TDMA Time Slot and Turn Optimization with Evolutionary Search Techniques

Arne Hamann, Rolf Ernst
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig
D-38106 Braunschweig / Germany
{**hamann**|**ernst**}**@ida.ing.tu-bs.de**

Abstract

In this paper we present arithmetic real-coded variation operators tailored for time slot and turn optimization on TDMA-scheduled resources with evolutionary algorithms. Our operators implement a heuristic strategy to converge towards the solution space and are able to escape local minima. Furthermore, we explicitly separate the variation of the admitted loads and the turn-length in order to give the designer increased control over the optimization process. Experimental results show that our variation operators have advantages over string-coded binary variation operators which are frequently used to solve continuous optimization problems.

I. Introduction

Design space exploration for heterogeneous MpSoC or distributed systems is a tedious task due to many design parameters with very different effects. Parameter selection is difficult enough for discrete design parameters, such as task priorities, but becomes even more challenging for continuous parameters, such as TDMA timing.

One popular approach in walking through the search space is the use of evolutionary search techniques. However, their effectiveness strongly depends on the coding of the problem variables as well as the used crossover and mutation operators.

While for discrete and permutation problems efficient coding techniques and variation operators are known to achieve good solutions, continuous optimization problems, like TDMA time slot assignment, challenge the search-power of string-coded binary evolutionary algorithms.

In this paper we propose arithmetic real-coded variation operators tailored for time slot and turn optimization on TDMA-scheduled resources. Thereby, our strategy of walking through the search space is split into two aspects: optimizing the admitted loads of the mapped tasks as well as optimizing the TDMA turn-length. Both factors together define the quality of a time slot assignment. According to these two important factors we introduce four variation operators. One crossover and one mutation operator that vary the admitted loads of the mapped tasks while letting the turn-length constant as well as one crossover and one mutation operator that vary the turn-length and make sure that the admitted loads of the tasks stay constant.

The crossover operators implement a heuristic strategy of converging towards solutions lying ”between” individuals currently considered by the evolutionary algorithm, whereas the mutation operators serve to break out of local minima by increasing or decreasing the admitted loads and the turn-length, respectively, within certain limits.

For experiments, this approach is used for local optimization of TDMA-scheduled resources within a design space exploration framework for global optimization of heterogeneous MpSoC and distributed systems [5].

II. Design Space Exploration Framework

Figure 1 shows the design space exploration loop performed in our framework [5]. The *Optimization Controller* is the central element. It is connected to scheduling analysis and to an evolutionary multi-objective optimizer. Scheduling analysis checks the validity of a given system parameter set, that is represented by an individual, in the context of the overall heterogeneous system. The evolutionary multi-objective optimizer is responsible for the problem-independent part of the optimization problem, i.e. elimination of individuals and selection of interesting individuals for variation. Currently, we use SPEA2 (Strength Pareto Evolutionary Algorithm 2) [12] for this part, which is coupled via PISA (Platform and Programming Language Independent Interface for Search Algorithms) [1].

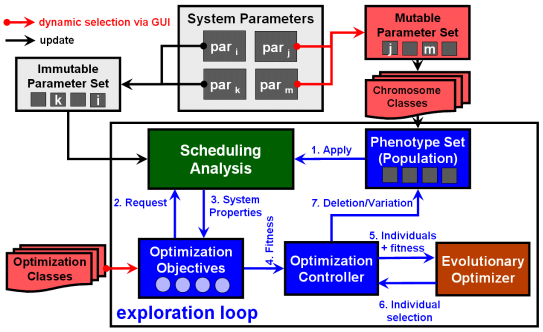


Fig. 1 - Exploration Framework

Different parameters of a system, such as time slots or priorities, are encoded on separate chromosomes. The user selects a subset of all parameters for optimization. The chromosomes of these parameters form an individual and are included in the evolutionary optimization while all others are fixed and immutable. The variation operators of the evolutionary algorithm are applied chromosome-wise for these individuals. More details on the optimization system can be found in [5].

In this paper we explain in detail one possible search parameter in our exploration framework, the time slot

assignment on TDMA-scheduled resources. The results are, however, applicable beyond this exploration system.

The remainder of this paper is structured as follows. After an overview of related work, we introduce the proposed variation operators which are used within the framework to optimize the time slot assignments of tasks mapped on TDMA-scheduled resources. By means of extensive experiments we then compare the new operators with binary-coded single-point crossover and binary-coded mutation.

III. Related Work

There is a large body of work in the area of design space exploration and optimization of heterogeneous MpSoC and distributed systems. In the following we give a small overview of approaches for the optimization of different system parameters as well as frameworks allowing to explore given systems at different levels of abstraction.

The approach described in [11] introduces an analysis technique to estimate end-to-end packet delays and queuing memory in network processor architectures. Based on this information a measure is defined to characterize the performance of such architectures under different usage scenarios. By means of design space exploration pareto-optimal architectures are searched trading good performance under several usage scenarios versus cost. In [7] the authors treat the reverse problem. Instead of determining worst-case buffer requirements and output stream properties for given input streams and scheduling policies, the authors search for the input stream rates that can be supported by a given stream processing architecture without violating on-chip buffer constraints. The authors propose the integration of this technique into a tool for automated design space exploration for fast performance evaluation of different stream processing architectures.

[3] presents a heuristic algorithm for priority assignments in distributed hard real-time systems to optimize end-to-end deadlines. The algorithm iteratively decomposes the global deadlines into artificial local deadlines and then assigns deadline monotonic priorities on each resource. This approach is thus not applicable to more general priority assignments or other types of scheduling policies.

The approach in [9] focuses on bus access optimization (TDMA and static priority preemptive) in multi-cluster embedded systems interconnected via gateways. Thereby, the application structure is feed-forward. Optimization objectives are end-to-end deadlines. The authors propose a partitioning and mapping heuristic and a heuristic adjusting TDMA slot sizes in time-triggered clusters. For the priority assignments in event-triggered clusters a heuristic presented in [3] is used.

[4] describes the *Platune* framework allowing performance and power tuning of a specific parameterized SoC platform. For a given application to be mapped on the target SoC, *Platune* determines all sets of architectural parameter values representing pareto-optimal solutions regarding power and performance. The detection of all pareto-optimal solutions is achieved effectively by clustering the search space into independent parts, for which pareto-optimal solutions can be determined separately.

The *Spacewalker* [10], part of the *PICO* project from HP Labs, pursuits a similar approach. For given applications, it searches for pareto-optimal embedded systems. The search space is explored using a divide-and-conquer approach. In the first step different subsystem are explored independently. From the sets of obtained pareto-optimal subsystems, global systems are constructed and evaluated. This hierarchical exploration approach seems to work well for the architecture presented in the paper. However, for performance dependent subsystems the combination of local pareto-optima rarely leads to global pareto-optima.

The *Sesame* framework, part of the *Artemis* [8] project, is used in [2] to tackle the mapping decision problem of complex applications on heterogeneous embedded systems. The authors use evolutionary techniques to search for solutions which are pareto-optimal regarding maximum processing time, power consumption and total cost. These solutions are then input to a simulation framework for further evaluation. An interesting aspect in this approach is the explicit distinction of pareto-optimal individuals and pareto-optimal individuals fulfilling given constraints. By this means a possible convergence of the evolutionary optimizer towards a set of pareto-optimal infeasible solutions is prevented.

IV. TDMA Time Slot Chromosome

The search space of all time slot assignments for the tasks on a TDMA-scheduled resource is very large, even if we fix the turn-length and the arithmetic precision. Turn-length variation, which is often necessary to find good solutions, adds another search dimension. Since it is unrealistic to try all possible time slot assignments and turn-lengths, a good strategy to walk through the search space is indispensable. In this section we first motivate our choice for the coding of the problem parameters. Then we present the algorithms, i.e. creation of initial population (section IV-B) and variation operators (section IV-C), of the TDMA time slot chromosome used by our exploration framework to walk through the search space of TDMA time slot assignments.

A. Representation of the Chromosome

There are at least two basically different possibilities for coding parameters in continuous optimization problems for the use with an evolutionary algorithm: discretizing the search space into a power of 2 and using a binary string representation, or using a real number representation.

Encoding the problem parameters as real numbers seems to be a suitable approach because of the value dynamic which is appropriate for a continuous optimization problem. We decided to use arithmetic variation operators, as mutation or crossover on the floating point representation has drastic effects even beyond the ”hamming cliff” problem of binary encoding.

B. Initial population

For the creation of the initial population we specify an initial TDMA turn-length $turn_{init}$. Note that choosing a sub-optimal initial turn-length for the initial population does not lead to the incapability of the time slot chromosome to find valid solutions because the variation operators proposed in section IV-C are capable of adapting the turn-length in the course of optimization. Nevertheless, if the designer chooses a good initial turn-length the chromosome converges faster towards the solution space.

Let R be a TDMA scheduled resource subjected to optimization with the tasks $T_0,\ldots,T_{k-1}$ mapped on it. The worst-case execution time, i.e. assuming no interrupts, of T_i is denoted by $WCET_i$, its activating period by $period_i$ and the length of its time slot by $slot_i$. In the following we refer to a specific time slot assignment as *individual* and to the set of individuals used by the evolutionary algorithm as *population*.

In order to create only valid (i.e. resource not overloaded, etc.) individuals for the initial population, we have to ensure that for each task T_i its maximum load $load_{max;i}$ does not exceed its admitted load $load_{adm;i}$.

$$load_{adm;i} \geq load_{max;i} \quad \Leftrightarrow \quad \frac{slot_i}{turn_{init}} \geq \frac{WCET_i}{period_i}$$
$$\Leftrightarrow \quad slot_i \geq \frac{WCET_i}{period_i} * turn_{init}$$

This implies for T_i a minimum time slot

$$slot_{min;i} = \frac{WCET_i}{period_i} * turn_{init}.$$

Algorithm 1 is used to create the initial population which is uniformly distributed in the search space of all valid time slot assignments with a turn-length of $turn_{init}$. To do so, it randomly distributes the initial turn to the tasks $T_0,\ldots,T_{k-1}$. It respects above mentioned minimum time slot length to prevent the creation of non-schedulable individuals.

Algorithm 1 (Create valid initial individual) .

Input: initial turn-length $turn_{init}$
minimum time slots $slot_{min;0},\ldots,slot_{min;k-1}$
Output: valid time slot assignment for $T_0,\ldots,T_{k-1}$

1. $free = turn_{init}$;
2. $set = \{0,1,\ldots,k-1\}$;
3. **while** $(set \neq \emptyset)$ {
4. choose random $r \in set$;
5. $set = set \setminus r$;
6. **if** $(set = \emptyset)$ $slot_r = free$;
7. **else** {
8. $slot_{max} = free - \sum_{x \in set} slot_{min;x}$;
9. $slot_r = random(slot_{min;r}, slot_{max})$;
10. $free = free - slot_r$;
11. }
12. }

C. Variation Operators

The quality of a time slot assignment on a TDMA-scheduled resource is determined by two factors: the admitted loads of the mapped tasks and the turn-length. According to these two problem parameters we introduce in this section two real-coded crossover and two real-coded mutation operators for the time slot optimization on TDMA-scheduled resources. While one crossover and mutation operator, respectively, varies the admitted loads of the mapped task and holds the turn-length constant, the other crossover and mutation operator, respectively, varies the turn-length and leaves the admitted loads untouched.

Thereby, the two crossover operators implement a heuristic strategy to converge towards the solution space, whereas the mutation operators serve to break out of local minima by decreasing or increasing the problem parameters by up to a configurable maximum percentage.

The reason for separating these two problem parameters is the increased control over the optimization process. By configuring the probabilities for the use of the two different operator types for crossover and mutation, the designer can decide which of them is the preferred search parameter. In the extreme case, she can, for example, hold the turn-length constant and optimize only by varying the admitted loads. Note that for the experiments performed in section V, both types of operators are used with the same probability, giving the evolutionary algorithm maximum freedom in walking through the search space.

1) Crossover Operators: Algorithm 2 describes the crossover operator varying the admitted loads while letting the turn-length constant. As input it takes two parent individuals from which it creates two offsprings. Its optimization strategy is related to a binary search method.

The admitted loads of the offsprings are placed evenly (i.e. at $\frac{1}{3}$ and $\frac{2}{3}$) in the respective admitted load interval defined by the two parents. The time slots of offsprings 1 and offspring 2, respectively, are then calculated according to the turn-length given by parent 1 and parent 2, respectively.

Figure 2(a) gives an example for this crossover operator.

Algorithm 2 (Crossover Admitted Load) .

Input: time slots of parent p_1: $[slot_{p_1;0},\ldots,slot_{p_1;k-1}]$
time slots of parent p_2: $[slot_{p_2;0},\ldots,slot_{p_2;k-1}]$
Output: time slots of offspring o_1: $[slot_{o_1;0},\ldots,slot_{o_1;k-1}]$
time slots of offspring o_2: $[slot_{o_2;0},\ldots,slot_{o_2;k-1}]$

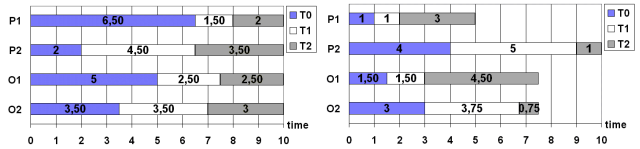
1. $turn_{p_1} = 0$;
2. $turn_{p_2} = 0$;
3. **for** $(i = 0;\ i <= k-1;\ i = i+1)$ {
4. $turn_{p_1} = turn_{p_1} + slot_{p_1;i}$;
5. $turn_{p_2} = turn_{p_2} + slot_{p_2;i}$;
6. }
7. **for** $(i = 0;\ i <= k-1;\ i = i+1)$ {
8. $load_{adm;p_1;i} = slot_{p_1;i}/turn_{p_1}$;
9. $load_{adm;p_2;i} = slot_{p_2;i}/turn_{p_2}$;
10. $difference = \mid load_{adm;p_1;i} - load_{adm;p_2;i} \mid$;
11. **if** $(load_{adm;p_1;i} < load_{adm;p_2;i})$ {
12. $slot_{o_1;i} = (load_{adm;p_1;i} + difference/3) * turn_{p_1}$;
13. $slot_{o_2;i} = (load_{adm;p_2;i} - difference/3) * turn_{p_2}$;
14. }
15. **else** {
16. $slot_{o_1;i} = (load_{adm;p_1;i} - difference/3) * turn_{p_1}$;
17. $slot_{o_2;i} = (load_{adm;p_2;i} + difference/3) * turn_{p_2}$;
18. }
19. }

The crossover operator varying the turn-length is described by algorithm 3. It also pursuits a binary search related strategy. Given the two parent individuals it calculates their average turn-length (lines 1-7). Offspring 1 and offspring 2, respectively, is then created by adapting the time slots of parent 1 and parent 2, respectively, to the average turn-length letting the admitted loads untouched (lines 8-11). Figure 2(b) visualizes the functionality of this crossover operator by means of an example.

Algorithm 3 (Crossover Turn) .

Input: time slots of parent p_1: $[slot_{p_1;0},\ldots,slot_{p_1;k-1}]$
time slots of parent p_2: $[slot_{p_2;0},\ldots,slot_{p_2;k-1}]$
Output: time slots of offspring o_1: $[slot_{o_1;0},\ldots,slot_{o_1;k-1}]$
time slots of offspring o_2: $[slot_{o_2;0},\ldots,slot_{o_2;k-1}]$

1. $turn_{p_1} = 0$;
2. $turn_{p_2} = 0$;
3. **for** $(i = 0;\ i <= k-1;\ i = i+1)$ {
4. $\quad turn_{p_1} = turn_{p_1} + slot_{p_1;i}$;
5. $\quad turn_{p_2} = turn_{p_2} + slot_{p_2;i}$;
6. }
7. $turn_{new} = (turn_{p_1} + turn_{p_2})/2$;
8. **for** $(i = 0;\ i <= k-1;\ i = i+1)$ {
9. $\quad slot_{o_1;i} = slot_{p_1;i}/turn_{p_1} * turn_{new}$;
10. $\quad slot_{o_2;i} = slot_{p_2;i}/turn_{p_2} * turn_{new}$;
11. }



(a) Admitted Load Crossover (b) Turn Crossover

Fig. 2 - CROSSOVER OPERATORS

2) Mutation Operators: The crossover operators described in section IV-C.1 lead to the convergence of the obtained time slot assignments towards (locally) optimal solutions contained "between" individuals considered by the evolutionary algorithm. Of course, it is possible that the variety of the initial population is insufficient to find good solutions only by using these crossover operators. Additionally, the exploration may get stuck in a local optimum, without the possibility to reach globally better solutions.

In this section we introduce two mutation operators, enabling the evolutionary algorithm to break out of local optima and to reach parts of the search space not yet considered.

The mutation operator varying the admitted load while letting the turn-length constant is described by algorithm 4. As input it takes one parent individual from which it creates one offspring. After initialization, $\frac{r}{2}$ pairs of tasks are chosen (line 6-7). For each of these pairs the first task gives a part of its disposable time slot (i.e. the time slot it can dispense without overloading the resource) to the second (lines 8-13). The percentage of the disposable time slot dispensed is randomly chosen in the interval $]0, d_{max}]$, where d_{max} is configurable. Figure 3(a) shows the functionality of this mutation operator by means of an example.

Algorithm 4 (Mutate Admitted Load) .

Input: time slots of parent p: $[slot_{p;0},\ldots,slot_{p;k-1}]$
max. % of disposable time slot dispensed: d_{max}
Output: time slots of offspring o: $[slot_{o;0},\ldots,slot_{o;k-1}]$

1. $turn_p = 0$;
2. **for** $(i = 0;\ i <= k-1;\ i = i+1)$ {
3. $\quad slot_{o;i} = slot_{p;i}$;
4. $\quad turn_p = turn_p + slot_{p;i}$;
5. }
6. choose pair random $r \in [2,\ldots,k]$;
7. choose r distinct integers $q_0,\ldots,q_{r-1} \in [0,k-1]$;
8. **for** $(i = 0;\ i <= r-1;\ i = i+2)$ {
9. $\quad slot_{disposable} = slot_{p;q_i} - load_{max;q_i} * turn_p$;
10. $\quad$ choose random double $d_{applied} \in]0, d_{max}]$;
11. $\quad slot_{o;q_i} = slot_{o;q_i} - d_{applied} * slot_{disposable}$;
12. $\quad slot_{o;q_{i+1}} = slot_{o;q_{i+1}} + d_{applied} * slot_{disposable}$;
13. }

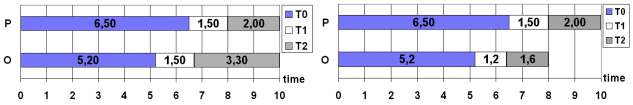
Algorithm 5 describes the mutation operator varying the turn-length. First the target turn-length is chosen, by increasing or decreasing the turn-length of the parent by a percentage randomly chosen in the interval $]0, d_{max}]$, where d_{max} is configurable (lines 1-7). The offspring's time slot assignments are then calculated to sum up in the target turn-length without altering the admitted loads given by the parent's time slot assignment (lines 8-9).

Figure 3(b) shows the functionality of this mutation operator for a turn-length reduction of 20 %.

Algorithm 5 (Mutate Turn) .

Input: time slots of parent p: $[slot_{p;0},\ldots,slot_{p;k-1}]$
max. % by which turn is cut or extended: d_{max}
Output: time slots of offspring o: $[slot_{o;0},\ldots,slot_{o;k-1}]$

1. $turn_p = 0$;
2. **for** $(i = 0;\ i <= k-1;\ i = i+1)$
3. $\quad turn_p = turn_p + slot_{p;i}$;
4. choose random boolean b;
5. choose random double $d_{applied} \in]0, d_{max}]$;
6. **if** $(b = true)$ $turn_{new} = turn_p + d_{applied} * turn_p$;
7. **else** $turn_{new} = turn_p - d_{applied} * turn_p$;
8. **for** $(i = 0;\ i <= k-1;\ i = i+1)$
9. $\quad slot_{o;i} = slot_{p;i}/turn_p * turn_{new}$;



(a) Admitted Load Mutation (b) Turn Mutation

Fig. 3 - MUTATION OPERATORS

V. EXPERIMENTAL RESULTS

In this section we compare the variation operators introduced in section IV-C with binary-coded single-point crossover and binary-coded mutation operator.

The experiments are conducted with our exploration framework [5] using SPEA2 (Strength Pareto Evolutionary Algorithm 2) [12] as selector. The initial population is constructed according to algorithm 1 described in section IV-B with an initial turn-length $turn_{init} = 10$. The operators varying the admitted loads and the operators varying the turn-length are used with the same probability. For the mutation operators we use $d_{max} = 40\%$.

We conduct three different experiments. In the first experiment we compare both approaches regarding the time needed to find a solution for random task sets. In the second and third experiment we investigate their convergence in the search for valid solutions for systems with global constraints.

A. Needed time to find a solution

In the first experiment we compare the binary-coded operators with the proposed real-coded operators regarding the time, expressed by the number of evaluated individuals, they need to find a working time slot assignment. We use random task sets containing 10 tasks with the following attributes:

- Activating period $\mathcal{P}$ between 200 and 600 time units
- Jitter $\mathcal{J}$ between 10 and 400 time units
- Minimum distance between successive events within bursts d between 0 and 100 time units

- Core execution time between 5 and 15 time units
- Constraint $100 + 10*$ core execution time

Figure 4 shows the average results of 100 evaluated random task sets. Compared to the binary-coded operators, the proposed real-coded operators perform approximately $33,5\%$ better in the average case.

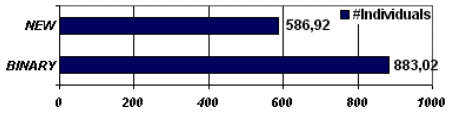


Fig. 4 - AVERAGE NUMBER OF TESTED INDIVIDUALS TO FIND A SOLUTION

B. Convergence towards the solution space

In the second experiment we take into account the 3 task sets described in table I. Each task in each system is constrained by a hard deadline given in table II. We assume that all tasks within a system are mapped on the same TDMA-scheduled resource. Note that the systems are chosen so that it is easiest to find a working time slot assignment for system 1, whereas system 3 is the most difficult to optimize.

We perform an optimization loop of 30 generation each containing 100 individuals. Thereby we use the following optimization objective:

$$\text{minimize} \sum_{i=0}^{9} 1.5^{R_i - D_i},$$

where R_i, resp. D_i, denotes the response time, resp. the deadline, of task T_i.

During optimization we are interested in the best individual after each generation. This gives us an idea of how fast the compared operators converge to working time slot assignments. Figures 5(a), 5(b) and 5(c) give an overview of the average results obtained by 50 optimization runs per system.

We observe that the binary variation operators perform well in the first few generations, whereas they stagnate in later ones, leading to a slow convergence against the solutions space. The new operators converge slower in the beginning but sustain a steady improvement in later generations.

The new operators perform much better for all 3 example systems. Particularly with regard to system 3 we see a big difference in the quality of the obtained results. Where the binary-coded operators do not reach an average fitness value below 10^8, the proposed real-coded operators perform six orders of magnitude better in the average case. We will see in section V-C that the performance difference between the two operator sets grows with increasing difficulty to optimize a given system (which is e.g. caused by narrow deadlines).

C. Needed time to find a solution: narrow deadlines

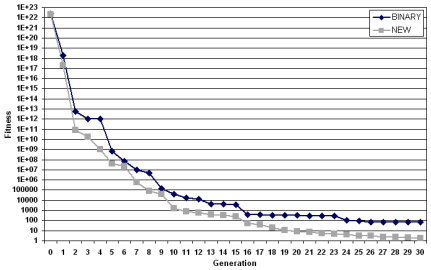
In the last experiment we compare the binary-coded operators with the proposed real-coded operators regarding their performance in optimizing a system with narrow hard deadlines.

	System 1				System 2				System 3			
	CET	$\mathcal{P}$	$\mathcal{J}$	d	CET	$\mathcal{P}$	$\mathcal{J}$	d	CET	$\mathcal{P}$	$\mathcal{J}$	d
T0	12	267	3	-	11	368	51	-	12	198	387	48
T1	11	350	45	-	13	261	198	82	7	102	70	45
T2	8	227	42	-	12	372	155	-	7	283	269	58
T3	12	206	33	-	7	110	197	87	11	354	387	17
T4	12	216	31	-	6	227	73	-	8	239	222	65
T5	9	190	14	-	5	189	47	-	5	194	260	32
T6	10	136	28	-	7	375	182	-	13	148	91	78
T7	13	322	26	-	13	109	20	-	14	114	13	-
T8	11	176	25	-	15	357	45	-	5	313	302	86
T9	9	212	50	-	12	390	50	-	6	119	187	89

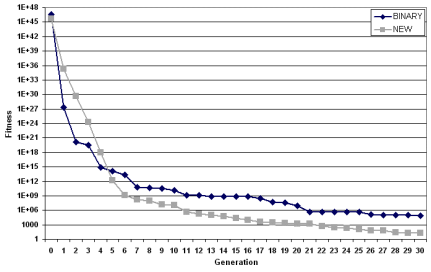
TABLE I - EXAMPLE SYSTEMS WITH 10 TASKS

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9
System 1	180	110	185	100	120	110	100	120	90	115
System 2	100	105	125	78	85	75	105	95	200	120
System 3	110	140	115	145	180	140	200	120	140	100

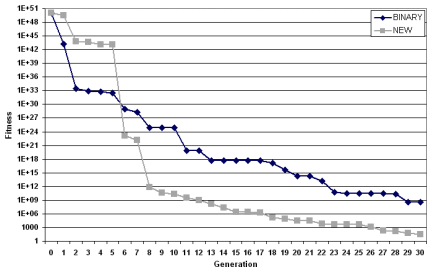
TABLE II - CONSTRAINTS OF EXAMPLE SYSTEMS



(a) System 1 - Low Jitter



(b) System 2 - Medium Jitter



(c) System 3 - High Jitter

Fig. 5 - EVOLUTION OF BEST SOLUTION

Therefore, we consider the task set described in table III. We perform several optimization runs, whereas we tighten the constraints of the tasks in each of them (table IV). We measure the time, expressed as number of evaluated individuals, until the first solution is obtained.

Figure 6 shows the average results obtained by 20 optimization runs per constraint set.

We observe that the binary-coded as well as the proposed real-coded operators have more difficulties in finding working individuals as the deadlines decrease, which is not surprising. However, the binary-coded operators need comparatively more time to find working time slot assignments. Thereby, the performance difference between the binary-coded and the proposed real-coded operators is increasing with decreasing deadlines.

	CET	$\mathcal{P}$	$\mathcal{J}$	d
T0	14	424	257	-
T1	9	287	38	-
T2	10	451	159	-
T3	6	539	11	-
T4	15	309	153	-
T5	9	506	250	-
T6	13	357	393	3
T7	8	304	278	40
T8	5	510	296	-
T9	5	298	184	-
T10	8	243	400	18
T11	9	457	300	-
T12	13	502	312	-
T13	9	247	365	83
T14	15	226	278	85

TABLE III - EXAMPLE SYSTEM WITH 15 TASKS

	con. 1	con. 2	con. 3	con. 4	con. 5	con. 6	con. 7	con. 8
T0	315	303,75	292,5	281,25	270	258,75	247,5	225
T1	210	202,5	195	187,5	180	172,5	165	150
T2	245	236,25	227,5	218,75	210	201,25	192,5	175
T3	196	189	182	175	168	161	154	140
T4	413	398,25	383,5	368,75	354	339,25	324,5	295
T5	245	236,25	227,5	218,75	210	201,25	192,5	175
T6	336	324	312	300	288	276	264	240
T7	378	364,5	351	337,5	324	310,5	297	270
T8	126	121,5	117	112,5	108	103,5	99	90
T9	161	155,25	149,5	143,75	138	132,25	126,5	115
T10	469	452,25	435,5	418,75	402	385,25	368,5	335
T11	574	553,5	533	512,5	492	471,5	451	410
T12	560	540	520	500	480	460	440	400
T13	133	128,25	123,5	118,75	114	109,25	104,5	95
T14	301	290,25	279,5	268,75	258	247,25	236,5	215

TABLE IV - DECREASING DEADLINES

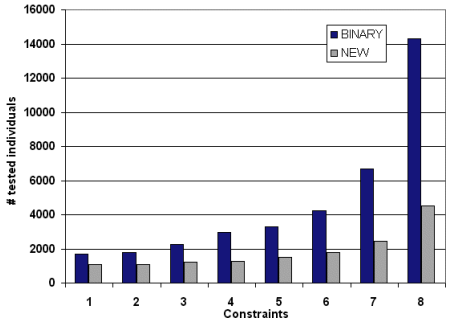


Fig. 6 - AVERAGE NUMBER OF TESTED INDIVIDUALS TO FIND A SOLUTION - DECREASING DEADLINES

VI. CONCLUSION

In this paper we have presented arithmetic real-coded variation operators, which can be used in conjunction with an evolutionary algorithm to optimize time slot assignments and turn-lengths on TDMA-scheduled resources. Our operators implement a heuristic strategy to converge towards the solution space and are able to escape local minima. One special characteristic of our approach is the separated variation of the turn-length and the admitted loads of the mapped tasks, giving the designer increased control over the optimization process.

Extensive experiments using synthetic applications have shown that the proposed variation operators are superior to standard binary-coded operators, in respect of time needed to find valid time slot assignments for given systems. Thereby, the performance difference is particularly noticeable for systems with narrow deadlines.

The approach presented in this paper was recently integrated in our exploration framework for global optimization of heterogeneous MpSoC and distributed systems [5]. In future we will further investigate its performance in combination with other system parameters in the optimization of large systems with complex performance dependencies.

REFERENCES

[1] Stefan Bleuler, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. PISA — a platform and programming language independent interface for search algorithms. *http://www.tik.ee.ethz.ch/pisa/*.

[2] C. Erbas, S.C. Erbas, and A.D. Pimentel. A Multiobjective Optimization Model for Exploring Multiprocessor Mappings of Process Networks. In *Proc. of the International Conference on HW/SW Codesign and System Synthesis (CODES-ISSS '03)*, Newport Beach, USA, October 2003.

[3] J.J.G. Garcia and M.G. Harbour. Optimized priority assignment for tasks and messages in distributed real-time systems. In *Proc. Workshop on Parallel and Distributed Real-Time Systems*, 1995.

[4] T. Givargis and F. Vahid. Platune: A tuning framework for system-on-a-chip platforms. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v21, n11*, 2002.

[5] A. Hamann, M. Jersak, K. Richter, and R. Ernst. Design Space Exploration and System Optimization with SymTA/S - Symbolic Timing Analysis for Systems. In *Proc. 25th International Real-Time Systems Symposium (RTSS'04)*, Lisbon, Portugal, 2004.

[6] Arne Hamann, Rafik Henia, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. SymTA/S - Symbolic Timing Analysis for Systems. *http://www.symta.org/*.

[7] A. Maxiaguine, S. Künzli, S. Chakraborty, and L. Thiele. Rate analysis for streaming applications with on-chip buffer constraints. In *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 131–136, Yokohama, Japan, January 2004.

[8] A.D. Pimentel, P. Lieverse, P. van der Wolf, L.O. Hertzberger, and E.F. Deprettere. Exploring embedded-systems architectures with Artemis. In *IEEE Computer*, November 2001.

[9] Paul Pop, Petru Eles, Zebo Peng, Viacheslav Izosimov, Magnus Hellring, and Olof Bridal. Design optimization of multi-cluster embedded systems for real-time applications. In *Proc. of Design, Automation and Test in Europe (DATE'04)*, Paris, France, 2004.

[10] G. Snider. Automated design space exploration for embedded computer systems. Technical Report HPL-2001-220, Hewlett-Packard Laboratories, 2001.

[11] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *Proc. 39th Design Automation Conference (DAC)*, pages 880–885, New Orleans, USA, 2002. ACM Press.

[12] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.