Solving NP-Complete Problems in Real-Time System Design by Multichromosome Genetic Algorithms

Roman Nossal Thomas M. Galla

Institut für Technische Informatik
Vienna University of Technology
Treitlstr. 3/182-1, A-1040 Vienna, Austria
email: {nossal,tom}@vmars.tuwien.ac.at

January 10, 1997

Abstract

Most problems in the design of real-time applications like task allocation or scheduling belong to the class of NP-complete problems and can be solved efficiently only by heuristics. Genetic Algorithms are a relatively new method to attack these problems. Conventional Genetic Algorithms, however, have a number of drawbacks that reduce their applicability to design problems of real-time systems.

The Genetic Algorithm presented in this paper implements enhancements to standard Genetic Algorithms to eliminate these problems. It allows arbitrary gene values and supports multiple chromosomes per individuum. The paper focuses on the advantages of the use of Genetic Algorithms in real-time system design in comparison to other heuristic problem solving techniques. The applicability of the enhanced algorithm is shown by solving a typical problem of real-time system design, the determination of a bus access schedule for a real-time LAN.

1 Introduction

During the design process of a distributed real-time application a number of decisions must be made. How many node computers should be used? How should the tasks be allocated to the

nodes? Which execution order meets the temporal demands of the application best? Most of these problems are *NP complete* [Gar79]. Even if one of them can be solved in polynomial time the overall problem is still in NP. Heuristics are an efficient means to handle these problems. In the past two decades a number of heuristic techniques have been proposed. These include heuristic branch-and-bound algorithms like IDA* [Kor85], stochastic hill climbing algorithms like Simulated Annealing [Kir83] resembling the slow cooling of materials and Tabu Search [Glo93] operating with lists of forbidden moves, and Genetic Algorithms (GA) [Hol75]. The latter work like evolution in nature, they are based on Darwin's "Survival of the Fittest" principle [Dar59]. What makes GAs so attracting to the real-time application designer is that they relieve him from knowing how to construct a solution and just require to know how to assess a given solution. We have applied a GA to various problems arising in the design process of real-time applications. On the one hand this has taught us how to tackle problems with GAs, i.e., how to adapt the problem representation and the GA to each other. On the other hand we have learned how a GA suited for solving design problems must work.

This paper has two main goals. First it presents our implementation of an enhanced GA[1] featuring extensions to conventional GAs that increase their suitability for real-time application design. The most prominent enhancement is the introduction of the *multichromosome* feature. Unlike conventional GAs which are capable of only one chromosome per individuum our algorithm features the use of multiple chromosomes. Each of them expresses a certain characteristic of the problem, which resembles the genotype in nature. The second aim of the paper is to line out the application of the GA to one problem of real-time application design, namely the static assignment of communication bandwidth to node computers depending on the amount of application data each node has to transmit.

The rest of this paper is organized as follows. Section 2 gives an overview of GAs in general. The main structure and the different stages of a GA are discussed. The improvements made to standard GAs are stated in section 3. In section 4 the benefits of GAs compared to other heuristic techniques are summarized. Section 5 deals with the application of a GA. It gives a brief outline of the real-time system TTP and presents how a multichromosome GA is applied to

[1]We are planning to make the Genetic Algorithm described in this paper available via FTP. At the time being we are checking the legal situation.

the message scheduling problem in the communication system of TTP. Section 6 concludes the paper and presents a brief outlook on some future applications of multichromosome GAs.

2 Genetic Algorithms

Genetic Algorithms are a relatively new strategy for solving NP-complete problems. They were first introduced by Holland in 1975 [Hol75], yet many applications of this strategy have emerged only in the past few years. This section gives an introduction into the basic principles and mechanisms of GAs. Good surveys of GAs including the latest developments can be found in [Sri94, Gol89].

2.1 Characteristics

GAs are a so-called "uninformed" search strategy, i.e., they do not incorporate any knowledge on the special problem they solve. All problem specific knowledge is included in the *fitness function* that assigns a *fitness value* to each solution produced by the GA. This value is used as a feedback for the algorithm. The better a solution's fitness is, the more likely is its survival and reproduction.

As indicated the problem specific knowledge is not part of the GA itself. Rather, GAs use *bit-strings* to encode the problem. The chosen encoding scheme is called the *problem representation*. A certain number of bits, a *gene* forms one part of the problem.

A problem representation for GAs must satisfy two goals. On the one hand it should encode the whole solution space, i.e., the problem representation must not exclude certain parts of the possible solutions. On the other hand it should be easy to decode. A substantial part of a GA execution is spent in the fitness function[2] because before evaluating the quality of a solution the function must decode the problem to its original representation. This implies that easing the decoding has a large impact on the overall execution time of the algorithm.

[2]Experiments with the GA based scheduler described in section 5 have shown that up to 95% of the execution time are consumed by the fitness function.

2.2 Structure

The main actors in GAs are so called *individuums* that comprise one *chromosome* each. A chromosome consists of a number of genes, i.e., bitstrings. One individuum represents *one possible solution* to the problem that has to be solved by the GA. Each individuum is encoded in one or more bitstrings. This representation is equivalent to the *genotype* in genetics, whereas the actual "appearance" (i.e. the solution the individuum represents) corresponds to the *phenotype*. Individuums are grouped together in *populations* (figure 1).

The GA starts with an initial population that is either selected randomly or specified by the user. New individuums are created out of old ones by selecting and reproducing individuums of the old population. Like in nature two main processes, *crossover* and *mutation* occur during reproduction. Crossover splits up individuums into parts, exchanges and recombines them. Mutation changes an individuum randomly thus offering the possibility to introduce new features into a population. The new individuums form a new population, whereas the individuums of the old population "die". Each population is called a *generation*. The "evolution" stops, when a solution of sufficient quality, i.e., an individuum with sufficient fitness, is found.

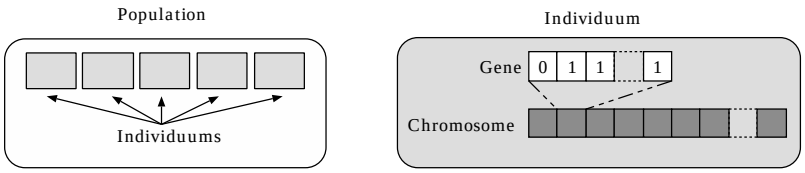

Figure 1: Structure of a Population

2.3 Stages

A GA performs several iterations producing a new generation in each iteration. One iteration of a GA consists of four sequential stages, **selection** of individuums with a selection probability proportional to the fitness, **mutation** of some of these individuums according to a *mutation rate*, **crossover** between some of the selected individuums according to a *crossover rate*, and **evaluation** of the fitness of each new individuum. Each of these steps will be discussed in more detail in the following sections.

Selection

The selection process decides which individuums of the old population are used to form new individuums for the new population, i.e., which individuums are allowed to create offspring.

In order to imitate Charles Darwin's *Survival of the Fittest*, it is a common approach to base this choice of individuums on their respective fitness (the ones with a high fitness are more likely to be chosen). In addition a special selection strategy, the *elitist strategy*, is used. The elitist strategy stipulates that the individuum with the best fitness *always* survives and makes its way into the new population without being altered.

Mutation

The selected individuums are mutated according to a *mutation rate*. This mutation rate defines the percentage of individuums that are to be mutated.

If a specific individuum is mutated, an arbitrary part of a gene (a random bit of the bitstring representation of the gene) is changed. Figure 2 gives an example, where the second bit of the third gene (this bit is colored in gray) is mutated by changing the value from 1 to 0.

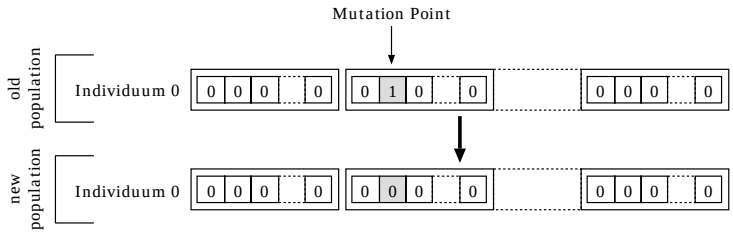

Figure 2: Mutation

Crossover

Crossover splits the chromosomes of two individuums into several parts, swaps the parts and recombines them. The number of parts a chromosome is split into is determined by the number of crossover points. A *two point crossover* splits the chromosome into three parts by two crossover points (figure 3). The middle part (indicated by gray background in figure 3) is swapped.

The percentage of individuums of the population that are to be combined by crossover is defined by the *crossover rate*.

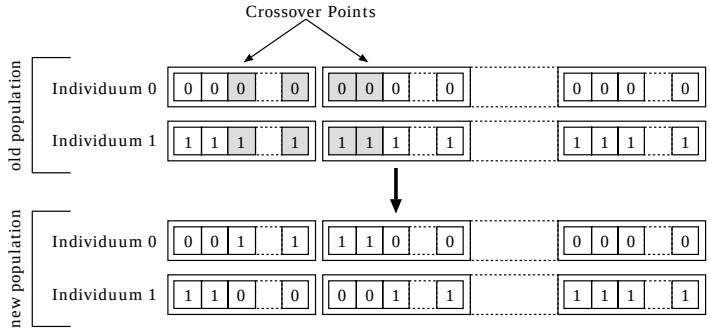

Figure 3: Two Point Crossover

Evaluation

Subsequent to selection, mutation and crossover the fitness of the newly created individuums is evaluated using the fitness function.

This function provides the actual "intelligence" of the algorithm. It transforms the bitstring representation (the genotype) of the individuum back into the actual solution (the phenotype). This solution is *evaluated* by computing a *fitness value* that is used to decide which of two individuums provides a better solution to the problem.

After the individuums have been assessed, the whole process is started anew. Usually it is iterated until a defined fitness has been reached or a maximum number of iterations has been exceeded.

3 Enhancements

The GA described so far is a conventional version. In the following we will present two enhancements that increase the suitability of the GA for real-time application design problems. The enhanced algorithm has been implemented in *C*.

3.1 Arbitrary Gene Values

Most GAs (e.g., Grefenstette's GA package GENESIS [Gre90]) restrict the value of a gene to the interval $[0, 2^n)$[3] where $n \in N$. Unfortunately for most problems an upper bound for the gene values cannot be restricted to a power of 2. Therefore a more complex encoding scheme must be used in order to facilitate gene values of the interval $[0, x_{max})$ $(x_{max} \in N)$.

[3] $[x, y)$ denotes a semi-open interval, i.e., all values $i | x \leq i < y$.

An elegant solution to this problem is to choose $[0, x_{max})$ as interval for all valid gene values (i.e. to allow the upper bound for the valid genes to be any positive integer). This approach provides more flexibility and simplifies the encoding scheme. One drawback of this solution is the fact that there are bitstrings that represent invalid genes (i.e. genes with values greater than or equal to x_{max} and smaller than 2^n where $2^{n-1} < x_{max} < 2^n \quad n \in N$). Any mutation or crossover process can result in an invalid gene. Thus the existing operators have been modified in order to produce only valid genes.

The GA described in this paper only restricts the gene value to the interval $[0, x_{max})$ where $x_{max} \in N$ and is therefore well suited for design problems.

Mutation. Mutation is basically applied at bit level. This, however, may result in invalid genes. Figure 4 shows an example, where the value of a gene is restricted to the interval $[0, 9)$. In this example the mutation of the gene (that has an initial value of 8) results in an invalid gene.

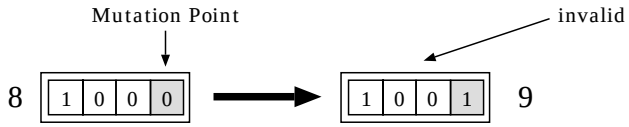

Figure 4: Invalid Genes After Mutation

If mutation produces an invalid gene, the gene is mutated a second time, now at the decimal level. A random number out of the interval of valid gene values $[0, x_{max})$ is chosen thus guaranteeing a valid gene after this second mutation.

Crossover. Like mutation, crossover is performed at the bit level. Therefore crossover can create invalid genes as well. In this case the genes again must be mapped into the valid interval. To do so we apply the *Stochastic Mapping* operator to each invalid gene. This operator partitions the invalid values proportionally to the number of allowed gene values. More than one valid value is assigned to each invalid one as the number of invalid values is always smaller than the number of allowed values[4]. A valid gene is calculated out of the invalid value by a random function. Figure 5 illustrates in a simplified manner how the mapping of old values to new ones is performed.

[4] otherwise it would be sufficient to use one bit less for the representation of the gene

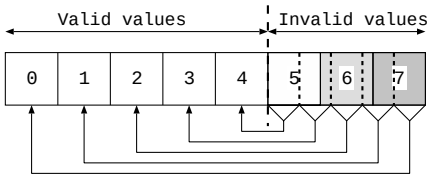

Figure 5: Stochastic Mapping of Invalid Genes

3.2 Multiple Chromosomes

Structure. Each individuum has its own specific features which are defined by its genotype. It is a common approach in existing GAs to encode these features within a single chromosome.

An obvious drawback of this technique is that no strict separation of independent features exists. Thus whole features might be exchanged during the crossover process, which leads to unwanted, significant changes in an individuum. Consider the crossover example given in figure 6 where feature 1 would be completely exchanged between the two individuums.

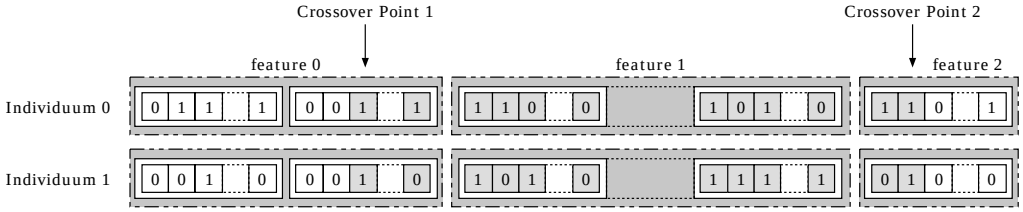

Figure 6: Mixture of Features in Single Chromosome GAs

A more sophisticated approach, the *multichromosome* GA, tries to overcome this problem. As suggested by the name an individuum of a multichromosome GA consists of more than one chromosome (see figure 7). Each of the independent features can be assigned to a different chromosome. Thus it is impossible to exchange whole independent features during crossover. This approach is the more natural way to handle things (the human species has 23 pairs of chromosomes – each carrying different features). On the other hand multichromosome GAs still facilitate the assignment of more than one feature to one chromosome. It is the user's task to decide whether an exchange of whole features should be possible or not, i.e, whether features should be assigned to one or to different chromosomes.

To be able to handle this new structure of individuums and feature encoding, the mutation and the crossover operator have to be adapted.

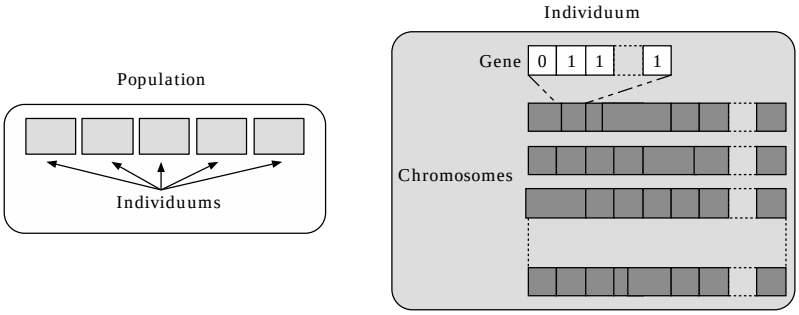

Figure 7: Structure of a Multichromosome GA

Adaptation of the Mutation Operator. The modifications to the mutation operator, in order to be able to deal with multiple chromosomes per individuum, are straight forward. A random chromosome of the individuum is chosen. For this chromosome the standard operator described in section 2.3 with the enhancements of section 3.1 is used to mutate a single bit.

Adaptation of the Crossover Operator. For the new crossover operator two different approaches are possible. Either

- only a single chromosome is used for crossover, or
- all chromosomes of an individuum undergo the crossover process.

In the first case a single chromosome of each individuum is randomly chosen, and the enhanced operator (section 3.1) is applied. In the second case the standard operator is applied to each chromosome of the two individuums separately. Thus more genetic information is exchanged in the second version making it more likely to create completely new solutions. For our implementation we have used the second modification of the crossover operator.

4 Benefits of Genetic Algorithms

So far we have explained the general function of GAs and the extensions we have applied to increase their suitability for real-time application design. This section summarizes the benefits of using GAs instead of other heuristic problem solving techniques.

Solution Evaluation Instead Of Construction. The most obvious benefit is that GAs relieve the user of knowing how to construct a solution. In order to implement a "standard"

problem solving technique the user is required to know how to construct a solution to a problem. There has to be an algorithm that yields a valid solution to the given problem. For example, consider a branch-and-bound algorithm like IDA* [Kor85]. Each branch in such an algorithm is a step towards a solution. If the algorithm finishes successfully the way from the root of the search tree to the leaf node represents the solution.

The situation is completely different when using GAs. GAs use the problem in an encoded form, they operate on bitstrings. Thus a GA constructs a solution by rearranging bitstrings. These construction steps are not defined by the user, they are inherent to the GA. It is the task of the evaluation function to decode the solution to its original representation and then to assess the solution. This means the user has to provide knowledge on the quality of a solution, i.e., how to evaluate a given solution.

In most cases assessing a solution is much easier than constructing it. For example it is easy to determine if a given number is the square root of 7 by simply multiplying it with itself. Calculating the square root of 7 on the other hand is a tedious task if performed without a calculator or computer.

Problem Structuring. Besides defining a fitness function finding an appropriate representation of the problem is one of the main tasks of the user of a GA. A prerequisite for deriving a representation is a deep understanding of the problem. To gain this understanding it is inevitable to structure the problem. The determinants of the problem, the entities involved, the optimization goal and the variables that can be changed during the search process must be identified.

For many problems the problem structure is obvious. On the other hand, there are problems for which the user has various possibilities to choose the variables and the optimization goal. A problem belonging to the first class is the task allocation problem. The starting point is a set of tasks, a set of nodes on which these tasks should be executed and the communication cost between any two nodes. The aim is to allocate the tasks to nodes such that the accumulated communication cost between the tasks is minimized. In this case it is apparent to represent the numbers of the nodes the tasks should be allocated to as bitstrings and to choose the communication cost as optimization goal.

An example for a problem where the structure is not as obvious as in the task allocation

problem is the task scheduling problem. The problem aims at scheduling tasks on a CPU such that the overall time span to complete the tasks is minimized and the timing constraints are met. In this problem one can either use the sequence of tasks as variable or let the GA select the ready times of the tasks.

Integration of Various Problems. GAs offer a simple way to integrate different search procedures into one optimization process. Thus different problems can be solved in one step, providing a global optimum instead of local optima for each search.

Each of the involved problems is encoded into a number of chromosomes and has its own fitness function. The problems can be solved in one step by (1) constructing an individuum that consists of the chromosomes of all involved problems and (2) by deriving an appropriate algebraic conjunction of the fitness functions. In many cases it will be sufficient to simply sum up the individual fitnesses.

An application for the integration is the task allocation and scheduling problem. It consists of two basically separate problems, allocation of tasks to nodes and scheduling of the allocated tasks (see above). Goal of the scheduling is to minimize the time span required to complete all tasks. Solving both problems in two separate search processes yields a local optimum for each step, the combination of which will be different from the global optimum in most cases. If allocation and scheduling are integrated the GA minimizes the communication cost between the nodes and at the same time the overall time required to finish all tasks.

5 An Application of GAs

In this section we will show how to apply the multichromosome GA that we have developed to a real-world problem of system design. During application design for the real-time system TTP built by our group the problem to allocate application messages to bus messages and to schedule those bus messages on the communication medium arises. This task is similar to the famous bin-packing problem [Gar79] which is known to be NP-complete. In the following we will provide a brief introduction to the real-time system TTP and then elaborate on the message scheduling process.

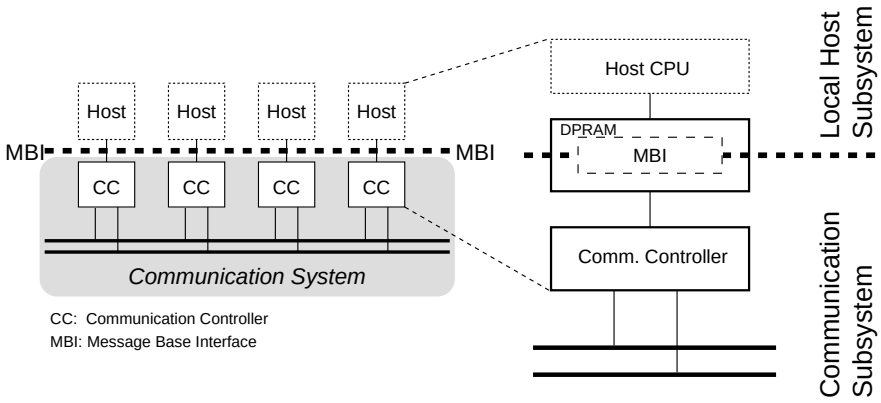

Figure 8: TTP System Structure

5.1 The Real-Time System TTP

Real-Time System. A real-time system is a computer system that closely interacts with an controlled object in its environment. For a real-time system providing timely results is as important as providing correct results since the controlled object must not and often can not be slowed down by the computer system. The timing requirements of the system are determined by the application.

System Structure. A TTP system consists of a set of self-contained computers, the nodes, which are interconnected via a broadcast bus. One broadcast domain, i.e., one bus together with the nodes attached to it, is called a *cluster*. A cluster and each node consist of two subsystems, the communication and the host subsystem (see figure 8). The latter performs the real-time application. Our application model comprises tasks and messages exchanged between them, the so-called *data elements*. The communication subsystem transports *bus messages* between the nodes of a cluster using the communication protocol TTP/C[5]. Each bus message contains a number of data elements. The two subsystems interact with each other via the *Message Base Interface (MBI)* [Krü95] that is located in a dual-ported memory between the two CPUs.

The Time Triggered Protocol. TTP/C [Kop94] is a round based protocol. Medium access is controlled by a synchronous time division multiple access (TDMA) scheme derived from a global time base. The global time base is established by a protocol inherent clock synchronization

[5]The "C" refers to the classification of vehicle multiplexing systems as introduced by the Society of Automotive Engineers [SAE93].

mechanism. The points in time when to send a message are determined by the message scheduling algorithm described below.

In one round of communication, a *TDMA cycle*, each node is assigned exactly one sending slot. One or more TDMA cycles together form a *cluster cycle*. The messages sent in a cluster cycle may vary for each TDMA cycle, which facilitates the transmission of data elements with update periods longer than one TDMA cycle. Message transmission in TTP/C is strictly periodical, the shortest possible message period being one TDMA cycle.

5.2 Message Scheduling

Message scheduling comprises the allocation of data elements to bus messages, taking into account the requirements originating from the real-time application as well as those dictated by the communication protocol. The most important of these requirements is the *Update Frequency*. A data element is updated whenever the task producing it finishes. Thus the update frequency is equal to the period of the sending task. In order not to lose any value the update frequency is the lower bound for the transmission frequency of the data element on the bus. This is the most important requirement for the scheduling algorithm since it reflects the temporal constraints of the application. The reciprocal value of the minimum update frequency is called *maximum period*, p_{max}.

There are a number of further demands dictated by the protocol or certain protocol mechanisms like a restriction of the maximum size of a bus message. Listing all these constraints and the appropriate scheduling algorithm is beyond the scope of this paper. Our intention is to give an impression of the basic principles of scheduling based on GAs. The reader interested in all details of the message scheduling process is referred to [Nos96]. In this paper the message scheduler which is a major part of the *Cluster Compiler* [Kop95], the design tool for TTP, is described in a comprehensive manner.

The aim of message scheduling is to find a *valid message schedule*, i.e. an assignment of data elements to bus messages that satisfies all constraints, in particular the temporal constraints of the application.

5.3 A Scheduler Based on GAs

The following subsection presents the GA based message scheduler. It focuses on the two main problems of applying a GA, finding an adequate problem representation and deriving a fitness function that provides a good classification of individuums.

Problem Representation

The aim of the message scheduler is to produce a valid assignment of data elements to bus messages. The assignment of one data element can be described by the "start cycle", i.e., the first TDMA cycle the data element is transmitted in and the period at which it is transmitted ("period" gene). This period is given in TDMA cycles as well. In our implementation we have restricted both the number of TDMA cycles per cluster cycle and the periods of data elements to powers of 2. This guarantees that the number of TDMA cycles forming a cluster cycle is always a multiple of any data element period avoiding large cluster cycles being the least common multiple of all data element periods. Each data element is thus described by two genes, one encoding the start cycle and the other denoting the period. The two genes are located in two chromosomes making use of the multichromosome feature of the GA.

Fitness Function

The fitness function of the message scheduler uses the so-called *period ratio* to evaluate the quality of a solution. The period ratio is the quotient of the actual period of the data element as determined by the GA and the maximum period p_{max} specified in the temporal requirements of the application. The fitness function, which is shown in figure 9, has a shape that is comparable to an inverted "V". It has a maximum at a period ratio of 1, i.e., a data element period equal to the application's demands. This point is optimal since neither bandwidth is wasted due to scheduling the data element more often than necessary nor temporal constraints are violated. For period ratios smaller than 1, i.e., an "overscheduling" of data elements, the function shows a moderate descent. Period ratios larger than 1 denote a violation of temporal requirements and are thus reflected by a steep descent.

The fitness of an individuum is calculated by summing up the fitness values of the individual data elements and dividing the sum by the number of data elements.

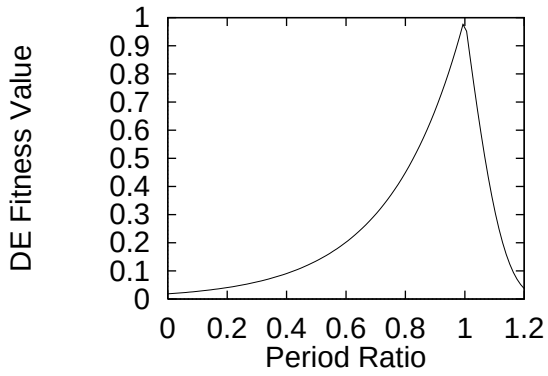

Figure 9: Message Scheduler Fitness Function (DE...Data Element)

Overall Evaluation

Above we have shown how the GA evaluates one individuum. The complete evaluation process including the decoding procedure is described in the following.

The evaluation function takes one individuum created by the GA as input. It assigns the data elements to the bus messages of the respective sending node according to the gene values. Note that there is exactly one bus message per node per TDMA cycle. Thus each data element is assigned to the bus message of its sending node. First a data element is assigned to the bus message of its sending node in the TDMA cycle denoted by the start cycle. Subsequently the period is added to the number of the start cycle and the data element is assigned to the message in the resulting cycle. This process is iterated until the maximum number of TDMA cycles per cluster cycle is reached.

Based on the assignment the evaluation function calculates the actual length of a TDMA and cluster cycle and determines the fitness of one data element and the individuum as described in the previous subsection. Finally it checks if all other protocol specific constraints like the maximum message size are met. For each violated constraint a so-called *penalty term* is calculated which reduces the fitness value. In our scheduler each violation yields a fitness reduction of 10% of the original value. Applying the penalty term to the original fitness value results in the overall fitness of the individuum under evaluation.

Stopping Criterion

As indicated the GA stops when a solution of sufficient quality is found. We have chosen an overall fitness slightly smaller than 1 (e.g., 0.97) as stopping criterion for the GA. The resulting

solution does not guarantee that all data elements meet their temporal requirements. Yet a fitness value smaller than 1 does not imply that at least one data element is not scheduled according to its maximum period. The fitness of one data element decreases for period ratios smaller than the period ratio 1 though these period ratios result in a temporally valid schedule. Hence each data element has to be checked in order to determine whether the actual solution meets the maximum period requirement.

6 Conclusion

In this paper we presented a GA with new features that simplify the use of the GA in real world problems and provide more flexibility than the common approaches in GAs.

A less restrictive interval for all possible gene values was chosen, which facilitates the encoding and decoding task in the evaluation function by providing a more straight forward encoding scheme. New operators for crossover and mutation were introduced to deal with the invalid genes that might result form this less restrictive interval.

Furthermore we showed that multiple chromosomes facilitate a better encoding of independent features of an individuum. Assigning independent features to different chromosomes avoids the exchange of whole features during the crossover process.

The benefits of GAs in comparison to other heuristic techniques were pointed out, and the use of the presented multichromosome GA in a real world application (as a message scheduler for the real-time system TTP) were explained in detail.

The GA based message scheduler described in this paper is part of the Cluster Compiler, the software development tool of the real-time system TTP. Besides the message scheduler the Cluster Compiler comprises a task allocation module and a task scheduler.

At the moment we are testing the implementation of a task scheduler that is based on the same GA as the message scheduler. The GA determines the arrival times of the tasks which are then scheduled using a conventional least laxity scheduler. The resulting schedule is evaluated with respect to the deadlines of the tasks. Though intended for the difficult problem of multiprocessor scheduling with constraints among the tasks the results encountered up to now are encouraging. We also plan to integrate both, task and message scheduling into one optimization process.

References

[Dar59] C. Darwin. On the Origin of Species by means of Natural Selection; or, The Preservation of Favoured Races in the Struggle for Life, 1859.

[Gar79] M.R. Garey and D.S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-completeness.* W.H. Freeman Co., San Francisco, CA, 1979.

[Glo93] F. Glover, E. Taillard, and D. de Werra. User's Guide to Tabu Search. *Annals of Operations Research*, 41, 1993.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[Gre90] J.J. Grefenstette. A User's Guide to GENESIS, Version 5.0. Technical report, Department of Computer Science, Vanderbilt University, Nashville, 1990.

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, 1975.

[Kir83] S. Kirkpatrick, C. Gelatt, and M. Veechi. Optimization by Simulated Annealing. *Science*, 220:671–680, May 1983.

[Kop94] H. Kopetz and G. Grünsteidl. TTP — A Protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer*, pages 14–23, January 1994.

[Kop95] H. Kopetz and R. Nossal. The Cluster Compiler — A Tool for the Design of Time-Triggered Real-Time Systems. In *ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Real-Time Systems*, La Jolla, California, USA, June 1995.

[Kor85] R. Korf. Depth-First Iterative-Deepening: An Optimal Admissable Tree Search. *Artificial Intelligence*, 27(3):97–109, 1985.

[Krü95] A. Krüger and H. Kopetz. A Network Controller Interface for a Time-Triggered Protocol. In *SAE Symposium on Future Transportation Electronics: Multiplexing and In-Vehicle Networking.* Society of Automotive Engineers, August 1995. SAE Paper No. 952576.

[Nos96] R. Nossal. A Pre-Runtime Planning Algorithm for Real-Time Communication Systems. Research Report 1/96, Institut für Technische Informatik, Technische Universität Wien, Vienna, Austria, March 1996. Submitted for publication at the 17th International Conference on Distributed Computing Systems.

[SAE93] Class C Application Requirement Considerations. SAE Recommended Practice J2056/1, SAE, June 1993.

[Sri94] M. Srinivas and L.M. Patnaik. Genetic Algorithms: A Survey. *IEEE Computer*, pages 17–26, June 1994.

Acknowledgments

This work was in part supported by the European Commission under Project No. BRPR-CT95-0032 (DG12-RSMT) "Safety Related Fault Tolerant Systems in Vehicles (X-by-Wire)" and under ESPRIT Long Term Research Project No. 20072 "Design for Validation (DeVa)" and by HP Labs, Palo Alto, California.