Multi-Criteria Function Inlining for Hard Real-Time Systems

Kateryna Muts
Hamburg University of Technology
Hamburg, Germany
k.muts@tuhh.de

Heiko Falk
Hamburg University of Technology
Hamburg, Germany
heiko.falk@tuhh.de

ABSTRACT

Modern hard real-time systems shall satisfy some special requirements. Besides timing constraints, the additional design criteria such as code size and energy consumption are also not negligible. To optimize a system towards the mentioned specifications simultaneously is impossible, since the improvement in one of them may lead to the degradation of others. Many compiler-based optimizations techniques have been proposed to optimize an embedded application taking into account only one requirement. Nevertheless, some heuristics consider other requirements as constraints, but not many works have tried to solve a multi-objective problem in this context. We aim to extend a well-known compiler-based optimization, function inlining, to a multi-objective problem. We show that in case of such setup, the considered optimization leads to a set of trade-offs between timing constraints, code size, and energy consumption. Depending on the requirements, a system designer can utilize the output set to make a final decision about the system configuration without building an expensive hardware.

CCS CONCEPTS

• **Computing methodologies** → *Optimization algorithms*; • **Mathematics of computing** → *Evolutionary algorithms*; • **Computer systems organization** → *Real-time systems*; • **Software and its engineering** → *Compilers.*

KEYWORDS

Compiler, Multi-Criteria, Optimization, Evolutionary Algorithm, Real-Time Systems

ACM Reference Format:
Kateryna Muts and Heiko Falk. 2020. Multi-Criteria Function Inlining for Hard Real-Time Systems. In *28th International Conference on Real-Time Networks and Systems (RTNS 2020), June 9–10, 2020, Paris, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394810.3394819

1 INTRODUCTION

A *hard real-time system* is defined as an embedded system that must react within a given deadline and missing the deadline might lead to a catastrophic consequences. One of the most important properties of such systems is the *Worst-Case Execution Time* (WCET), which is the worst possible execution time of a program, independently from its input data [12]. Many approaches have been proposed in the past in order to optimize WCET at compile time [7, 15, 16].

Also, embedded systems have limited memory space, whereas the complexity of the code for embedded applications grows. Therefore, another important criterion in modern embedded systems is *code size*. Different approaches were presented to decrease the code size by compressing the final executable [2, 6, 14, 18].

Apart from that, many embedded systems operate on battery. One way to maximize the time between battery charges is to minimize *energy consumption* of a system. Some optimization techniques were proposed in this context [17, 21, 25].

One way to improve the quality of the embedded applications according to the requirements is to apply optimization techniques at compile time. Originally, the optimizing *WCET-Aware C Compiler Framework* (WCC) [5] was developed to minimize the WCET and to guarantee that the timing constraints of the compiled program are met. Recently, WCC was extended with energy model [19] in order to provide the information about energy consumption of the compiled program. Therefore, we use WCC as a basis for our multi-objective compiler-based optimization.

One of the well-known compiler-based optimizations is so-called *function inlining*. The idea is to replace a function call by the body of the callee. In this case, the function calls and return instruction can be removed from the code which reduces the calling overhead together with improving the pipeline's behavior. The original function inlining focuses on minimizing the *Average-Case Execution Time* (ACET) of a program and, in general, increases the WCET. Nevertheless, it has been shown that function inlining can also be utilized to minimize the WCET of the final program [10]. However, the main drawback of this transformation is the growing code size of the program due to duplicates of the function body. Moreover, considering WCET and code size as two objectives performing function inlining leads to a bi-objective optimization problem [13], since the objectives contradict each other and cannot be minimized simultaneously. Besides that, to the best of our knowledge, energy consumption has never been considered within the context of function inlining.

The main contributions of this paper are:

- We formulate function inlining as a multi-objective optimization problem considering WCET, code size, and energy consumption as objectives;
- After solving the problem, the compiler returns a set of the best solutions corresponding to the most optimal trade-offs between the objectives;
- We analyze two evolutionary algorithms in terms of capability to solve the multi-objective function inlining;
- We compare the result of the standard function inlining to the multi-objective solution set.

RTNS 2020, June 9–10, 2020, Paris, France

ACM ISBN 978-1-4503-7593-1/20/06…$15.00
https://doi.org/10.1145/3394810.3394819

This paper is organized as follows: Section 2 gives a brief overview of related work with regard to function inlining and multi-objective compiler-based optimizations. Section 3 briefly introduces the WCC compiler framework, which we utilize as a basis for the proposed multi-objective function inlining. Section 4 explains the multi-objective function inlining in detail. In Section 5, evaluation results are presented. This paper closes with a conclusion.

2 RELATED WORK

Many approaches have been proposed in the past in order to optimize WCET at compile time. Oehlert *et al.* [15] proposed an optimization to reduce final WCET performing bus-aware static instruction scratchedpad memory allocation. The achieved WCET reduction is 26 % in average.

Originally, function inlining is a compiler-based optimization that aims to minimize the ACET of a program. However, it has great potential to be combined with other optimization techniques.

Woerteler *et al.* [24] applied function inlining to XQuery, which was developed as a query language for XML database and later became a complete functional programming language. In their paper, the authors combined function inlining and query optimizer techniques to achieve a better performance.

Moreover, function inlining can also be applied to improve the worst-case performance. Lokuciejewski *et al.* [10] presented WCET-aware function inlining approach based on machine learning heuristics. The authors showed that the WCET-driven inlining heuristics based on *random forests* outperform standard heuristics in terms of WCET. In the paper, the code size increase was also taken into account choosing functions for inlining. However, in contrast to our approach, the influence on the energy consumption was not considered. Moreover, the proposed method returns just one solution for the inlining problem, whereas we aim to get the set of the best trade-offs between WCET, code size, and energy consumption.

Multi-objective mathematical approaches are rarely used to perform compiler-based optimizations in order to find the set of trade-offs between the objectives. Lokuciejewski *et al.* [11] considered a problem of finding the optimal compiler optimization sequences. The authors examined two pairs of objectives (WCET, ACET) and (WCET, code size) separately making the problem bi-objective. They exploited the evolutionary algorithm to identify the set of optimal compiler optimization sequences for each pair of objectives.

3 WCET-AWARE C COMPILER FRAMEWORK

We use the WCET-aware C compiler framework WCC [5] as a basis for our approach. Figure 1 presents the structure of the WCC. The main parts of it are a parser, the high-level representation, a code selector, the low-level representation, and a code generator. The integration of a static WCET analyzer within WCC, the tool aiT [1], provides the information about WCET at compile time and enables WCET-aware compiler-based optimizations.

A user can annotate the input ANSI-C source code of WCC with flow facts that provide information about the code structure. The flow facts such as the number of loop iterations or depth of the recursion are mandatory for a static WCET analysis. We consider this data as given, since it is out of scope of this work.

The WCC parser creates the high-level intermediate representation ICD-C from the ANSI-C source files. The high-level representation is machine independent and moreover, it is close to the source C language. Function inlining considered in the paper is a high-level optimization technique, i.e., it does not require the knowledge of the target architecture.

A code selector generates the low-level intermediate representation ICD-LLIR from the high-level representation. At this level, the memory hierarchy and machine instructions of the target architecture are modeled. Finally, an assembler and a linker are involved to produce the final executable from the low-level representation.

To perform a static WCET analysis as well as to get energy consumption and code size of a program, characteristics of the target architecture's instructions are required. Hence, WCC is tightly coupled to the static WCET analyzer aiT [1] at low-level intermediate representation. Moreover, at this level, the energy model that enables energy-aware optimizations is integrated into WCC.

aiT performs the static WCET analysis using generic data flow, loop and path analyses together with processor-specific microarchitectural analyses like, e.g., pipeline analysis. For energy analysis, we use offline the measurement setup from [19], which is based on [20], to determine the base- and inter-instruction-costs per machine instruction. The compiler exploits the measurement data during compile-time such that per basic block, the instruction-level energy costs are accumulated. Next, the costs per basic block are scaled up by the number of executions per basic block that are determined using profiling. This way, energy costs get analyzed and estimated bottom-up from instructions over basic blocks and functions up to the whole program.

In order to utilize the WCET, energy consumption, and code size data performing the high-level optimizations, e.g., function inlining, WCC features Back-Annotation, which connects two levels of abstraction. Back-annotation translates the timing and energy data as well as code size of the program from the low-level representation into the high-level representation.

4 MULTI-OBJECTIVE FUNCTION INLINING

In this section, we describe the proposed multi-criteria function inlining approach. Section 4.1 shortly introduces the main definitions of multi-objective optimization that are required for the approach described in the paper. In Section 4.2, we formulate function inlining as a multi-objective optimization problem and discuss the methods that we utilize to solve it.

4.1 Multi-Objective Optimization Problem

In real-world problems, usually, several criteria have to be taken into account. For example, as described in Section 1, the worst possible execution time, code size, and energy consumption of a program play an important role for hard real-time systems. Furthermore, optimizing all objectives simultaneously is usually impossible due to the contradictions between the considered objectives. Such problems are called *Multi-Objective Optimization Problems* (MOPs). In contrast to Single-Objective Optimization Problems, MOPs are more complex and difficult in terms of finding the optimal solution. Besides, the result of solving a MOP is not a single solution but a set of solutions that represent the trade-off between the objectives.

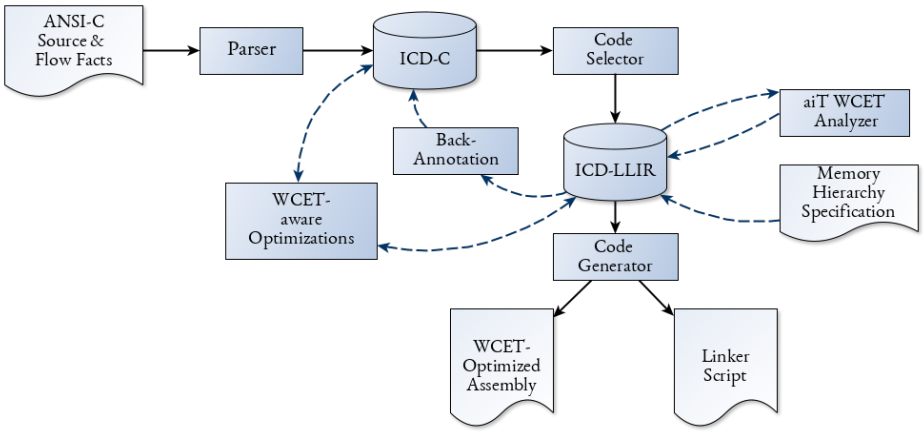


Figure 1: Structure of the WCET-aware C compiler framework WCC [5]

MOPs without constraints can be formulated as a minimization problem without loss of generality:

$$\min_{x \in X} F(x) = (f_1(x), f_2(x), \ldots, f_s(x)) \tag{1}$$

where $x = (x_1, x_2, \ldots, x_n) \in X$ is a vector of decision variables, $f_t(x)$, $t = 1, 2, \ldots, s$ is an objective function.

Next, we introduce some definitions that are necessary for the further explanations.

Definition 4.1. The set X of all possible decision variables is called a **decision** or **search space** of the problem.

Hence, the search space describes the unknowns of the problem.

Definition 4.2. The **objective space** Z of the problem is

$$Z = \{F(x) | x \in X\}. \tag{2}$$

The objective space of the problem consists of all possible vectors $(f_1, f_2, \ldots, f_s)$ of objective values on the given search space.

Let us consider two vectors of decision variables x and y to introduce the definitions of multi-objective optimization. In the following definitions, we consider a minimization problem.

Definition 4.3. x **dominates** y ($x \prec y$), if

$$\forall t \in \{1, 2, \ldots, s\} \quad f_t(x) \leq f_t(y) \tag{3}$$

and

$$\exists r \in \{1, 2, \ldots, s\} : \quad f_r(x) < f_r(y). \tag{4}$$

In words, the vector x dominates another vector y, if for all objectives, the values at x is not worse than the values at y and there is at least one objective whose value at x is better than the value at y. The next definition, weak dominance, describes the case if the values at x are not worse than the values at y for all objectives.

Definition 4.4. x **weakly dominates** y ($x \preceq y$), if

$$\forall t \in \{1, 2, \ldots, s\} \quad f_t(x) \leq f_t(y). \tag{5}$$

Definition 4.5. The dominance relation $x \prec y$ is called **Pareto dominance** and $x \preceq y$ is **weak Pareto dominance**.

Definition 4.6. A solution that is not dominated by any other solution is called **Pareto optimal**.

Definition 4.7. The **Pareto optimal set** P^* is a subset of the search space consisting of all Pareto optimal solutions:

$$P^* = \{x | x \text{ is Pareto optimal}\}. \tag{6}$$

The goal of any multi-objective optimization is to find the Pareto optimal front:

Definition 4.8. The **Pareto optimal front** PF^* is defined as a subset of the objective space as follows:

$$PF^* = \{F(x) | x \in P^*\}. \tag{7}$$

In general, the search space of MOPs is often too large, therefore, a problem of determining a single Pareto optimal solution might be even *NP hard* [8]. Moreover, for many real-life problems, a proof of optimality is computationally infeasible. For this reason, the aim is to find a *Pareto front approximation*.

Well-known methods to approximate Pareto fronts are evolutionary multi-objective algorithms. The main advantage of them is that they are population-based heuristics that allow to consider several solution candidates within one iteration. A general framework for any evolutionary algorithm is presented in Algorithm 1.

Algorithm 1 Evolutionary algorithm.

```
1: Input: initialized initial population, stopping criterion;
2: Output: approximated Pareto front.
3: while Stopping criterion is not reached do
4:     Crossover;
5:     Mutation;
6:     Selection
7: end while
```

Every evolutionary algorithm starts with some initial random *population*. Elements of a population are called *individuals* and represent variables from the search space. At every iteration, crossover, mutation, and selection operators try to improve the population. *Crossover* is a genetic operator that creates a new individual called

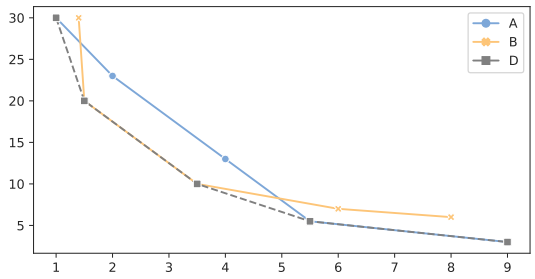


Figure 2: Example of two sets A and B with the set D of all non-dominated points of the union of sets ($A \cup B$).

child combining some individuals from the current population, which are also called *parents*. Another genetic operator is *mutation* that produces a new individual by changing some characteristics of a known one. As a final step at each iteration, a *selection* operator chooses the best individuals for the next generation. An appropriate stopping criterion for such algorithms is usually a predefined number of function evaluations or a maximum generation number.

To evaluate the performance of a multi-objective algorithm, quality indicators are often used. Since the true Pareto front is unknown in the function inlining problem described in Section 4.2, we define a new set D that represents the best available approximation of the true Pareto front. Assuming that A and B are two Pareto fronts returned by different algorithms, we define the set D as a set of all non-dominated points of the union of sets A and B, $A \cup B$, as shown in Figure 2.

Since a Pareto front returned by an evolutionary algorithm might have some duplicates of a solution, we introduce the operator $Unique(\cdot)$ that returns the number of unique elements in the Pareto front. Meanwhile, the operator $|\cdot|$ represents the total size of a set.

Non-dominance ratio is a quality indicator that represents the ratio of non-dominated solutions in each set with respect to the set D. For the set A it is defined as follows:

$$NR_A = \frac{Unique(A \cap D)}{|D|}. \tag{8}$$

Another quality indicator that describes a total number of dominated points in a set is called *coverage*:

$$C_A = 1 - \frac{|A \cap D|}{|A|}. \tag{9}$$

The last quality indicator that we consider is *internal non-dominance ratio*. It is defined similar to the non-dominance ratio:

$$INR_A = \frac{Unique(A \cap D)}{|A|}. \tag{10}$$

However, INR_A describes the ratio of the unique non-dominated points with respect to the set A itself, whereas NR_A represents the ratio with respect to the set D.

The values of the considered quality indicators are always in the interval $[0, 1]$. Besides, NR and INR are operators which to be maximized, whereas C has to be minimized.

4.2 Function Inlining

Function inlining is a well-known compiler-based optimization that can be used to minimize the WCET of a program. The reduction of WCET is achieved by replacing a function call by the body of the callee. Arguments corresponding to function parameters are stored in variables. Additionally, the function call and return instructions together with parameter handling are removed from the code that yields the reduction of calling overhead and a smoother pipeline behavior. However, duplicates of the function body lead to an increasing code size. Furthermore, due to the insertion of additional variables, more registers are required and therefore, the register pressure increases. As a result, the performance of the program as well as energy consumption may be degraded.

For example, let us apply function specialization to optimize the benchmark *codecs_dcodhuff* [4]. If we consider only the code size as an objective to be minimized, then, obviously, the solution is the original program without any modifications, since function specialization always increases the code size. Minimizing only with respect to WCET leads to 9 % decrease in WCET, but code size and energy consumption increase by 51 % and 25 %, respectively. Finally, optimizing only energy consumption increases WCET by 10 % and code size by 61 %, however, energy consumption decreases only by 5 %. As the example suggests, optimizing only one objective is not sufficient to get a good final result. For this reason, we formulate function specialization as a multi-objective problem aiming to find the best trade-offs between the objectives.

In our approach, binary N-dimensional vectors x represents the search space X of the problem with N corresponding to the number of function calls in the code:

$$X = \left\{ \begin{array}{c} x = (x_1, x_2, \ldots, x_N), \\ x_i \in \{0, 1\} \quad \forall i = 1, 2, \ldots, N. \end{array} \right\} \tag{11}$$

Every x_i stands for a function call (of any function) in the input source code. If x_i is equal to 1, then the function is to be inlinined at this place of the code, and x_i equals to 0, otherwise.

We make some natural **assumptions** collecting candidates for inlining:

- the function is not recursive;
- the argument list of the function does not contain a variable number of arguments;
- the function does not declare any static symbols.

Furthermore, as already mentioned, we consider a 3-dimensional objective function U:

$$U = (\text{WCET},\ \text{Code Size},\ \text{Energy Consumption}) \tag{12}$$

Then, we formulate the optimization problem as follows:

$$\min_{x \in X} U(x). \tag{13}$$

In our approach, we do not consider any other constraints for the objectives, because we aim to find the full set of possible trade-offs between the objectives.

Durillo *et al.* [3] showed that the most promising evolutionary algorithm, which outperforms widely used *SPEA*2 and $NSGA-II$ algorithms, is the third version of Generalized Differential Evolution (GDE3) [9]. Moreover, according to the current state of the art, GDE3 is *the best* choice in multi-criteria optimization. Therefore, we utilize *GDE*3 as a basis for our approach. However, the standard GDE3

handles real-valued variables, i.e., the search space of the problem is a subspace of a real-valued vector space. Hence, we cannot apply GDE3 directly to binary optimization problems. Meanwhile, as described above (cf. (11)), we formulate function inlining as a binary-encoded problem. For this reason, we make some modifications of the GDE3 algorithm. The difficulties to apply GDE3 to a binary-encoded problem occur at the mutation stage of the algorithms, therefore, we describe this part of GDE3 in details.

In the context of any evolutionary algorithm (cf. Algorithm 1), mutation describes the process of creating a new individual from some selected individuals from the current population. At this stage of the algorithm, it might happen that a new individual is not in the search space of the problem. GDE3 suffers from this problem in our case, since the mutation operator selects three random, except mutually different, individuals x^1, x^2, x^3 from the current population and creates a new vector x^{mut} as follows:

$$x^{mut} = x^3 + F \cdot (x^1 - x^2), \tag{14}$$

where $F \in \mathbb{R}_{\geq 0}$ is a *scaling factor* of the algorithm. The term $F \cdot (x^1 - x^2)$ defines the magnitude and direction of the mutation [9].

If x^1, x^2, and x^3 in (14) are binary vectors, then x^{mut} is not necessary a binary vector any more, since F is a real scalar parameter. To tackle this issue, we utilize a *probability estimation operator* P. It is a sigmoid function which is widely used to map a real variable into the interval $[0, 1]$. The probability estimation operator was originally proposed by Wang *et al.* [22] for a single-objective optimization problem. We integrate this operator into the multi-objective GDE3 algorithm. For the vector $x^{mut} = (x_1^{mut}, x_2^{mut}, \ldots, x_N^{mut})$ with $x_j^{mut} \in \mathbb{R}$, operator P is defined as follows:

$$P(x_j^{mut}) = \frac{1}{1 + e^{(2b \times (x_j^{mut} - 0.5))/(1+2F)}}, \tag{15}$$

where $b \in \mathbb{R}_{\geq 0}$ is a *bandwidth* parameter, which describes the smoothness of the operator P.

Then, a new binary vector $bx = (bx_1, bx_2, \ldots, bx_N)$ is generated as in (16),

$$bx_j = \begin{cases} 1, \text{ if } rand \leq P(x_j^{mut}), \\ 0, \text{ otherwise,} \end{cases} \tag{16}$$

where $rand \in [0, 1)$ is a random number.

All other parts of GDE3 algorithm are preserved and described further. First, we introduce the following notation:

- X is a search space;
- NP is the maximum number of individuals in a population;
- Individuals are N-dimensional vectors;
- $G \subset X$ is the current population.

Then at every new iteration (creating a new population), a new individual $\hat{x}^i$ with $i = 1, 2, \ldots, NP$ is created as follows [9]:

(1) Select randomly three individuals $\bar{x}^1, \bar{x}^2$, and $\bar{x}^3$ from the current population G that are mutually different and different from $x^i \in G$;
(2) Apply mutation operator to the individuals $\bar{x}^1, \bar{x}^2$, and $\bar{x}^3$ as described in (14)–(16) to get a new vector $bx^i \in \mathbb{R}^N$;
(3) Randomly select $j_{rand} \in \{1, 2, \ldots, N\}$;
(4) Derive a trial vector $u^i = (u_1^i, u_2^i, \ldots, u_N^i)$ as in (17)

$$\forall j \leq N \quad u_j^i = \begin{cases} bx_j^i, \text{ if } rand_j \leq CR \vee j = j_{rand}, \\ x_j^i, \text{ otherwise,} \end{cases} \tag{17}$$

where $rand_j \in [0, 1)$ is a random number and $CR \in \mathbb{R}_{\geq 0}$ is a *crossover* parameter, which controls the rotational invariance of the search [9].
(5) Select the individual $\hat{x}^i$ for the next population:

$$\hat{x}^i = \begin{cases} u^i, \text{ if } u^i \preceq x^i, \\ x^i, \text{ otherwise.} \end{cases} \tag{18}$$

Scaling factor F, bandwidth parameter b, and crossover parameter CR from (14), (15), and (17), respectively, are control parameters defined by a user. They are fixed for all iterations of the algorithm.

Moreover, to evaluate the results returned by the proposed modified GDE3 algorithm, we compare them to another evolutionary algorithm, MBPOA [23], since MBPOA was designed for binary-encoded problems and also has GDE3 algorithm as a basis. We apply MBPOA to the considered function inlining problem without any modifications.

5 EVALUATION

We perform all experiments on an Intel Xeon Server using the WCET-aware C compiler framework WCC for the ARM7TDMI micro-controller in the thumb mode. Moreover, the optimization considered in the paper, function inlining, shows the most significant improvement of the final executable in combination with other optimizations that were impossible due to function boundaries, e.g., constant propagation and dead code elimination. Therefore, the optimization level O2 was considered performing all evaluations. Finally, since the ARM7TDMI micro-controller implies no hardware floating point unit, the software libraries were used to tackle this issue. Hence, we considered functions of the floating point libraries also as candidates for the inlining.

We use for evaluation the benchmarks from the public available test suites PolyBench, MediaBench, MRTC, DSPstone, and UTDSP with annotated loop bounds from the TACLeBench project [4].

In addition to the assumptions in Section 4.2, during the evaluations the code size limit of a function to be inlined is set to 5000 bytes. The last assumption prevents the inlining of too "heavy" functions and decrease the search space of the problem.

As described in Section 4.2, we utilize the modified GDE3 and MBPOA algorithms to solve the multi-objective function inlining problem (13). For both evolutionary algorithms, modified GDE3 and MBPOA, and for every benchmark, the population size NP is set to 20 and the number of algorithm's iterations NI is defined by

$$NI = min\left(10, max\left(5, \frac{2^N}{NP}\right)\right), \tag{19}$$

where N is the dimension of the search space. Table 1 presents the dimension of the search space for each benchmark.

In addition, MBPOA algorithm has one control parameter, namely *crossover*, whereas the modified GDE3 has three parameters: *scaling factor* F, *bandwidth* b, and *crossover* CR (cf. (14), (15), and (17)). Since the value of each control parameter is crucial for the performance of the algorithm, we provide a careful sensitivity analysis to identify

Table 1: Evaluation results. N is the dimension of the search space or, in other words, the number of function calls. The runtime corresponds to a single run of the evolutionary algorithm, whereas the best parameter values are the control parameter values for which the evolutionary algorithm shows the best performance in terms of quality indicators. Algorithms' control parameters: CR is crossover parameter, b is bandwidth, and F is scaling factor. Quality indicators: C is coverage (9), NR is non-dominance ratio (8), and INR is internal non-dominance ratio (10).

Benchmark	N	Runtime (sec)		Best Parameters Values		Quality Indicators					
		GDE3	MBPOA	GDE3	MBPOA	C		NR		INR	
				(CR, b, F)	(CR)	GDE3	MBPOA	GDE3	MBPOA	GDE3	MBPOA
adpcm_g721_board_test	23	177	179	(0.1, 5.0, 2.0)	0.1	0.7500	0.9500	0.0588	0.0102	0.3000	0.0556
adpcm_g721_verify	25	200	186	(0.1, 500.0, 10.0)	0.1	0.5500	0.7000	0.0756	0.0435	0.5294	0.2941
atax	126	235	252	(0.1, 1000.0, 1.0)	0.1	0.3000	0.5500	0.2373	0.1667	0.7000	0.4500
cholesky	128	271	277	(0.2, 500.0, 0.2)	0.5	0.4000	0.7000	0.1429	0.0811	0.6316	0.3000
cjpeg_jpeg6b_wrbmp	16	115	118	(0.5, 500.0, 2.0)	0.8	0.5000	0.6000	0.0923	0.0783	0.6000	0.4500
cnt	128	208	230	(0.3, 5.0, 0.2)	0.2	0.8000	0.7778	0.4000	0.2500	0.4000	0.2222
codecs_codrle1	25	47	46	(0.5, 100.0, 2.0)	0.6	0.6471	0.7500	0.0569	0.0394	0.4375	0.2941
codecs_dcodhuff	18	53	57	(0.6, 1000.0, 0.2)	0.4	0.2500	0.5000	0.0631	0.0476	0.7368	0.5500
correlation	127	245	274	(0.3, 500.0, 0.2)	0.3	0.7857	1.0000	0.1200	0.0000	0.2143	0.0000
expint	125	209	223	(0.6, 50.0, 0.2)	0.6	0.0000	0.5000	0.5000	0.3333	1.0000	0.5000
fdtd-2d	126	233	253	(0.1, 20.0, 5.0)	0.1	0.2500	0.5000	0.2083	0.1493	0.7500	0.5000
fft1	131	318	323	(0.8, 20.0, 0.2)	0.6	0.5500	0.7500	0.0982	0.0549	0.5500	0.2500
floyd-warshall	126	229	249	(0.1, 20.0, 10.0)	0.1	0.3684	0.5789	0.2400	0.1739	0.6316	0.4211
gemm	125	234	251	(0.3, 1000.0, 1.0)	0.3	0.6667	1.0000	0.0870	0.0000	0.3333	0.0000
gemver	125	258	257	(0.1, 100.0, 10.0)	0.5	0.1429	0.4737	0.3429	0.2941	0.8571	0.5263
gsm_decode	23	470	417	(0.7, 100.0, 0.5)	0.5	0.5000	0.7500	0.0820	0.0362	0.5000	0.2500
huffc	86	173	181	(0.8, 100.0, 0.5)	0.1	0.6875	0.9444	0.0610	0.0217	0.3125	0.1111
jacobi-1d-imper	126	229	241	(0.1, 0.0, 10.0)	0.1	0.4000	0.6500	0.1412	0.0875	0.6000	0.3500
ludcmp	126	251	265	(0.6, 100.0, 1.0)	0.6	0.2500	0.3077	0.4615	0.3913	0.7500	0.6923
md5	13	3	3	(0.5, 20.0, 1.0)	0.3	0.0000	0.0000	0.8500	0.9000	1.0000	1.0000
mvt	126	205	231	(0.2, 100.0, 5.0)	0.2	0.7895	1.0000	0.0976	0.0000	0.2105	0.0000
qurt	125	253	269	(0.5, 50.0, 0.2)	0.7	0.7500	0.7500	0.1579	0.0625	0.3750	0.2500
sqrt	124	245	252	(0.6, 100.0, 0.2)	0.4	0.5556	0.5000	0.2174	0.1304	0.5556	0.5000
st	129	272	283	(0.1, 500.0, 1.0)	0.1	0.6154	0.8000	0.2222	0.1538	0.4615	0.2000
trisolv	126	248	248	(0.6, 500.0, 0.2)	0.6	0.4375	0.5000	0.1875	0.1739	0.5625	0.5714
trmm	125	230	240	(0.3, 1000.0, 0.2)	0.3	0.6667	1.0000	0.0952	0.0526	0.3333	0.1667

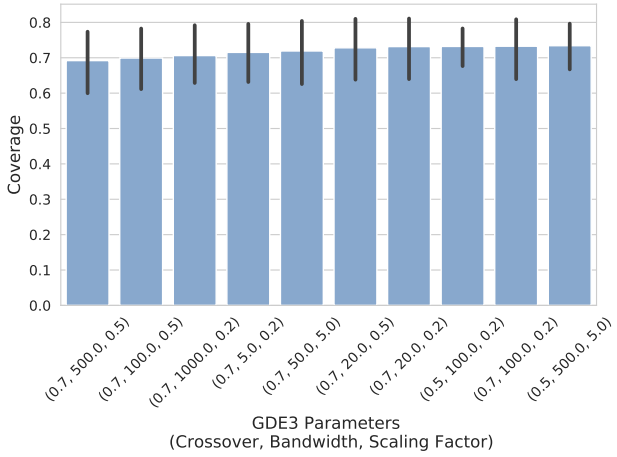
the actual parameters' values for each algorithm. We consider the following possible values of parameters:

- Crossover $CR \in \{0.1, 0.2, 0.3, \ldots, 0.8\}$;
- Bandwidth $b \in \{0, 5, 20, 50, 100, 500, 1000\}$;
- Scaling Factor $F \in \{0.2, 0.5, 1.0, 2.0, 5.0, 10.0\}$.

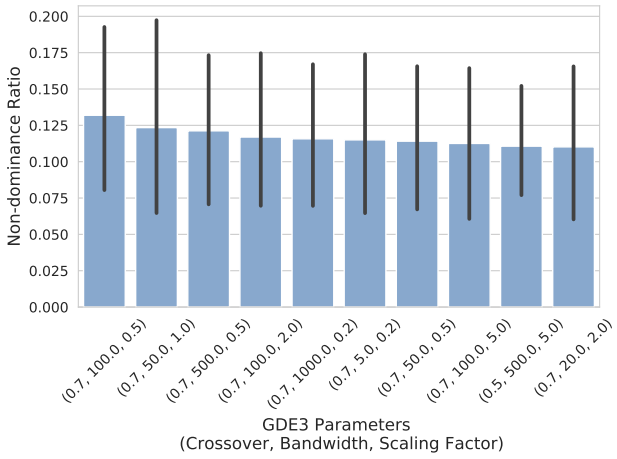
The initial population of any evolutionary algorithm is defined randomly and the results of the algorithm usually depend on the initial population. For this reason, we repeated the experiments 5 times for different initial populations. Evaluation results consist of quality indicator values considering all possible combinations of control parameters in case of GDE3 algorithm or all possible values of the crossover parameter in case of MBPOA algorithm. It means that in total for 5 repetitions of 26 benchmarks, 44720 evaluations were done, because of considering 336 and 8 possible configurations of GDE3 and MBPOA algorithm, respectively.

Moreover, Table 1 shows the runtime of executing GDE3 or MBPOA algorithm once. The results for both algorithms are almost the same, because most of the runtime is taken to evaluate the objectives by the static WCET analyzer. However, due to a quite large runtime for almost all benchmarks, the extensive evaluations of the algorithms are problematic. It should be mentioned, we had to exclude a lot of benchmarks, which are not listed in the table, from the evaluations due to an extremely large runtime. For the same reason, we set the population size just to 20 individuals as described above, despite the fact that for the benchmarks with a large search space, it is not enough in order to find the Pareto front.

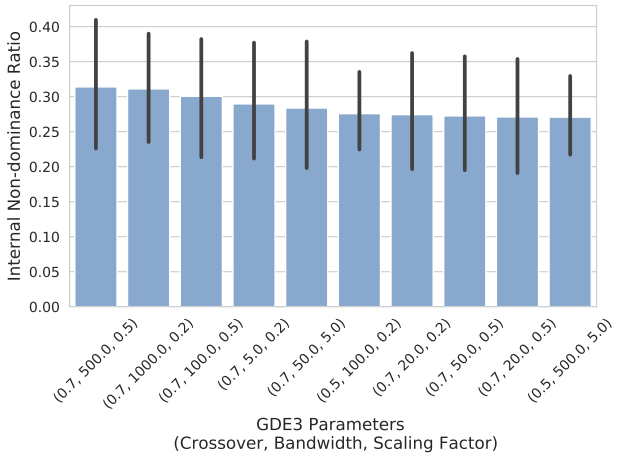
Choosing the best values for the GDE3 control parameters is very difficult, since all possible combinations of control parameters have to be examined. Figure 3 presents the results for 10 combinations of the algorithm's parameters which result to the best values of the quality indicators for all benchmarks in average. The best results in terms of coverage and internal non-dominance ratio, GDE3 achieves with crossover $CR = 0.7$, bandwidth $b = 500$, and scaling factor $F = 0.5$ as shown in Figures 3a and 3c. However, the largest value for the non-dominance ratio is returned when bandwidth parameter is set to 100 according to Figure 3b.



(a) Coverage.



(b) Non-dominance ratio.



(c) Internal non-dominance ratio.

Figure 3: Quality indicators for GDE3 control parameters. The results show the average over all considered benchmarks evaluated 5 times for different initial populations.

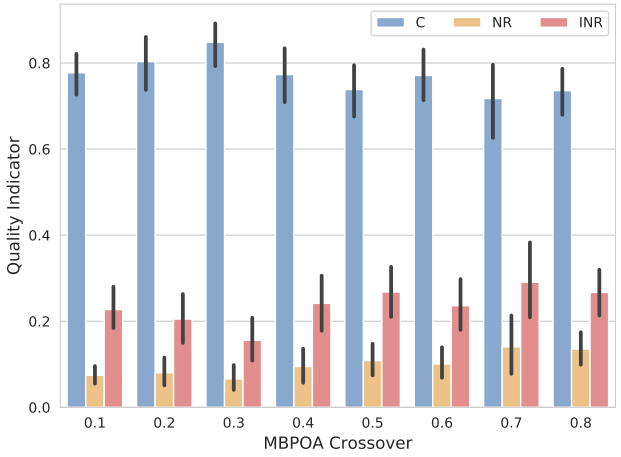


Figure 4: Quality indicators for different values of the crossover parameter in MBPOA algorithm. C is coverage (9), NR is non-dominance ratio (8), and INR is the internal non-dominance ratio (10). The results show the average over all considered benchmarks evaluated 5 times for different initial populations.

Figure 4 shows the average values of quality indicators for all benchmarks considering different values of the parameter crossover during evaluation of MBPOA algorithm. We consider the quality indicators, coverage C, non-dominance ratio NR, and internal non-dominance ratio INR described in Section 4.1. The results suggest that the best crossover value for the MBPOA algorithm is 0.7, since in this case, the quality indicators NR and INR are maximal and C gets the smallest value.

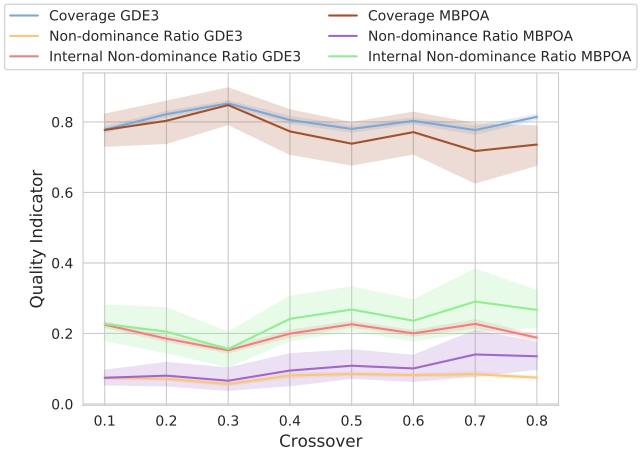
To find the final values of control parameters for both algorithms, we additionally analyze the general dependency of the quality indicators from the control parameters. Since both algorithms, GDE3 and MBPOA, depend on the control parameter crossover, Figure 5a presents an estimate of the central tendency and a confidence interval for the considered quality indicators as functions of crossover. The central tendency of all quality indicators in the case of MBPOA is better than using GDE3 algorithm. Moreover, the figure also suggests to choose the crossover parameter equal to 0.7 for both algorithms, that is consistent with the results in Figures 3 and 4.

Additionally, GDE3 algorithm also depends on the control parameters bandwidth and scaling factor. Figures 5b and 5c show results for parameters bandwidth and scaling factor, respectively.

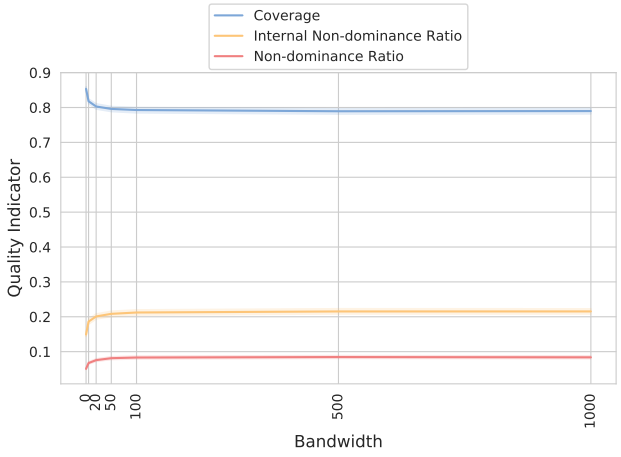
Figure 5b shows that the values of the quality indicators for bandwidth greater than 100 remains almost unchanged, therefore, Figure 3b suggests the value 100 for bandwidth parameter, whereas in Figures 3a and 3c, the algorithm achieves the best behavior with bandwidth equal to 500.

Regarding the results in Figure 5c, the minimum of coverage and the maximums of non-dominance ratio as well as internal non-dominance ratio are at scaling factor equal to 0.5 that explains the best value for the scaling factor in Figure 3.

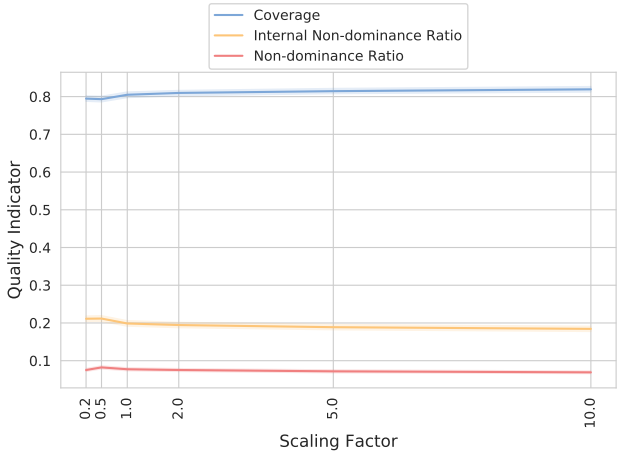
Based on the analyses above, we fix the control parameters of both algorithms to the best found values, namely:



(a) Quality indicators as functions of crossover.



(b) Quality indicators as functions of GDE3's bandwidth.



(c) Quality indicators as functions of GDE3's scaling factor.

Figure 5: The behavior of quality indicators. The results show the average over all considered benchmarks evaluated 5 times for different initial populations.

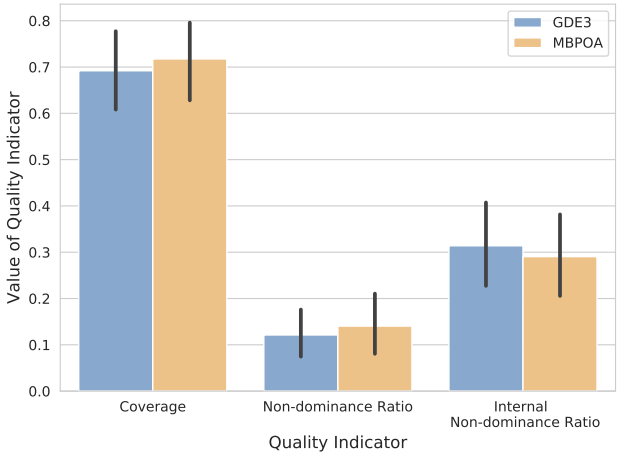


Figure 6: Comparison of quality indicators for GDE3 and MBPOA. The results show the average over all considered benchmarks evaluated 5 times for different initial populations.

- GDE3: $CR = 0.7$, $b = 500$, $F = 0.5$;
- MBPOA: $CR = 0.7$.

We decide to set the bandwidth of GDE3 algorithm equal to 500 due to results in Figures 3a and 3c, and since, as we have shown in Figure 5b, changing the bandwidth values between 100 and 1000 has little impact on quality indicators.

In Figure 6, we present the comparison of quality indicators for the fixed control parameters. It shows that the results for both algorithms are almost the same. Nevertheless, GDE3 outperforms MBPOA a little bit in terms of coverage and internal non-dominance ratio, i.e., that performing GDE3 the ratio of non-dominated solutions on the returned Pareto front is larger than in case of MBPOA algorithm. However, MBPOA shows a better result in terms of non-dominance ratio, i.e., MBPOA returns in general more non-dominated solutions than GDE3.

In Figure 7, we show the results for some benchmarks fixing the values of parameters and repeating experiments 5 times. To present the results, we chose the benchmarks with different sizes of the search space. Some of the selected benchmarks, namely *gsm_decode, cjpeg_jpeg6b_wrbmp, codecs_codrle1, codecs_dcodhuff*, and *md5* have the dimensions of the search spaces less than 25 (cf. Table 1), whereas *huffc, qurt, st*, and *trisolv* have much larger search spaces with the dimensions 86, 125, 129, and 126, respectively. MBPOA algorithm returns a lit bit better Pareto front with respect to the non-dominance ratio for the smallest benchmark *md5* with the search space of the size 13, whereas GDE3 shows better coverage and internal non-dominance ratio in this case. However, for the benchmark *qurt, st*, and *trisolv*, which have the largest search spaces with the dimensions 125, 129, and 126, respectively, GDE3 outperforms MBPOA. Another fact, which is worth to mention, is related to the comparison of the benchmarks *gsm_decode* and *trisolv* with search space dimensions 23 and 126, respectively. Nevertheless

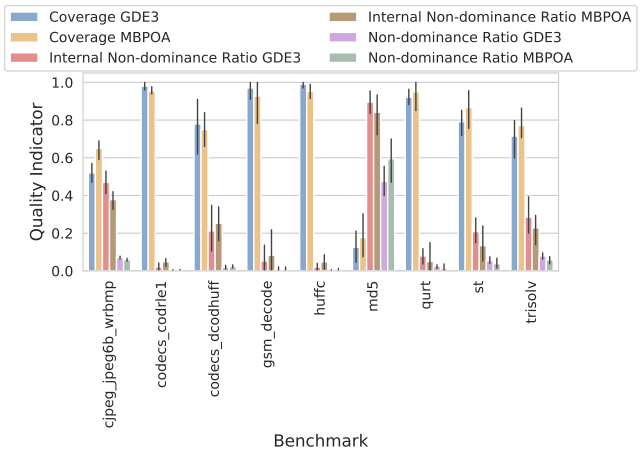
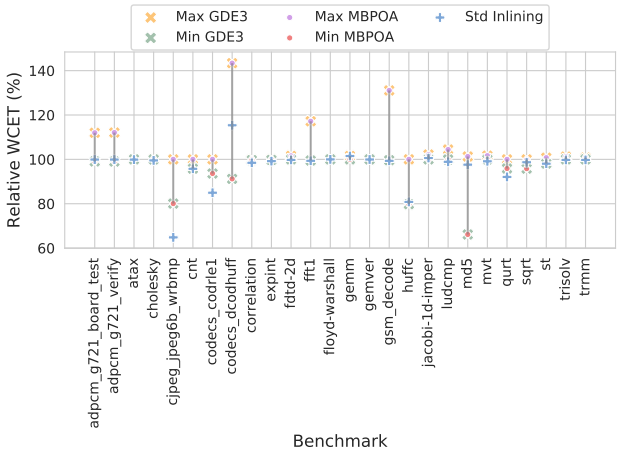


Figure 7: Quality indicators for the fixed parameters: crossover is 0.7 (for both algorithms), bandwidth is 500 and scaling factor is 0.5. Every benchmark was evaluated 5 times for different initial populations.

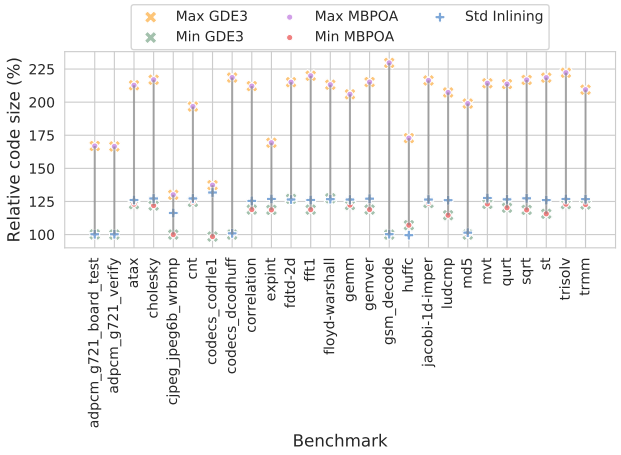
the search space of the benchmark *gsm_decode* is much smaller, both algorithms show much better results for the benchmark *trisolv*.

In practice, setting the control parameters to the values, which lead to the best quality indicators' values in average, could be enough. However, at this point it is clear that, unfortunately, it is impossible to find unique values for control parameters that lead to the best behavior for all benchmarks. Additional evidence for this is the results presented in Table 1. Column *Best Parameters Values* lists the best values for the control parameters for each benchmark. Quality indicators presented in Table 1 show that in case of choosing control parameters specifically for every benchmark GDE3 outperforms MBPOA in all benchmarks. The reason for this is that GDE3 has more control parameters that can be set by a user and allow to control the behavior of the algorithm better. To sum up, on the one hand, GDE3 has 3 control parameters that have to be defined by a user, whereas MBPOA has only one parameter. On the other hand, selecting and setting carefully the control parameters allows to achieve a better performance of GDE3 algorithm.

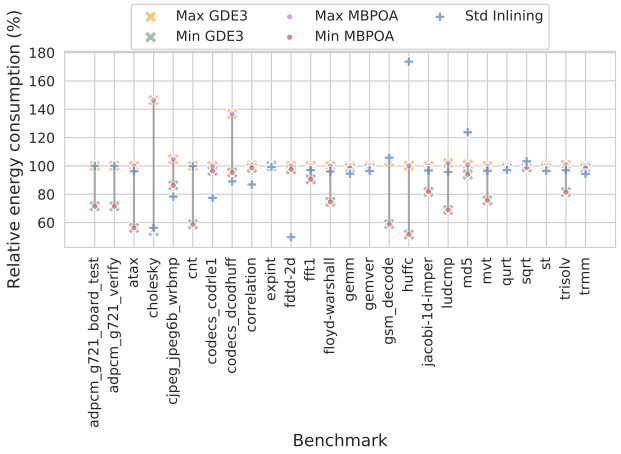
Our aim was to find all possible trade-offs between the objectives and, therefore, the results of our approach cannot be directly compared to the standard single-objective heuristics. Nevertheless, we present the results of the standard function inlining in order to show the position of its solution in the search space of the multi-objective function inlining. The standard function inlining follows a very simple rule: a function is inlined, if its number of expressions at high-level representation (cf. Section 3) is less than or equal to 20. Figure 8 presents the objective values returned by the standard function inlining, where 100 % corresponds to the results considering the optimization level O2 without function inlining. Moreover, recall that the true Pareto front is infeasible for the problem considered in the paper. Hence, to find the best approximation of the true Pareto front by every considered algorithm for every benchmark, we merge all approximated Pareto fronts returned by



(a) Worst-case execution time (WCET).



(b) Code size.



(c) Energy consumption.

Figure 8: Minimum and maximum values of the objectives in the approximated Pareto front and the objectives' values corresponding to the standard function inlining.

GDE3 or MBPOA algorithm after 5 repetitions and select all non-dominated points of the merged sets. Eventually, in Figure 8, we present the extreme solutions, i.e., the maximum and minimum values for the objectives, of the sets of non-dominated points for each algorithm. The results of both algorithms are exactly the same as shown in the figures.

Figure 8a shows that in terms of WCET for many considered benchmarks, the standard function inlining returns the WCET that corresponds to the lower bound of the approximated Pareto front for both algorithms. However, for the benchmarks *cjpeg_jpeg6b_wrbmp*, *codecs_codrle1*, and *qurt*, both evolutionary algorithms were not able to find the smallest WCET value returned by the standard inlining.

In case of code size, Figure 8b demonstrates that the values of the standard function inlining are always very close to the smallest code size value of the approximated Pareto front. Nevertheless, for the benchmark *codecs_codrle1*, the code size returned by the standard function inlining is almost equal to the maximum value. Meanwhile, considering the benchmark *huffc*, the evolutionary algorithms could not find the best value corresponding to the result of the standard optimization.

The last considered objective is energy consumption. As shown in Figure 8c, on the one hand, for many benchmarks the standard function inlining results in the almost maximal values. However, on the other hand, for some benchmarks, the standard function inlining result is not in the range of minimum and maximum values of the found approximated Pareto front. Moreover, for the benchmark *huffc*, the standard function inlining returns a very large value for the energy consumption, whereas the code size and WCET are small. However, for the benchmark *md5*, only the code size coincides with the lower bound of the approximated Pareto front, whereas WCET is very close to the original value and energy consumption increases by more than 20 %. Another interesting result is considering the benchmark *fdtd-2d*. The WCET and code size for this benchmark are equal to the smallest values of the objectives and the energy consumption is much smaller than the value found by evolutionary algorithms. Obviously, the standard function inlining results in this case to the extreme solution from the Pareto front that the algorithms did not find.

To summarize, the results from Figure 8 show that for the considered benchmarks the standard function inlining results in a solution that is very close to the point on the Pareto front which has small WCET and code size values. Meanwhile, the energy consumption of the standard approach is usually very close to the original value of the input problem which also corresponds the upper bound of the found approximation of the Pareto front. Moreover, in some cases, the standard function inlining results in a solution from the approximated Pareto front which the evolutionary algorithms were not able to find. One of the reasons for this is the limitation described before that we set the population size and the number of generations for the evolutionary algorithms due to the large runtime of an algorithm execution (cf. Table 1). Unfortunately, in this case an evolutionary algorithm cannot fully explore the search space in order to find the best approximation of the true Pareto front. However, the standard function inlining cannot provide a user with a set of possible trade-offs between the objectives, whereas, as we have seen, function inlining formulated as a multi-objective problem results in a solution set which describes many different scenarios of the compiled program.

6 CONCLUSION

We have formulated the well-known compiler-based optimization function inlining as a multi-objective optimization problem. WCET, code size, and energy consumption were considered as objectives for the optimization. The results show that three considered objectives cannot be optimized simultaneously, because they contradict each other. For this reason, we exploited the multi-objective evolutionary algorithms to find the best trade-off solutions. We presented the results for two evolutionary algorithms, namely GDE3, that was modified in order to solve a binary-encoded problem, and MBPOA.

Both algorithms contain user defined control parameters. We made the choice of the best values of them based on experiments using benchmarks. After fixing the values of the control parameters, we demonstrated the comparison of the algorithms. For some benchmarks, MBPOA algorithm outperforms GDE3. However, we have shown that carefully chosen control parameters allow GDE3 to return a better solution set than MBPOA.

Both evolutionary algorithms are able to find the trade-offs between WCET, code size, and energy consumption. Moreover, we compared the results of the standard function inlining with the approximated Pareto front found by evolutionary algorithms. In some case the standard function inlining resulted to a non-dominated solution that any considered evolutionary algorithm did not find. One of the reasons for this is that an approach, which uses evolutionary algorithms, requires extensive evaluations of the objectives, whereas the computation of WCET and energy consumption at compile time is very time-consuming. For this reason, trying to find the values for the control parameters that have great impact on the algorithm's performance, first of all, only limited number of benchmarks can be utilized, secondly, the population size for every algorithm has to be limited by a small value. On the one hand, MBPOA has just one user defined control parameter in contrast to the modified GDE3 that has three parameters. On the other hand, choosing the control parameters specifically for every benchmark allows GDE3 to show the more promising results comparing to MBPOA, because more control parameters give more freedom for the user to control the behavior of the algorithm. In addition, due to expensive evaluations of the objectives, the experiments for quite large benchmarks are also almost impossible.

One possible way to overcome the issue described above is to minimize the number of expensive evaluations. E.g., the objectives' values can be predicted at some points of the search space using a surrogate model, which is a fast approximation of the original objective function in terms of evaluations. This approach is considered as a possible extension of the proposed multi-criteria function inlining. Predicting objectives' values and making smart decisions about the points at which the accurate expensive evaluations are required may lead to decrease in runtime of the optimization and enable experiments for larger benchmarks.

Besides, function inlining is just one optimization that we have considered as a multi-criteria problem. Other compiler-based optimizations also have a great potential to be reformulated and analyzed in this direction.

ACKNOWLEDGMENTS

This work received funding from Deutsche Forschungsgemeinschaft (DFG) under grant FA 1017/3-1 .

REFERENCES

[1] AbsInt Angewandte Informatik, GmbH. 2018. aiT Worst-Case Execution Time Analyzers.

[2] W. R. A. Dias and E. D. Moreno. 2012. Code Compression in ARM Embedded Systems Using Multiple Dictionaries. In *Proceedings of the 15th IEEE International Conference on Computational Science and Engineering* (Nicosia, Cyprus). 209–214. https://doi.org/10.1109/ICCSE.2012.36

[3] J. J. Durillo, A. J. Nebro, C. A. C. Coello, J. Garcia-Nieto, F. Luna, and E. Alba. 2010. A Study of Multiobjective Metaheuristics When Solving Parameter Scalable Problems. *IEEE Transactions on Evolutionary Computation* 14, 4 (2010), 618–635. https://doi.org/10.1109/TEVC.2009.2034647

[4] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sorensen, P. Wägemann, and S. Wegener. 2016. TACLeBench: A Benchmark Collection to Support Worst-Case Execution Time Research. In *Proceeding of the 16th International Workshop on Worst-Case Execution Time Analysis (OASIcs)*, Vol. 55. 2:1–2:10. https://doi.org/10.4230/OASIcs.WCET.2016.2

[5] H. Falk and P. Lokuciejewski. 2010. A Compiler Framework for the Reduction of Worst-Case Execution Times. *Real-Time Systems* 46, 2 (2010), 251–300. https://doi.org/10.1007/s11241-010-9101-x

[6] E.S. Helan, Mr. P.M. Sandeep, Mr.V.Suresh Babu, and Mr. M. Varatharaj. 2017. Compression and Decompression of Embedded System Codes. *International Journal of Engineering Trends and Technology* 45, 7 (2017), 325–330. https://doi.org/10.14445/22315381/IJETT-V45P268

[7] S. Jadhav and H. Falk. 2019. Multi-Objective Optimization for the Compiler of Real-Time Systems Based on Flower Pollination Algorithm. In *Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems (SCOPES '19)*. 45–48. https://doi.org/10.1145/3323439.3323977

[8] J. Knowles, L. Thiele, and E. Zitzler. 2006. *A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers.* Technical Report. ETH Zurich.

[9] S. Kukkonen and J. Lampinen. 2005. GDE3: the third evolution step of generalized differential evolution. *Proceedings of the IEEE Congress on Evolutionary Computation* 1 (2005), 443–450. https://doi.org/10.1109/CEC.2005.1554717

[10] P. Lokuciejewski, F. Gedikli, P. Marwedel, and K. Morik. 2009. Automatic WCET Reduction by Machine Learning Based Heuristics for Function Inlining. In *Proceedings of the Workshop on Statistical and Machine Learning Approaches to Architecture and Compilation (SMART '09)*. 1–15. https://doi.org/10.1.1.222.9895

[11] P. Lokuciejewski, S. Plazar, H. Falk, P. Marwedel, and L. Thiele. 2011. Approximating Pareto optimal compiler optimization sequences - a trade-off between WCET, ACET and code size. *Software: Practice and Experience* 41 (2011), 1437–1458. https://doi.org/10.1002/spe.1079

[12] P. Marwedel. 2006. *Embedded System Design.* Springer-Verlag, Berlin, Heidelberg.

[13] K. Muts, A. Luppold, and H. Falk. 2018. Multi-Criteria Compiler-Based Optimization of Hard Real-Time Systems. In *Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems (SCOPES '18)*. 54–57. https://doi.org/10.1145/3207719.3207730

[14] K. Muts, A. Luppold, and H. Falk. 2019. Compiler-Based Code Compression for Hard Real-Time Systems. In *Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems (SCOPES '19)*. 72–81. https://doi.org/10.1145/3323439.3323976

[15] D. Oehlert, A. Luppold, and H. Falk. 2017. Bus-aware Static Instruction SPM Allocation for Multicore Hard Real-Time Systems. In *Proceedings of the 29th Euromicro Conference on Real-Time Systems (LIPIcs)*. 1:1–1:22. https://doi.org/10.4230/LIPIcs.ECRTS.2017.1

[16] D. Oehlert, S. Saidi, and H. Falk. 2019. Code-Inherent Traffic Shaping for Hard Real-Time Systems. *ACM Transactions on Embedded Computing Systems* 18, 5s (2019), 108:1–108:21. https://doi.org/10.1145/3358215

[17] J. Pallister, S. J. Hollis, and J. Bennett. 2013. Identifying Compiler Options to Minimize Energy Consumption for Embedded Platforms. *Comput. J.* 58, 1 (2013), 95–109. https://doi.org/10.1093/comjnl/bxt129

[18] S. S. Pinter and I. Waldman. 2007. Selective Code Compression Scheme for Embedded Systems. In *Transactions on High-Performance Embedded Architectures and Compilers I*. 298–316. https://doi.org/10.1007/978-3-540-71528-3_19

[19] M. Roth, A. Luppold, and H. Falk. 2018. Measuring and Modeling Energy Consumption of Embedded Systems for Optimizing Compilers. In *Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems (SCOPES '18)*. 86–89. https://doi.org/10.1145/3207719.3207729

[20] V. Tiwari, S. Malik, and A. Wolfe. 1994. Power Analysis of Embedded Software: A First Step towards Software Power Minimization. *IEEE Tranaction on Very Large Scale Integration Systems* 2, 4 (1994), 437–445. https://doi.org/10.1109/92.335012

[21] P. Wägemann, T. Distler, T. Hönig, H. Janker, R. Kapitza, and W. Schröder-Preikschat. 2015. Worst-Case Energy Consumption Analysis for Energy-Constrained Embedded Systems. In *Proceedings of the 27th IEEE Euromicro Conference on Real-Time Systems (ECRTS '15)*. 105–114. https://doi.org/10.1109/ECRTS.2015.17

[22] L. Wang, X. Fu, Y. Mao, M. Menhas, and M. Fei. 2012. A novel modified binary differential evolution algorithm and its applications. *Neurocomputing* 98 (2012), 55–75. https://doi.org/10.1016/j.neucom.2011.11.033

[23] L. Wang, H. Ni, W. Zhou, P. M. Pardalos, J. Fang, and M. Fei. 2014. MBPOA-based LQR controller and its application to the double-parallel inverted pendulum system. *Engineering Applications of Artificial Intelligence* 36 (2014), 262 – 268. https://doi.org/10.1016/j.engappai.2014.07.023

[24] L. Wörteler, M. Grossniklaus, C. Grün, and M. H. Scholl. 2015. Function Inlining in XQuery 3.0 Optimization. In *Proceedings of the 15th Symposium on Database Programming Languages*. 45–48. https://doi.org/10.1145/2815072.2815079

[25] C. Xian, Y.-H. Lu, and Z. Li. 2007. Energy-aware Scheduling for Real-time Multiprocessor Systems with Uncertain Task Execution Time. In *Proceedings of the 44th Annual Design Automation Conference (DAC '07)*. 664–669.