real Time Technology and Applications Symposium*, 1998, pages 144–154.

[Neg02] Negnevitsky, M., *Artificial Intelligence, A Guide to Intelligent Systems*. Pearson Education Limited, 2002. ISBN 0201-71159-1.

[Nis98] Nisbet, A., Gaps: A compiler framework for genetic algorithm (ga) optimised parallelisation, 1998

[OBWK00] Oh, J., Bahn, H., Wu, C. and Koh, K., Pareto-based soft real-time task scheduling in multiprocessor systems. *Proceedings Seventh Asia Pacific Software Engieering Conference ASPEC 2000*, 2000.

[OCW00] Oh, J., Choi, S. and Wu, C., A formal model for allocation of objects into heterogeneous distributed environments. *Journal of Electrical Engineering and Information Science*, 5,1(2000), pages 41–51.

[OVW98] O'Sullivan, M., Vässner, S. and Wegener, J., Testing temporal correctness of real-time systems - a new approach using genetic algorithms and cluster analysis. *Proceedings of the 6th European Conference on Software Testing, Analysis et Review (EuroSTAR 1998)*, Munich, Germany, Dec 1998, `http://www.systematic-testing.com/et_-tt.htm` Cited 12.3.2004

[PD95] Petry, F. E. and Dunay, B. D., Automatic programming and program maintenance with genetic programming. *International Journal of Software Engineering and Knowledge Engineering*, 5,2(1995), pages 165–177.

[PHP99] Pargas, R., Harrold, M. and Peck, R., Test-data generation using genetic algorithms. *Software Testing, Verification and Reliability*, 9. Wiley.

[PN98] Puschner, P. and Nossal, R., Testing the results of static worst-case execution time analysis. *Proceedings of the 19th IEEE Real-Time Systems Symposium RTSS'98*, Madrid, Spain, 1998, IEEE Computer Society Press, pages 134–143.

[PP98] Pedrycz, W. and Peters, J., *Computational intelligence in software engineering*. Advances in Fuzzy Systems - Applications and Theory, Vol. 16. World Scientific Publishing Co. Pte. Ltd., 1998. ISBN 981-02-3503-8.

[PPR02] Portland pattern repository, Dec 2002. `http://c2.com/cgi/wiki` Cited 12.3.2004

[Pre01] Pressman, R. S., *Software engineering: a practitioner's approach*. McGraw-Hill, fifth edition, 2001. ISBN 0-07-365578-3.

[PWW93] Patton, R., Wu, A. and Walton, G., A genetic algorithm approach to focused software usage testing. *Annals of Software Engineering*

[Ram01] Ramakrishnan, S., Object-oriented simulation and ga. *Proceedings of the ISCA 14th International Conference Parallel and Distributed Computing Systems*, Aug 2001, pages 57–61.

[Rec73] Rechenberg, I., *Evolutionsstrategie: Optimierung technischer Systeme und Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

[RL90] Rewini, H. E. and Lewis, T. G., Scheduling parallel program tasks onto arbitrary target machines. *J. Parallel Distrib. Comput.*, 9,2(1990), pages 138–153.

[RMB^{+}95] Roper, M., MacLean, I., Brooks, A., Miller, J. and Wood, M., Genetic algorithms and the automatic generation of test data. Technical Report RR/95/195, University of Strathelyde, London, UK, 1995.

[Rop96] Roper, M., Cast with gas-automatic test data generation via. evolutionary computation. *IEE Colloquium on Computer Aided Software Testing Cast Tools Digest*, London, UK, Apr 1996.

[RPA02] Ramanna, S., Peters, J. and Ahn, T., Software quality knowledge discovery: a rough set approach. *Proceedings 26th Annual International Computer Software and Applications*. IEEE Comput. Soc, Los Alamitos, CA, USA, Aug 2002, pages 1140–1145.

[Rya00] Ryan, C., *Automatic re-engineering of software using genetic programming*. Genetic Programming Series. Kluwer Academic Publishers, 2000. ISBN 0-7923-8653-1.

[SBW01] Sthamer, H., Baresel, A. and Wegener, J., Evolutionary testing of embedded systems. *Proceedings of 14th International Internet et Software Quality Week 2001*, San Francisco, USA, May 2001.

[SC94] Seljak, B. K. and Cooling, J., Optimization of multiprocessor real-time embedded system structures. *Proceedings of MELECON '94 Mediterranean Electrotechnical Conference*, Apr 1994, pages 313–316.

[Sch81] Schwefel, H.-P., *Numerical Optimization of Computer Models*. John Wiley et Sons, Inc., New York, USA, 1981.

[Sem98] Semeraro, G., Evolutionary approach to real-time analysis. *Genetic Programming 1998 Proceedings of the Third Annual Conference*, Jul 1998, page 592.

[SEM02] Software engineering using metaheuristic innovative algorithms (seminal), 2002.
`http://www.discbrunel.org.uk/seminalproject/`
Cited 12.3.2004.

[Ser98] Seredynski, F., Scheduling tasks of a parallel program in two-processor systems with use of cellular automata. *Parallel and Distributed Processing, 10 IPPS/SPDP'98 Workshops Held in Conjunction with the 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing*, volume 1388 of *Lecture Notes in Computer Science*. Springer, 1998, pages 261–305.

[SF96] Smith, J. and Fogarty, T., Evolving software test data-ga's learn self expression. *Evolutionary-Computing AISB-Workshop Selected Papers*, Brighton, UK, 1996, Fac. of Comput. Studies et Math., West of England Univ., Springer-Verlag, Berlin, Germany, pages 137–146.

[SGDJ93] Schultz, A., Grefenstette, J. and De-Jong, K., Test and evaluation by genetic algorithms. *IEEE-Expert*, 8,5(1993), pages 9–14.

[Sha90] Shaw, M., Prospects for an engineering discipline of software. *Software, IEEE*, 7,6(1990), pages 15–24.

[Shu00] Shukla, K. K., Neuro-genetic prediction of software development effort. *Information and Software Technology*, 42,10(2000), pages 701–713.

[SM98] Sandnes, F. E. and Megson, G. M., Improved static multiprocessor scheduling using cyclic task graphs: A genetic approach. *Parallel Computing: Fundamentals, Applications and New Directions, Proceedings of the Conference ParCo'97*, D'Hollander, E. H., Joubert, G. R., Peters, F. J. and Trottenberg, U., editors, volume 12. Elsevier, North-Holland, Sep 1998, pages 703–710

[SMLE02] Shan, Y., McKay, R. I., Lokan, C. J. and Essam, D. L., Software project effort estimation using genetic programming. *Proceedings of International Conference on Communications Circuits and Systems*, 2002.

[SNYF00] Saito, T., Nakanishi, T., Y.Kunieda and Fukuda, A., Genetic algorithm based data and program partitioning. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications: PDPTA'2000*, volume 2, Jun 2000, pages 1173–1179.

[Sth95] Sthamer, H.-H., *The Automatic Generation of Test Data Using Genetic Algorithms*. Ph.D. thesis, University of Glamorgan. Pontypridd, Wales, 1995.

[Sun92] Sundqvist, S., On finding optimal potential customers from a large marketing database - a genetic algorithm approach. *STeP-92 Tekoälyn uudet suunnat*, Hyvönen, E., Seppänen, J. and Syrjänen, M., editors, Espoo, Finland, Jun 1992, Finnish Artificial Intelligence Society (FAIS), pages 35–38.

[Tha97] Thaller, G., *Software Engineering für Echtzeit und Embedded Systems*. Kaarst-Büttgen, 1997. ISBN 3893605428.

[TJJM00] Traon, Y. L., Jeron, T., Jezequel, J.-M. and Morel, P., Efficient oo integration and regression testing. *IEEE Transactions on Realiability*, 49,1(2000), pages 12–25.

[TOK98] Tsuchiya, T., Osada, T. and Kikuno, T., Genetics-based multiprocessor scheduling using task duplication. *Microprocessors and Microsystems*, 22,3-4(1998), pages 197–207.

[Tra00] Tracey, N. J., *A Search-Based Automated Test-Data Generation Framework for Safety-Critical Software*. Ph.D. thesis, University of York, UK, 2000.

[VB96] Vössner, R. and Braunstingl, R., G.o.a.l. (genetic optimization algorithm), 1996. Genetic Optimization Lab, Graz, Austria.

[Wat95] Watkins, A., The automatic generation of test data using genetic algorithms. *Proceedings of the 4th Software Quality Conference*, et al., I. M., editor, Dundee, Scotland, UK, Jul 1995.

[WBS02] Wegener, J., Baresel, A. and Sthamer, H., Suitability of evolutionary algorithms for evolutionary testing. *Proceedings of 26th Annual International Computer Software and Applications Conference, 2002. COMPSAC 2002,* Aug 2002, pages 287–289.

[WG98] Wegener, J. and Grochtmann, M., Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Systems*, 6,2(1998), pages 275–298

[WG99] Wadekar, S. and Gokhale, S., Exploring cost and reliability tradeoffs in architectural alternatives using a genetic algorithm. *10th International Symposium on Software Reliability Engineering 1999*, November 1999, pages 104–113.

[WGJ97] Wegener, J., Grochtmann, M. and Jones, B., Testing temporal correctness of real-time systems by means of genetic algorithms. *Proceedings of the 10th International Software Quality Week (QW '97)*, San Francisco, USA, May 1997.

[Whi95] Whitten, T., Method and computer program product for generating a computer program product test that includes an optimized set of computer program product test cases, and method for selecting same, 1995. U.S. patent 5,805,795.

[WJHZ98] Wei, J., Junkai, X., Hongyu, X. and Zhongyi, G., Genetic algorithms and its application in software test data generation. *Journal of Beijing University of Aeronautics and Astronautics*, 24,4(1998), pages 434–437.

[WPS99] Wegener, J., Pohlheim, H. and Sthamer, H., Testing the temporal behavior of real-time tasks using extended evolutionary algorithms. *Proceedings of the 7th European Conference on Software Testing, Analysis and Review (EuroSTAR '1999)*, Barcelona, Spain, Nov 1999, `http://www.systematic-testing.com/et_tt.htm`, Cited 12.3.2004

[WYKH97] Woo, S. H., Yang, S. B., Kim, S. D. and Han, T. D., Genetic scheduling algorithms in distributed computing systems. *Journal of KISSA Computer Systems and Theory*, 24,12(1997), pages 1247–1256.

[XES+92] Xanthakis, S., Ellis, C., Skourlas, C., Le-Gall, A., Katsikas, S. and Karapoulios, K., Application genetic algorithms applications to software testing. application of genetic algorithms to software testing. *Proceedings of the 5th International Conference on Software Engineering et and Its Applications.*, Toulouse, France, Dec 1992, EC2, Nanterre, France, pages 625–636.

[Xil98] Xile, Y., *Automatic Testing From Z specifications*. Ph.D. thesis, University of Glamorgan, Pontypridd, UK, 1998.

[YG02a] Yun, Z. and Guoyong, H., Real-time multi-processor cosynthesis using ga scheduling. *Transactions of the Institute of Electronics, Information and Communication Engineers*, 12,J85-C(2002), pages 1216–1228.

[YG02b] Yun, Z. and Guoyong, H., System level software/hardware partitioning by genetic algorithm. *Journal of Computer Aided Design et Computer Graphics*, 14,8(2002), pages 731–734.

[Yin93] Yin, X., Application of genetic algorithms to multiple load flow solution problem in electrical power systems. *Proceedings of the 32nd IEEE Conference on Decision and Control*, San Antonio, USA, Dec 1993, IEEE Control Systems Society, pages 3734–3738.

[YQN01] Yi, S., Qin, T. W. and Nian, Y. S., Genetic algorithm approach towards scheduling dag on multiprocessor. *Journal of Shanghai University*, SUP,5(2001), pages 86–91.

[YZC99] Yanyuan, Z., Zhining, J. and Cuilan, Z., Research on automatic testing of a kind of software by using generalized genetic algorithm. *Fifth International Conference for Young Computer Scientists ICYCS'99. Advances in Computer Science and Technology*, Luo, J., Xu, B., Wang, Y., Li, X. and Lu, J., editors, volume 1, Nanking, China, Aug 1999, Int. Acad. Publishers, Beijing, China.