A Comparative Evaluation of Multi-Objective Exploration Algorithms for High-Level Design

JACOPO PANERATI, École Polytechnique de Montréal
GIOVANNI BELTRAME, École Polytechnique de Montréal

This paper presents a detailed overview and the experimental comparison of 15 multi-objective Design Space Exploration (DSE) algorithms for high-level design. These algorithms are collected from recent literature and include heuristic, evolutionary and statistical methods. To provide a fair comparison, the algorithms are classified according to the approach used and examined against a large set of metrics. In particular, the effectiveness of each algorithm was evaluated for the optimization of a multi-processor platform, considering initial setup effort, rate of convergence, scalability, and quality of the resulting optimization. Our experiments are performed with statistical rigor, using a set of very diverse benchmark applications (a video converter, a parallel compression algorithm, and a fast Fourier transformation algorithm) to take a large spectrum of realistic workloads into account. Our results provide insights on the effort required to apply each algorithm to a target design space, the number of simulations it requires, its accuracy, and its precision. These insights are used to draw guidelines for the choice of DSE algorithms according to the type and size of design space to be optimized.

Categories and Subject Descriptors: B.7.1 [**Integrated Circuits**]: Types and Design Styles—*VLSI (very large scale integration)*; C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors); J.6 [**Computer-aided Engineering**]: *Computer-aided design (CAD)*

General Terms: Algorithms, Design, Experimentation, Measurement, Performance

Additional Key Words and Phrases: Design automation, Computer aided manufacturing, Pareto optimization, Guidelines

ACM Reference Format:
Jacopo Panerati and Giovanni Beltrame. 2013. A Comparative Evaluation of Multi-Objective Exploration Algorithms for High-Level Design. *ACM Trans. Des. Autom. Electron. Syst.* 0, 0, Article 00 (20XX), 22 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

The continuous increase of transistor density on a single die is leading towards the production of more and more complex systems on a single chip, with an increasing number of integrated components and processing units. This trend brought to the introduction of the System-On-Chip (SoC), that integrates on a single medium all the components of a full system. The design and development of such systems [Paulin and Knight 1989] raises challenges [Hill and Marty 2008] due to the large design space, and tight constraints [Martin 2006].

Parametrized embedded System-on-Chip (SoC) architectures must be optimally tuned, i.e. their configuration parameters must be appropriately chosen, to find the

This work was partially supported by the *Regroupement Stratégique en Microsystèmes du Québec* (ReSMiQ)
Author's addresses: J. Panerati and G. Beltrame, Département de Génie Informatique et Génie Logiciel, École Polytechnique de Montréal.

DOI: http://dx.doi.org/10.1145/0000000.0000000

best trade-off in terms of the selected figures of merit (e.g. energy, area, and delay) for a given class of applications. This tuning process is called *Design Space Exploration (DSE)* [Pimentel et al. 2006]. Using DSE, a designer can find the optimal *configuration* for a given system.

In general, this optimization problem involves the minimization (or maximization) of *multiple objectives*, making the definition of optimality not unique. The quality of a system configuration according to the various objectives is usually expressed using a set of *metrics* or *objective functions*. Solving *multi-objective* optimization problems consists of finding the points of the *Pareto curve* [Givargis et al. 2001], i.e. all the points which are better than all the others for at least one metric or objective function.

However, a Pareto curve for a specific platform is available only when all the points in the design space have been evaluated and characterized in terms of objective functions. This full search approach is often unfeasible due to the high cardinality of the design space, and to the high cost associated with the evaluation of the objective functions (e.g. long simulation times). A viable, but less expressive, solution would consist in the use of scalarizing functions [Miettinen and Mkel 2002]), which transform the problem into a single objective search.

Currently, Multi-Processor SoCs (MPSoCs) platforms are optimized using either by designer experience or by applying several different algorithms. Examples are classical heuristic algorithms (such as tabu search, simulated annealing, etc.) [Palesi and Givargis 2002], or pruning techniques that try to reduce the size of the design space [Mohanty et al. 2002]. These techniques rely on simulation (or estimation) for the evaluation of the system-level metrics corresponding to a newly found configuration.

Depending on the parameters of the design space (such as size and time needed for one simulation), algorithms show different performance and accuracy of results. This work identifies and compares 15 algorithms among several of the most recent approaches proposed in literature, for automated DSE with multiple performance metrics. These methods differ from theoretical background, applicability conditions, and performance. Through a rigorous analysis, we identify the advantages and drawbacks of each method, obtaining guidelines for their use with different applications.

The main contributions of this work are: (a) a comprehensive overview, as we take into account more and more diverse multi-objective optimization methodologies than previous surveys in the field, (b) a quantitative analysis, as our results provide numbers and metrics allowing a clear comparison of multi-objective optimization methodologies on the common ground of DSE for MPSoCs, and (c) insights, as we give recommendations for the choice of the most appropriate algorithm for a target design space.

The paper is structured as follows: Section 2 briefly reviews other comparative works in the field of multi-objective optimization and DSE; Section 3 describes the landscape of multi-objective optimization and DSE algorithms and proposes their partitioning into four classes; Section 4 summarizes the theory behind each one of the 15 algorithms included in our experimental comparison; Section 5 presents the experimental setup and the framework we used to compare the algorithms, while results are presented in Section 6; Section 7 contains the discussion of the results and the recommendations of the authors; finally, Section 8 concludes the paper.

2. RELATED WORK

Several other works reviewing and comparing state-of-the-art algorithms for multi-objective optimization exist in literature [Coello 1998; Zitzler et al. 1999b; Ulungu and Teghem 1994], with most of them providing algorithm descriptions in the form of a survey. In general, these surveys do not attempt quantitative comparisons and they do not provide any novel experimental evaluation.

Fonseca and Fleming [1995] claim that evolutionary algorithms (EA), since their appearance, quickly became one of the most popular ways to solve multi-objective problems. The paper provides theoretical insights on how to assign fitness values[1] for multi-objective problems, but doesn't provide guidelines for the choice of EAs to be used on specific problems.

Coello [2000] identifies three families of evolutionary approaches for multi-objective optimization based on the way the algorithms compute the fitness of individuals/solutions: algorithms using aggregating functions, algorithms using non-aggregating but non Pareto-based approaches, and Pareto-based approaches, e.g., the NSGA algorithm included in our experimental pool. Some experiments are reported, but the results are not systematically compared.

Marler and Arora [2004] broadens the discussion on multi-objective optimization including approaches other than evolutionary algorithms. Multi-objective optimization methodologies are divided into: (a) algorithms with *a priori* specification of preferences (e.g. scalarizing and utility functions), (b) algorithms with *a posteriori* specification of preferences, returning a set of Pareto points, (c) algorithms with no specification of preferences, typically simplifications of approaches from (a) with constant parameters, and (d) genetic algorithms (GA), e.g., the IMMOGLS algorithm also described in this work. The cost of programming and the computational complexity of the algorithms are compared only in a qualitative way.

In terms of experimental groundwork, the closest works to this paper were published in [Zitzler and Thiele 1999], [Zitzler et al. 1999a] and [Zitzler et al. 1999b]. Zitzler *et al.* compare seven evolutionary algorithms, including SPEA and NSGA, detailed later in this paper, using one realistic (the synthesis of a multiprocessor) and two artificial problems. As our results confirm, they discover that SPEA outperforms NSGA. The main flaw of these publications is that their scope is limited to evolutionary algorithms, while we consider algorithms exploiting very different methodologies.

Finally, it is worth mentioning Okabe et al. [2003], which summarizes several metrics for the evaluation of solutions to multi-objective optimization problems. Okabe et al. [2003] suggests that no single metric is sufficient to judge the quality of the Pareto set returned by a multi-objective optimization algorithm. Similar claims, together with innovative visualization tools for the Pareto sets discovered by multi-objective evolutionary algorithms, can be found in [Taghavi and Pimentel 2011]. In our work, we use four different metrics to evaluate the accuracy, distribution, and cardinality of the Pareto sets found by the optimization algorithms. Moreover, we compare the performance of 15 algorithms as the number of evaluations needed to converge to a Pareto set.

3. PROPOSED TAXONOMY

The challenge of automated DSE can be divided in two sub-problems: (a) the identification of candidate solutions (i.e. valid system configurations), (b) the evaluation of metrics of interest for such solutions, and the selection of the optimal configurations.

Considering the context of embedded systems design, we identify four main classes of techniques proposed in literature for the problem (a):

— **Class 1: Heuristics and pseudo-random optimization approaches,** that try to reduce the design space and focus the exploration on regions of interest [Givargis et al. 2001; Fornaciari et al. 2002]. These techniques normally rely on full search or

[1]The metric used by evolutionary algorithms to evaluate the goodness of a solution and its probability of being recombined.

Table I. Comparison of experimental settings in DSE literature.

Paper Ref.	Class	Approach	Optim. Metrics			Adjustable Param.		
			Delay	Power	Area	Proc. Units	Freq./Volt.	Cache Ass./Size
Palermo et al. [2008]	1	Particle Swarm Optimization	√	√				√
Lukasiewycz et al. [2008]	1	MO Pseudo Boolean		√	√	√		
Givargis et al. [2001]	1	Parameter Dependency Model	√	√			√	√
Palesi and Givargis [2002]	2	Genetic Algorithms	√	√				√
Fornaciari et al. [2002]	1	Sensitivity-Based	√	√				√
Sheldon et al. [2007]	3	Design of Experiments	√		√	√		√
Mariani et al. [2010]	3	Correlation-Based	√	√		√		√
Palermo et al. [2009]	3	Response Surface-Based	√	√		√		√
Beltrame et al. [2010]	4	Decision Theoretic	√	√		√	√	√

Table II. Comparison of design space size in DSE literature.

Paper Ref.	Class	Design Space Size
Palermo et al. [2008]	1	196608
Givargis et al. [2001]	1	$> 10^{14}$
Fornaciari et al. [2002]	1	9216
Palesi and Givargis [2002]	2	$5.97 \cdot 10^{12}$
Sheldon et al. [2007]	3	16384
Mariani et al. [2010]	3	2^{17}
Palermo et al. [2009]	3	2^{17}
Beltrame et al. [2010]	4	8640

pseudo-random algorithms to explore the selected regions. Metaheuristics such as simulated annealing (SA) and multi-agent optimization belong to this class.

— **Class 2: Evolutionary algorithms.** These techniques are the most common and widely used, they rely on random changes of a starting set of configurations to iteratively improve the system under analysis. In this category we find genetic algorithms [Palesi and Givargis 2002].

— **Class 3: Statistical approaches without domain knowledge.** These techniques extract a *metamodel* from the design space and use it to predict which new configurations to consider [Sheldon et al. 2007; Lukasiewycz et al. 2008; Palermo et al. 2009; Mariani et al. 2010].

— **Class 4: Statistical approaches with domain knowledge.** These techniques use pre-defined rules associated to the design space to find the most promising solutions [Beltrame et al. 2010].

Problem (b), i.e. the evaluation of candidate solutions, is usually addressed *via* two mechanisms (or a combination): detailed simulation [Beltrame et al. 2010] or estimation using predictive models [Jaddoe and Pimentel 2008].

Class 1 features a vast range of algorithms that are currently used in various domains, including: (a) multi-agent optimization such as particle swarm optimization (PSO) [Palermo et al. 2008], (b) simulated annealing, and (c) other operations research algorithms (such as tabu search). In Givargis et al. [2001], the design space is divided into partitions, and exhaustive search is performed inside each partition. Then the Pareto-optimal configurations of each partition are combined to determine the global curve. A different technique, proposed in Fornaciari et al. [2002], aims at reducing the number of configurations from the product of the number of parameters to their sum. Although complexity is highly reduced, the exploration results remain in many cases sub-optimal.

Class 2 techniques are probably the most widely used. Genetic algorithms and exact methods are combined in Lukasiewycz et al. [2008]: the design space exploration prob-

lem is formalized as a multi-objective 0-1 Integer Linear Programming (ILP) problem. A pseudo-boolean solver is used to force the genetic algorithm to stay in the feasible search space. These algorithms require a relatively small effort to describe design space, and they do not require specific knowledge associated to the domain or the metrics used for the exploration. However, they do not guarantee optimality or convergence within certain accuracy or precision bounds, even though they generally perform well when running a sufficient number of evaluations.

Class 3 methods rely on statistical analysis. A group of techniques, referred to as *Design of Experiments* (DoE) [Sheldon et al. 2007; Palermo et al. 2009; Mariani et al. 2010] is often used to characterize the impact of the parameters on the system. This means estimating the portion of the variance of the objective functions associated to the variation of each parameter. Once sensitivity analysis is performed, heuristics or metamodels are used to modify the parameters and determine the optimal system configuration. Palermo et al. [2009] use DoE to generate an initial set of experiments, creating a coarse view of the target design space. Response surface modeling (RSM) is then used to refine the exploration; this process is iterated to cover the design space. The solid statistical foundation of these algorithms allows to extract the maximum amount of information from the initial training set, and the generated metamodels can be used to find new candidate configurations or quickly evaluate potential solutions.

The algorithms in **Class 4** use expert knowledge to set up a probabilistic framework to determine the best candidate solutions. Beltrame et al. [2010] introduces the use of decision theory to exploit this expert knowledge. The idea is to move the design space exploration complexity from simulation to probabilistic analysis of parameter transformations. Exploration is modeled as a Markov decision process (MDP) [Russell and Norvig 1995; Kaelbling et al. 1996], and the solution to such MDP corresponds to the sequence of parameter transformations to be applied to the platform to maximize (or minimize) a desired value function. This approach requires to simulate the system only in particular cases of uncertainty, massively reducing the simulation time needed to perform the exploration of a system, while maintaining the near-optimality of the results.

In this work we collected algorithms from all the four classes and we compared their strengths and weaknesses using a realistic benchmark scenario: a symmetric multi-processor platform. The results provide guidelines in the use of multi-objective optimization algorithms for DSE.

To allow a fair and clear comparison with previous DSE works, Table I reports the optimization metrics and parameters of previous publications, Table II contains information on the size of the explored domain spaces, while Table III briefly lists the software applications originally used as benchmarks.

4. MULTI-OBJECTIVE ALGORITHMS FOR DESIGN SPACE EXPLORATION

We include fifteen multi-objective algorithms for DSE in our experimental comparison.

4.1. Class 1

4.1.1. Adaptive Windows Pareto Random Search (APRS). The APRS algorithm is one of the two novel algorithms implemented in the Multicube Explorer framework [Zaccaria et al. 2010]. The algorithm starts with an initial set of candidate Pareto points and then attempts to improve the Pareto set by picking randomly among the points inside windows centered on the current points. The size of the windows is reduced over time (at each iteration) and proportionally with the quality of the point associated to them.

4.1.2. Multi-Objective Multiple Start Local Search (MOMSLS). Local search is a simple heuristic that iteratively refines an initial random solution by looking for improve-

Table III. Comparison of benchmark applications in DSE literature.

Paper Ref.	Class	Benchmark Applications
Palermo et al. [2008]	1	**FIR** (finite impulse filter), **Gamma** and **DCT** (numeric algorithms), **Gauss** and **Quarcube** (equations solvers).
Lukasiewycz et al. [2008]	1	**ALC** (adaptive light control), a large automotive design problem.
Givargis et al. [2001]	1	**"JPEG"**, a *jpeg* compression algorithm using the on-chip DCT CODEC core to perform the forward DCT transform.
Fornaciari et al. [2002]	1	**MESA**, a 3D graphics library close to OpenGL, **GSM06.10**, European standard encoder and decoder.
Palesi and Givargis [2002]	2	**Motorola PowerStone**, a collection of embedded and portable applications.
Sheldon et al. [2007]	3	A set of **EEMBC** (Embedded Microprocessor Benchmark Consortium) benchmark applications.
Mariani et al. [2010]	3	A set of benchmarks derived from the Stanford Parallel Applications for Shared Memory (**SPLASH**) suite.
Palermo et al. [2009]	3	**MPEG2** decoder application.
Beltrame et al. [2010]	4	**ffmpeg**, **pigz** and **fft6** (see Table IX for additional details).

Table IV. The most relevant parameters used by the MOPSO algorithm.

MOPSO [Palermo et al. 2008] - Class 1		
Param.	**Description**	**Val.**
W	inertia weight	$unused$
C_1	social learning factor	$unused$
C_2	cognitive learning factor	1
p	probability of taking a *random walk*	$0.9, 0.5, 0.2$

ments in a neighbourhood of the current solution [Russell and Norvig 1995]. MOMSLS [Jaszkiewicz and Dbrowski 2005] simply consists of a local search procedure using multiple solution points in its initial step. At each step the algorithm looks for better solutions in the N neighbourhoods of all the current solutions.

4.1.3. Multi-Objective Particle Swarm Optimization (MOPSO). Particle Swarm Optimization (PSO) [Kennedy and Eberhart 1995] is a heuristic search methodology that finds its biological inspiration in the behaviour of flocks of birds. At each search iteration, the particles in the swarm move towards an objective using a velocity vector that is the linear combination of three components:

— the previous velocity vector (weighted by inertia, W)
— the direction towards the best (i.e. closest to the objective) position ever reached by the swarm (weighted by a social learning factor, C_1)
— the direction towards the best position ever reached by the specific particle (weighted by a cognitive learning factor, C_2)

In multi-objective particle swarm optimization (MOPSO) [Palermo et al. 2008], PSO is applied to a multi-objective domain by using N swarms, each of which having as an objective the product of the multiple objectives combined with randomly chosen exponents.

In MOPSO, inertia and social learning factors are unused: in order to avoid local minima, at each iteration, particles are forced to move with random velocity (a random walk) with a fixed probability p.

4.1.4. Multi-Objective Simulated Annealing (MOSA). Simulated annealing (SA) is a local search technique that uses a special technique to avoid local minima: solutions that

Table V. The most relevant parameters used by the MOSA, PSA and SMOSA algorithms.

MOSA [Ulungu et al. 1999], **PSA** [Czyzak and Jaszkiewicz 1998], **SMOSA** [Serafini 1994] - Class 1		
Param.	**Description**	**Val.**
N_p	generating population size	10
T_0	initial temperature used to parametrize the Boltzmann distribution	2.50
T_f	final temperature used to parametrize the Boltzmann distribution	0.1
γ	weights change coefficient	0.1

are worse than the current are rejected with a given probability p, computed using a Boltzmann distribution (parametrized by a coefficient T, called temperature).

Ulungu et al. [1999] propose two ways of applying simulated annealing to multi-objective problems:

(1) probability scalarization, the computation of rejection probability for each performance metric and their aggregation
(2) criterion scalarization, the projection of the performance metrics into a single metric, and the use of such metric to compute the rejection probability of the new solution.

The latter approach is the one implemented in MOSA [Ulungu et al. 1999], and tested in this paper using the parameters in Table V, while approach (1) is the one used by the other SA-based algorithms presented below.

4.1.5. Pareto Simulated Annealing (PSA). PSA [Czyzak and Jaszkiewicz 1998] proposes two different criteria for the application of simulated annealing to multi-objective problems:

(1) rule C says the rejection probability p is proportional to the largest value among the differences between the performance metrics of the current and the new solution;
(2) rule SL states that p is a weighted linear combination of the differences between the performance metrics of the current and the new solution.

Concerning rule SL, the weights used to multiply each metric are increased or reduced at each iteration, depending whether the most recently introduced solution brought a deterioration in that specific metric or not.

See Table V for the algorithm specific parameters used in our experiments.

4.1.6. Serafini's Multiple Objective Simulated Annealing (SMOSA). Serafini [1994] propose several rules for the combination of multiple performance metrics in order to apply simulated annealing in the context of multi-objective optimization. Together with reviewing the C and SL rules described in [Czyzak and Jaszkiewicz 1998], a new composite rule is introduced. This rule is the linear composition, with coefficients α and $(1 - \alpha)$, of two simpler rules:

(1) rule P, saying that the rejection probability p is proportional to the product of the differences between the performance metrics of the current and the new solution;
(2) and rule W, saying that p is proportional to the smallest value among the differences between the performance metrics of the current and the new solution.

4.2. Class 2

4.2.1. Multiple Objective Genetic Local Search (MOGLS). MOGLS [Ishibuchi and Murata 1996] is an algorithm combining two well known methodologies, genetic algorithms and local search. Algorithms combining multiple search methodologies are usually

Table VI. The most relevant parameters used by the IMMOGLS, MOGLS and PMA algorithms.

IMMOGLS [Ishibuchi and Murata 1998], **MOGLS** [Ishibuchi and Murata 1996], **PMA** [Jaszkiewicz 2004] - Class 2		
Param.	**Description**	**Val.**
N_p	population size	20
N_i	number of iterations	7
f	scalarizing function family	$linear$

called hybrid approaches. Genetic algorithms start from a set of possible solutions, called population, and come up with new tentative solutions by combining couples of existing solutions X picked with a probability p given by a fitness function $f(X)$ [Sivanandam and Deepa 2007]. At each iteration of the MOGLS algorithm, (a) new solutions are generated using genetic operations and (b) a local search is performed in the neighbourhoods of these new solutions. The algorithm-specific parameters used in our experiments for MOGLS and the other genetic-local hybrid approaches, IMMOGLS and PMA, are reported in Table VI.

4.2.2. Ishibuchi-Murata Multi-Objective Genetic Local Search (IMMOGLS). IMMOGLS [Ishibuchi and Murata 1998] is similar to MOGLS, being the combination of genetic algorithms and local search. In a multi-objective minimization problem, a solution is said to be non-dominated with respect to a set of solutions, if it scores the minimum value in at least one of the metrics we want to minimize, the solution is dominated otherwise. At each iteration, IMMOGLS performs both genetic operations and a local search with three characteristics:

(1) the fitness function is a linear combination of the optimization metrics and the weights are chosen randomly at each iteration;
(2) the local search is limited to a number of k neighbours, where k is random;
(3) at each iterations, the current population is purged of any dominated solutions (elitist strategy).

4.2.3. Non-Dominated Sorting Genetic Algorithm (NSGA). NSGA [Srinivas and Deb 1994] is a straightforward application of the genetic approach to multi-objective optimization. At each iteration, NSGA assigns to the Pareto points in the current population their fitness values on the basis of non-domination. All non-dominated solutions are assigned the same fitness value. See Table VII for the algorithm specific parameters used in our experiments.

4.2.4. Controlled Non-Dominated Sorting Genetic Algorithm (NSGAII). Deb and Goel [2001] present NSGAII, an evolution of NSGA with two main differences:

(1) NSGAII is an elite-preserving algorithm,i.e., non-dominated solutions cannot be removed from the current population;
(2) solutions are sorted based on non-domination to reduce computational complexity.

4.2.5. Pareto Memetic Algorithm (PMA). PMA [Jaszkiewicz 2004] belongs to the family of hybrid genetic-local search algorithms, together with IMMOGLS and MOGLS. Its very own peculiarity resides in the way used to select the couple of current solutions that are combined with genetic operations: these two solution are not directly drawn from the current population. Instead, PMA samples with repetition a new set of solutions T from the current population. Then, the best two solutions in T are chosen for recombination.

Table VII. The most relevant parameters used by the NSGA, NSGAII and SPEA algorithms.

NSGA [Srinivas and Deb 1994], **NSGAII** [Deb and Goel 2001], **SPEA** [Zitzler and Thiele 1999] - Class 2		
Param.	**Description**	**Val.**
N_p	generating population size	15
N_g	number of generations	55
p	mutation probability	0.2

4.2.6. Strength Pareto Evolutionary Algorithm (SPEA). SPEA [Zitzler and Thiele 1999] belongs to the broad family of heuristic search methods called evolutionary algorithms. Its peculiarities are:

(1) all the non-dominated solutions are stored and preserved in a second external population;
(2) the fitness of a solution in the current population is determined only from the solutions stored in the external non-dominated set [Zitzler and Thiele 1999];
(3) a clustering step is applied to the non-dominated population in order to keep it small while preserving its characteristics.

4.3. Class 3

4.3.1. Response Surface Pareto Iterative Refinement (ReSPIR). ReSPIR [Palermo et al. 2009] is a DSE algorithm that uses statistical tools to create and keep an internal representation of the relations between the configuration parameters and the performance metrics in order to minimize the number of evaluations needed for successful optimization. These statistical tools are:

(1) Design of experiments (DoE), a methodology allowing to maximize the information gained from a set of empirical trials;
(2) response surface models (RSM), analytical representations of an objective trained with the available data. Different models can be used: linear regression, Shepard-based interpolation, artificial neural networks (ANN), etc.

The algorithm iteratively defines (using DoE) a set of experiments to be performed, trains the RSMs with the information collected and then produces an intermediate Pareto set.

4.4. Class 4

4.4.1. Markov Decision Process Optimization (MDP). The approach proposed by Beltrame et al. [2010] is based on a framework for sequential decision making called Markov decision process. Its components are states, actions, stochastic transitions from state to state, and rewards. The framework allows to find the correct action to perform in each state to collect the largest amount of rewards [Russell and Norvig 1995]. In [Beltrame et al. 2010], states are parameters configurations with their associated performance metrics, actions are parameters changes and rewards are improvements in the performance metrics. Stochastic transitions are initially believed to have uniform distribution and their estimates are refined over execution. It is worth noting that the algorithm requires built-in domain knowledge of the upper and lower bounds of each performance metric as a function of the tuning parameters. For this reason, MDP has a long set-up time and cannot be used without extensive domain knowledge regarding the platform used by the system under design. Table VIII reports the algorithm-specific parameters used in our experiments.

Table VIII. The most relevant parameters used by the MDP algorithm.

MDP [Beltrame et al. 2010] - Class 4		
Param.	**Description**	**Val.**
l	event horizon, maximum length of a decision path	3
ϵ	convergence margin, stopping criterion	10^{-6}
λ	accuracy factor, used to tune the discretization of the domain space	0.3
$\|A\|$	the number of α values, i.e., number of modifications that can be applied to a parameter	6

4.4.2. Multi-Objective Markov Decision Process (MOMDP). MOMDP [Beltrame and Nicolescu 2011] is an improved version of MDP. The main difference with respect to MDP is that MOMDP uses a different exploration strategy. MDP considers different objectives usign a parametric scalarizing function, and varies a parameter (called α) which represents the weight(s) associated to each objective. By sweeping α values, it is possible to discover a Pareto curve for a given number of separate obectives.

MOMDP uses a different approach: it maximizes (or minimizes) one of the objectives to derive a starting point, and then builds the Pareto curve using a value function that selects a point that is close to the starting point, but improving it in at least one of the objectives. The process is repeated using the newly found point until a full Pareto front is discovered.

MOMDP also introduces a special action, called the leap of faith, that allows to avoid local minima by searching in the direction of high rewards, however unlikely. This action is performed when all actions fail to improve any of the metrics. The algorithm-specific parameters used in the experiments are the same as the ones in Table VIII, with the exception of the event horizon, that was increased to 4.

5. EXPERIMENTAL SETUP

We collected the implementation of the 15 multi-objective algorithms that we evaluated and compared from different sources. Most class 1 and class 2 algorithms algorithms are implemented in the Multiple Objective MetaHeuristics Library in C++ (MOMHLib++) by Jaszkiewicz and Dbrowski [2005]. We also draw from the work of Zaccaria et al. [2010], Multicube Explorer (M3Explorer), that implements standard and enhanced versions of several well-known multi-objective optimization algorithms. Multicube Explorer provides some of the DSE algorithms in classes 1 and 3. Concerning the algorithms in class 4, MDP and MOMDP, we used the original source code. All of these libraries and research projects are open-source and the authors provide open access to their implementations.

Our experiments aim at:

— determining the effort needed to apply each algorithm to a given design space, and how the design space's characteristics drive the choice of the most effective exploration algorithm
— determining the number of evaluations required by each algorithm to obtain an approximate Pareto-set of given quality
— quantifying the properties of the Pareto-set found by each algorithm.

A qualitative comparison of the 15 algorithms is presented in Table XI. Each algorithm was applied to the same design space: a symmetric multi-processor platform running three different applications, shown in Figure 1. The platform consists of a collection of ARM9 cores with private caches, and a shared memory, interconnected by a simple system-bus model. Cache coherency is directory-based and using the MESI protocol. The ReSP [Beltrame et al. 2009] open-source simulation environment was

Table IX. The three benchmark applications chosen to represent a significant spectrum of workloads.

Application	Version	Source	Model	Synch.	Sim. Time	Description
pigz	2.1	C	pthreads	condition	~2m	a parallel implementation of gzip
fft6	2.0	C/Fortran77	OpenMP	barrier	~30s	implementation of Bailey's 6-step fast Fourier transformation algorithm
ffmpeg	49.0.2	C	pthreads	semaphore	~30m	a fast video and audio converter

Table X. The platform design space simulated using ReSP.

Parameter Name	Domain
# of PEs	{1,2,3,4,8}
PE Frequency	{100,200,250,300,400,500} MHz
L1 Cache Size	{1,2,4,8,16,32} KByte(s)
Bus Latency	{10,20,50,100} ns
Memory Latency	{10,20,50,100} ns
L1 Cache Policy	{LRU, LRR, RANDOM}

used to perform the simulations, providing a set of configurable parameters, listed in Table X. ReSP provides values for execution time and power consumption, which were used as the performance metrics for all optimization algorithms in our experiments.

The three applications used for testing are, more specifically, two large applications and a small benchmark, for which exhaustive search was possible, as listed in Table IX. emphffmpeg, a video transcoder, was used to convert a small clip from MPEG-1 to MPEG-4, and *pigz*, a parallel compression algorithm, was used to compress a text file. The small benchmark consists of an implementation of Bailey's 6-step FFT algorithm (*fft6*). All applications are data-parallel and are targeted towards a homogeneous shared-memory multi-processor platform (N processors accessing a common memory via bus). *ffmpeg* and *pigz* are implemented using pthreads, they create a set of working threads equal to the number of available processors and dispatch independent data to each thread. *fft6* uses OpenMP, with loop parallelization and static scheduling. These applications were specifically chosen in order to guarantee the maximum variability in their behaviour, in fact all applications use a different synchronization mechanism and require very different evaluation times.

The platform was explored using the parameters listed in Table X with a resulting design space of 8640 points[2], comparable with similar works (e.g. 6144 points in [Sheldon et al. 2007]). and the exhaustive exploration of any medium/large application would require an unfeasibly long simulation time (e.g. roughly two months for *ffmpeg*). Even the full exploration of the simple *fft6* benchmark required six days of uninterrupted simulation. To gather sufficient data for a statistical analysis, each of the 3 benchmark applications was optimized 10 times with each exploration algorithm ($N = 30$ executions for each algorithm).

According to Taghavi and Pimentel [2011], the quality of the result of a multi-objective optimization algorithm is two-fold: 1) solutions should be as close as possible to the actual Pareto set, and 2) solutions should be as diverse as possible. Therefore, no single metric is sufficient in assessing the quality of the discovered Pareto set.

[2]It is worth noting that bus and memory latency are not realistic parameters, but they enlarge the design space to better test the proposed algorithm. The linear dependence with performance prevents any strong biasing of the results.

We use the three metrics presented in [Erbas 2006] to compare the relative quality of the approximate Pareto sets obtained by each algorithm:

— **ADRS** – The Average Distance from Reference Set is used to compare the approximated Pareto-sets with the best Pareto set found combining the results of all experiments. This approximates the distance of a considered set from the Pareto-optimal front, and should be minimized.
According to its definition in [Zaccaria et al. 2010], the ADRS between an approximate Pareto set Λ and a reference Pareto set Π is computed as:

$$ADRS(\Pi, \Lambda) = \frac{1}{|\Pi|} \sum_{\boldsymbol{a} \in \Pi} \left(\min_{\boldsymbol{b} \in \Lambda} \{ \delta(\boldsymbol{b}, \boldsymbol{a}) \} \right) \tag{1}$$

where the δ function stands for:

$$\delta(\boldsymbol{b}, \boldsymbol{a}) = max_{j=1,..,m} \left\{ 0, \frac{\phi_j(\boldsymbol{a}) - \phi_j(\boldsymbol{b})}{\phi_j(\boldsymbol{b})} \right\} \tag{2}$$

Parameter m is the number of objectives and $\phi_i(\boldsymbol{a})$ is the value of the i-th objective metric measured in point $\boldsymbol{a}$.

— **Non-uniformity** – We measure how solutions are distributed in the design space. Lower non-uniformity means a more evenly-distributed approximate Pareto-set that better estimates the optimal Pareto-set.
Given a normalized Pareto set $\bar{\Lambda}$, where d_i is defined as the Euclidean distance between to consecutive points ($i = 1, .., |\bar{\Lambda}| - 1$), and $\hat{d}$ is the average values of all the d's, non-uniformity [Erbas 2006] can be computed as:

$$\sum_{i=1}^{|\bar{\Lambda}|-1} \frac{|d_i - \hat{d}|}{\sqrt{m}(|\bar{\Lambda}| - 1)} \tag{3}$$

— **Concentration** – We measure the span of each Pareto-set with respect to the range of the objectives. The lower the concentration, the higher the spread of the Pareto-set and the better coverage of the range of objectives.
Given a normalized Pareto set $\bar{\Lambda}$, where ϕ_i^{min} is defined as $\min\{\phi_i(\boldsymbol{a}) \; s.t. \; \boldsymbol{a} \in \bar{\Lambda}\}$ and ϕ_i^{max} is defined as $\max\{\phi_i(\boldsymbol{a}) \; s.t. \; \boldsymbol{a} \in \bar{\Lambda}\}$, concentration [Erbas 2006] can be computed as:

$$\prod_{i=1}^{m} \frac{1}{|\phi_i^{max} - \phi_i^{min}|} \tag{4}$$

6. EXPERIMENTAL RESULTS

6.1. Dependence on parameters and initial setup effort

All the examined algorithms differ in the way they converge to an approximate Pareto front, and the quality of their results depends on a number of different parameters, making a fair evaluation difficult to implement. There is no common rule for the choice of each algorithm's parameters: these range from four to twelve, and they can be anything from integers to the choice of an interpolation function. The selection of the parameters that are optimal to a specific optimization problem requires either expertise or it can be calculated by meta-exploring (also known as parameter screening) the parameters on the target design space, i.e. running multiple explorations while changing the parameters to optimize the result. Either way, finding the optimal parameters usually requires trial and error. The Effort column of Table XI qualitatively presents the tuning cost required by each algorithm.

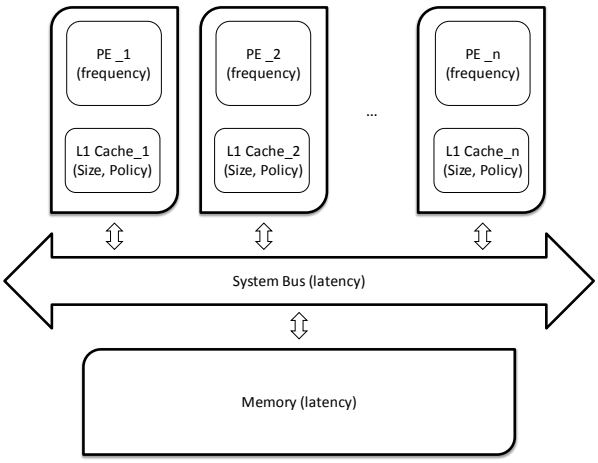

Fig. 1. The simulated multi-core processor architecture and its parameters.

Algorithms of classes 1 and 2 require few parameters (such as population size, mutation factors, initial temperature, etc.), and are generally robust to parameter choice. This means that small changes will not dramatically affect the outcomes of the exploration, although there is no guarantee that a given parameter choice will lead to optimal results. Their parameters have no direct link to any knowledge of the design space (e.g. initial temperature for MOSA and NSGAII), and can be determined only by experience or guesswork, rendering the best combination very difficult to obtain without screening and additional evaluations. In this work, we determined the best parameters via screening, which required running several thousand evaluations.

Algorithms like APRS do not require any special tuning, and rely on pre-determined heuristics, making their setup effort minimal. It is worth noting that both class 1 and 2 algorithms do not guarantee convergence to the optimal Pareto set and require that the user specify a maximum number of iterations in addition to any other stopping condition (e.g. when the results do not vary for more than two iterations).

Algorithms of class 3 demand a higher setup effort: the choice of a proper metamodel for a design space requires some expertise and an initial screening (and therefore additional evaluations) to properly determine which parameters are the most significant. Each metamodel needs specific additional parameters that are loosely linked to the designer's expertise of the design space. For our experiments we relied on the results described in Palermo et al. [2009], and we chose the Central Composite Design using a neural network (NN) interpolator. However, the NN produced results with a very high variance, with ADRS ranging from 0% to 160%, making the use of this interpolator impractical. We found the Shepard interpolation much more effective, although it required to determine the value of a *power* parameter, which expresses how jagged is the response surface of the design space. A low value of this *power* parameter will produce a smooth interpolation, while a higher value could better follow a more jagged curve, but could also introduce overfitting. We effectively replicated the results of Palermo et al. [2009] on our design space using a power of 16, which was found by parameter screening ($\sim 10^3$ evaluations).

Finally, algorithms of class 4 require a bound to be associated to the effects of parameter variations on the configuration's metrics. These bounds are left to the designer's experience, or can be determined via statistical modelling. This means detailed analysis of each design space, and large setup effort. The main difference between MDP

Table XI. A qualitative analysis of the chosen algorithms: setup effort, number of evaluations for 1% ADRS, number of Pareto points found, scalability

Acronym	Class	Effort	Evaluations	Pareto Points	Scalability
APRS [Zaccaria et al. 2010]	1	☆	☆☆☆☆☆	☆☆	☆
MOMSLS [Jaszkiewicz and Dbrowski 2005]	1	☆	☆☆☆	☆	☆☆☆
MOPSO [Palermo et al. 2008]	1	☆☆	☆☆☆☆	☆☆☆	☆☆☆☆
MOSA [Ulungu et al. 1999]	1	☆☆☆	☆☆☆☆	☆☆☆	☆☆☆☆
PSA [Czyzak and Jaszkiewicz 1998]	1	☆☆☆	☆☆☆	☆	☆☆☆☆
SMOSA [Serafini 1994]	1	☆☆☆	☆	☆	☆☆☆☆
MOGLS [Ishibuchi and Murata 1996]	2	☆☆	☆☆☆	☆☆☆	☆☆☆☆
IMMOGLS [Ishibuchi and Murata 1998]	2	☆☆	☆☆☆	☆☆	☆☆☆☆
NSGA [Srinivas and Deb 1994]	2	☆☆☆	☆☆☆	☆	☆☆☆☆
NSGAII [Deb and Goel 2001]	2	☆☆☆	☆☆☆☆	☆	☆☆☆☆
PMA [Jaszkiewicz 2004]	2	☆☆	☆☆☆	☆☆	☆☆☆☆
SPEA [Zitzler and Thiele 1999]	2	☆☆	☆☆☆	☆☆☆	☆☆☆☆
ReSPIR [Palermo et al. 2009]	3	☆☆☆☆	☆☆	☆☆☆☆	☆☆☆
MDP [Beltrame et al. 2010]	4	☆☆☆☆☆	☆	☆	☆☆
MOMDP [Beltrame and Nicolescu 2011]	4	☆☆☆☆☆	☆	☆☆☆	☆☆☆

and MOMDP is that the former is fundamentally a single-objective optimization algorithm. To effectively discover a Pareto-set, MDP "sweeps" the design space according to a set of scalarizing values that express the desired trade-off between the different objective functions. To determine the size and values of these scalarizing values for our design space, hundreds of additional evaluations were necessary. MOMDP does not require this screening, and its only parameter (the accuracy λ) is in fact the average simulation error, and can be chosen without effort.

It is worth noting that designer experience can reduce or remove the need for parameter discovery activities for all the aforementioned algorithms.

6.2. Estimation of the Number of Evaluations

In our experiments, we have tuned each algorithm parameters to obtain the best results for design space described in this paper, and we **do not** include the evaluations needed for screening and initial parameter estimation. Concerning ADRS, since exhaustive search is not possible, we compare the Pareto set generated by each algorithm with the best Pareto set found compounding all evaluations performed by all algorithms, which covers a sizable portion of the entire design space (around 30%).

Figure 2 shows the percentage of the design space (i.e. the number of evaluations divided by the number of points in the design space) explored by each algorithm in order to reach an ADRS of approximately 1% on average. Note that it was not possible to have all the algorithms converge to the exact same quality result, and some algorithms show high variability. In fact, SMOSA and NSGA do not often converge to acceptable solutions. The final accuracy values obtained are shown in Figure 3. Please note that the histograms and the error bars in Figures 2 to 6 show average values and standard deviations, respectively, for each algorithm over 30 experiments. No negative percentage was actually registered during the experiments

Figure 2 shows that the improvement can be worth the extra setup effort for class 3 and 4 algorithms: MDP, MOMDP and RESPIR have a factor 10 reduction in the number of evaluations, and a much tighter convergence (i.e. smaller variance of the results). Concerning class 2 algorithms, the performance is very similar, with IMMOGLS appearing to have the best combination of accuracy, number of evaluations and variance. APRS still provides excellent results given the zero-effort setup, although with at least twice as many evaluations when compared to class 2 algorithms.

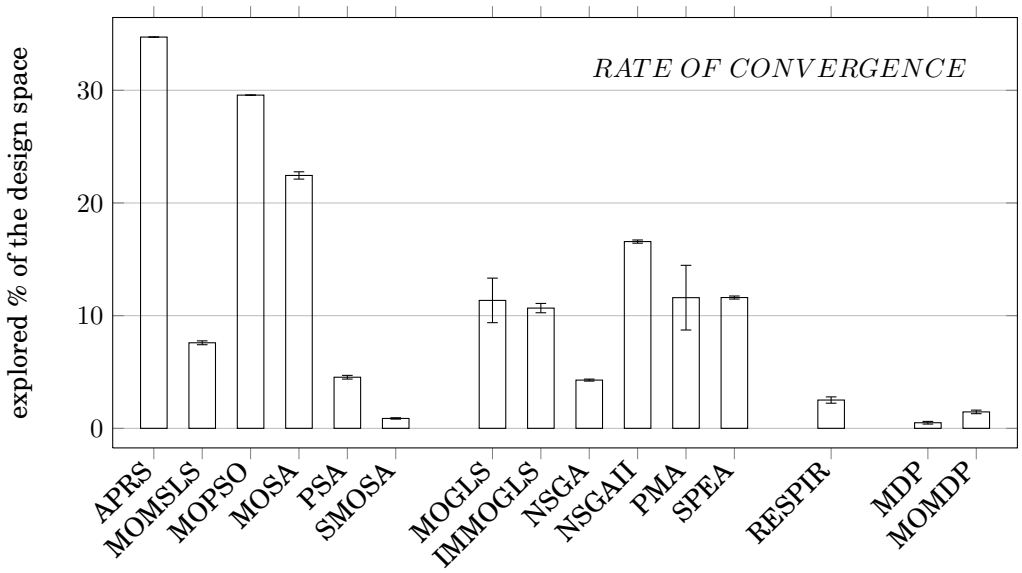

Fig. 2. The percentage of points in the design space evaluated by each algorithm for similar levels of accuracy (around 1%).

6.3. Characteristics of the Resulting Approximate Pareto-set

Figure 4 shows the number of Pareto points found by each algorithm, normalized by the average found for each benchmark to allow for global comparison.

Most algorithms appear to not have any statistically significant difference, with the exception of RESPIR, which finds 25% more points than all the other algorithms on average, but with a slightly higher variance. Once again, SMOSA and NSGA display the poorest results. It is worth noting that some of the points found by RESPIR are Pareto-covered by the points found by the other algorithms: the number of points on the actual Pareto curve is smaller than what found by RESPIR.

Concerning non-uniformity and concentration, all algorithms behave similarly, covering the design space fully and without concentrating on specific areas. We could not report any statistically significant difference between the algorithms, with the only exception of SMOSA and NSGA, which show worse results coupled with high variance. Results are presented in Figure 5 and Figure 6.

6.4. Scalability

The scalability column of Table XI refers to how the needed effort scales with the size of the design space: more scalable algorithms can be applied to more complex design spaces with lower effort. To test the effective scalability of each algorithm, we progressively increased the size of the design space, starting with three parameters and adding the remaining, three one by one. Similarly, we started with three values of each parameter and then increased their number progressively.

Most algorithms of classes 1 and 2 are very scalable: the number of parameters or the number of values per parameter do not affect the overall result, even though the number of additional evaluations needed grows proportionally with the number of parameters (i.e. remains around $\sim 10 - 15\%$ of the design space).

Although APRS require little setup effort, its applicability to large design spaces cannot be guaranteed. In fact, the already very high number of evaluations required

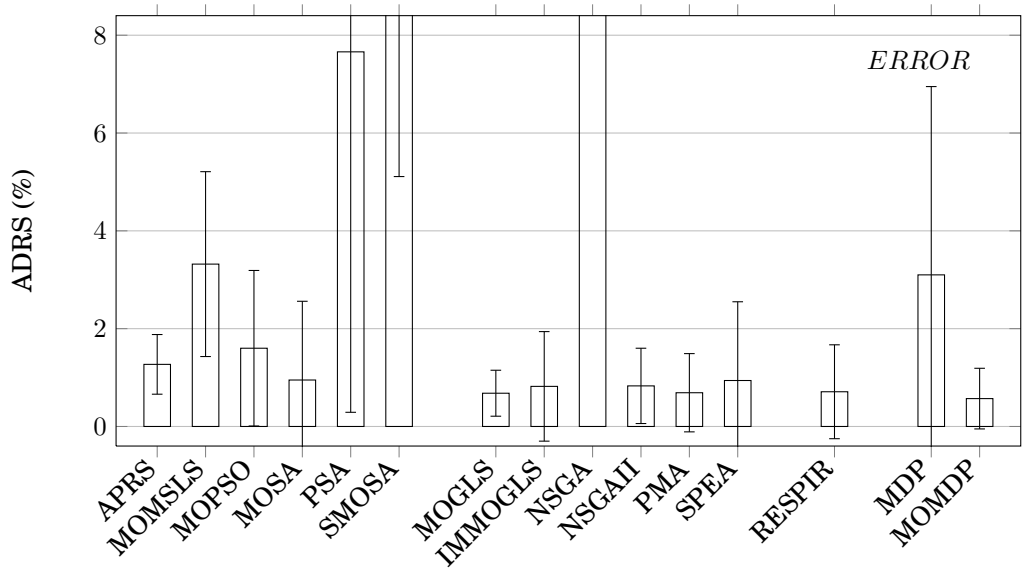

Fig. 3. Accuracy (ADRS) reached by each algorithm at convergence.

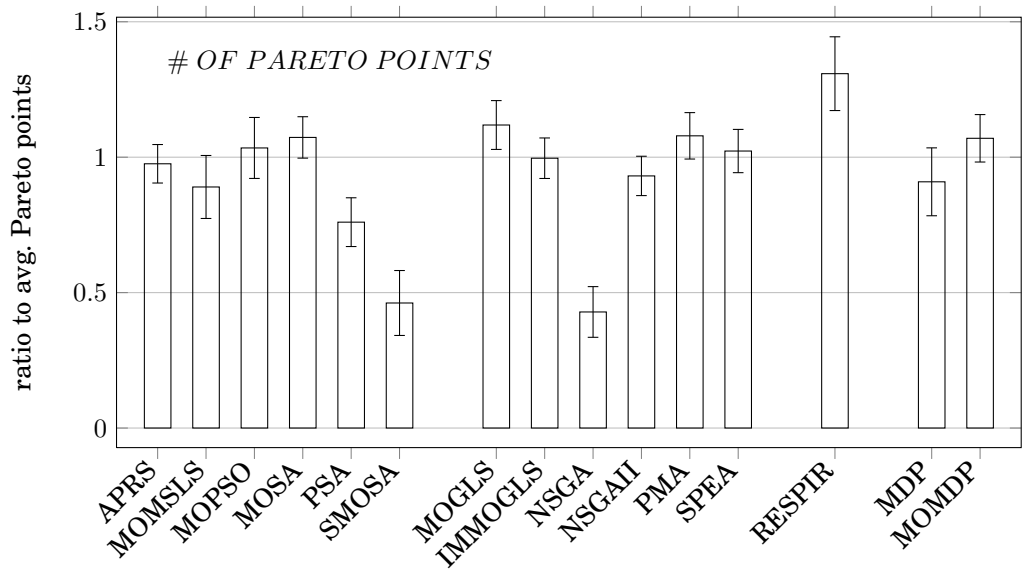

Fig. 4. The average number of points in the approximate Pareto set found by each algorithm, normalized with the average for each benchmark.

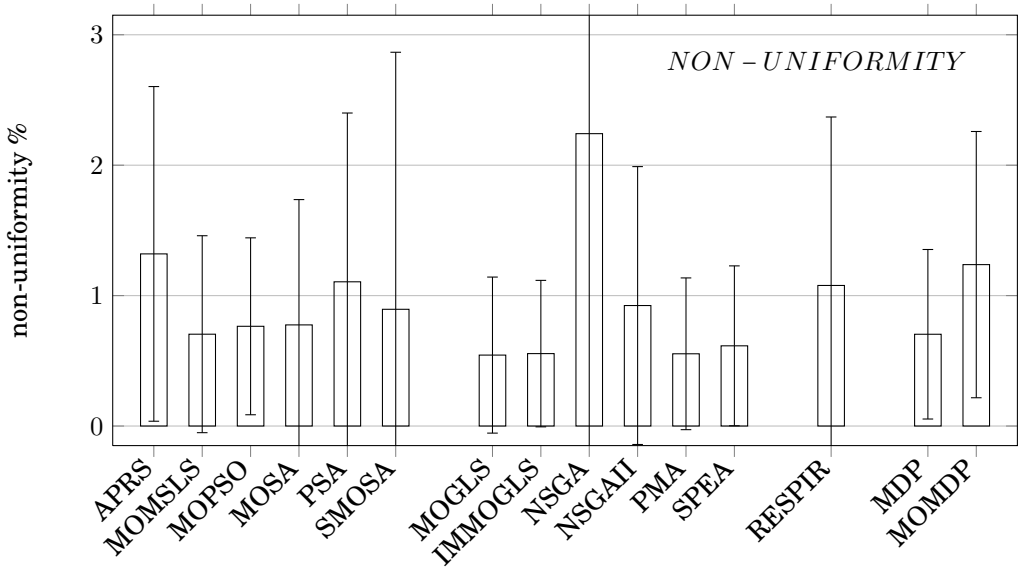

Fig. 5. The non-uniformity of the distribution of the points found in the approximate pareto set found by each algorithm.

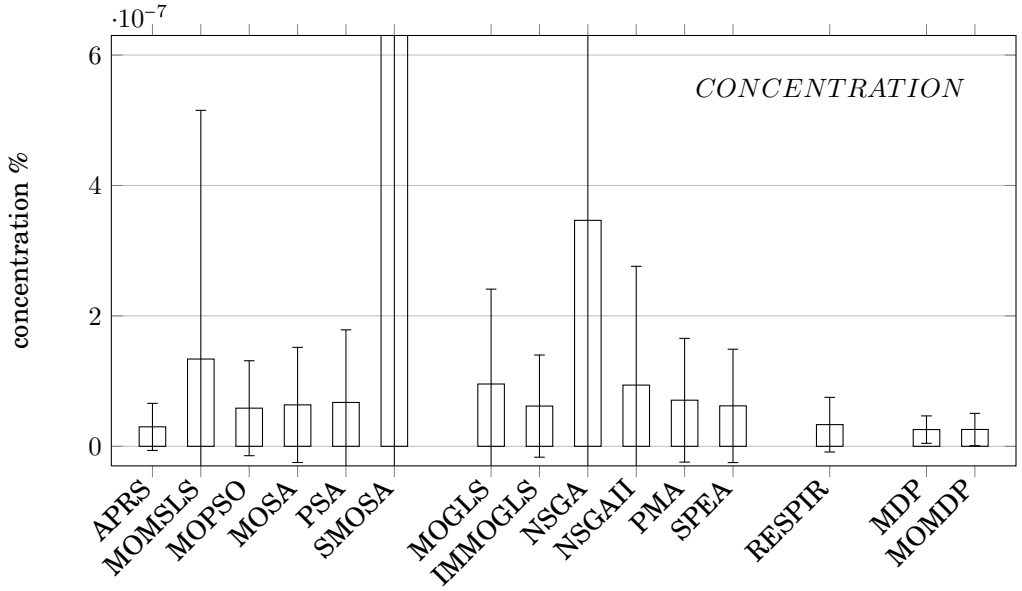

Fig. 6. The concentration of the points found in the approximate Pareto set found by each algorithm.

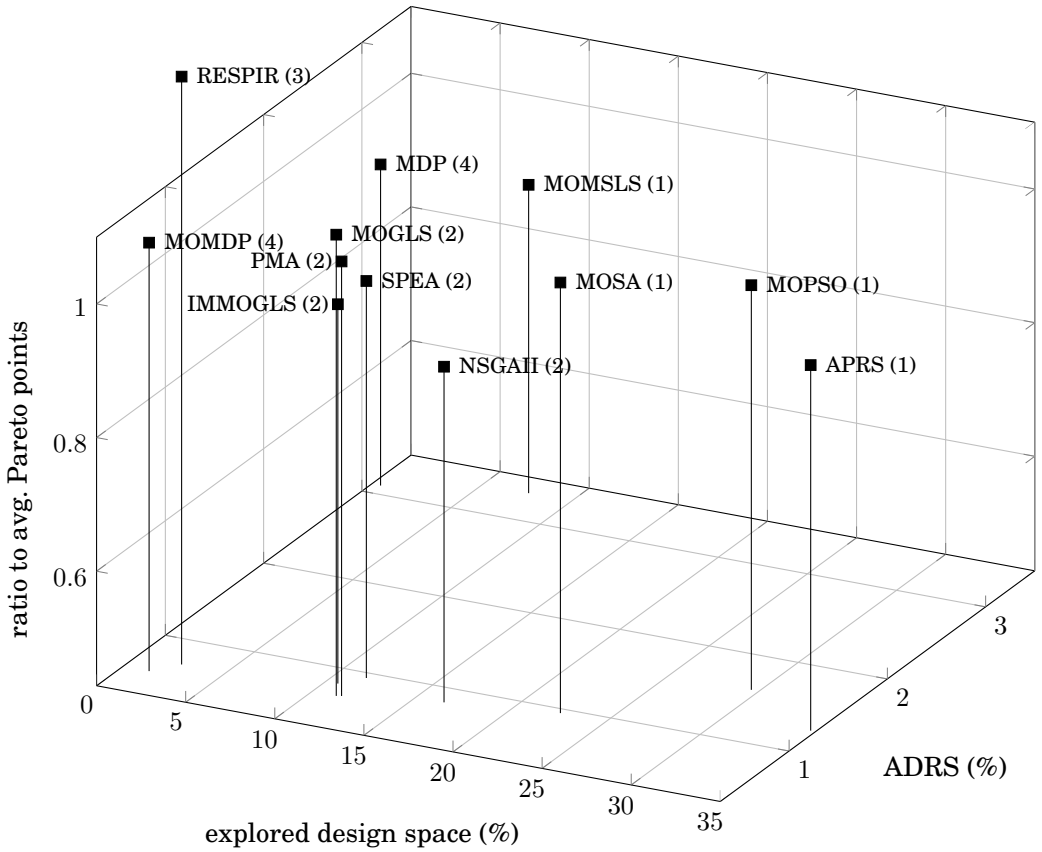

Fig. 7. 3D representation of the algorithms performance showing % of the design space explored in order to reach convergence, the number of Pareto points (w.r.t the average number) and the ADRS error metric (NSGA, PSA and SMOSA are omitted because of their large ADRS). The number after each algorithm name indicates the its class.

increases exponentially with the design space size, practically limiting its use to small design spaces with fast evaluations.

RESPIR (class 3) scales well to design spaces with many parameters, but only if few values per parameter are present. This is due to one of the main limitations of central composite design: it can only consider three levels for each parameter, therefore reducing the accuracy of the method in presence of many parameter values, especially if they lead to non-linear behaviour. The number of evaluations for convergence remains $\sim 4\%$ of the design space.

Finally, class 4 algorithms scale orthogonally with respect to class 3: they scale well with the number of values per parameter, but not when the number of parameters increase. While adding a parameter require defining new bounds, adding new values comes without effort, and the number of evaluations needed increases less than linearly [Beltrame et al. 2010]. One drawback of MDP when compared to MOMDP is that it requires the estimation of an additional parameter (α, see Beltrame et al. [2010]) when increasing the size of the design space, which might require additional evaluations.

7. DISCUSSION

The selection of the best algorithm for a particular application is difficult, and requires a trade-off between setup effort, scalability, expected number of simulations and expected accuracy. Given the experiments shown in Section 6, one can draw some general guidelines according to the cost of each evaluation (e.g. simulation time) and the size of the design space.

Higher evaluation costs make algorithms requiring a low number of simulations more appealing, even in case of a high upfront setup cost. On the contrary, if each evaluation has a very small cost, one might want to trade-off a higher number of evaluations with a no-effort setup. Similarly, large design spaces favor scalable algorithms, while smaller spaces do not justify the extra work to apply sophisticated algorithms.

In order to quantify the *actual* convergence time of an algorithm i we have to take into consideration the design space size ($|S|$ points), the time required by each simulation/evaluation (T_{sim} seconds), the percentage of the design space explored before reaching convergence (ν_i) and the set-up time $SETUP_i$ of the algorithm:

$$\text{ACTUAL}_i = (\nu_i \times |S| \times T_{sim}) + \text{SETUP}_i \tag{5}$$

The values of ν_i's are reported in Figure 2. Regarding the $SETUP_i$ values, we translated the qualitative information in Table XI into a 10'-to-30hrs range: we have observed during our experiments that algorithms having one "effort star" can be set-up in a few minutes while algorithms with five "effort stars" require more than a day of work.

Figure 8 presents four plots having the size of the design space on the x axis and the time required by each simulation on the y axis. In each plot, areas are labelled with the name of the more suitable algorithm according to Equation 5. Subplots (a) and (b) report the result for algorithms able to obtain ADRS of about 1%, whether domain knowledge is available (a), or not (b). Subplots (c) and (d) relax the ADRS requirement to about 5%.

Whether or not domain knowledge is available, Multi-Objective Multiple Start Local Search (MOMSLS) and Adaptive Windows Pareto Random Search (APRS) are the most appealing solutions for small to medium design spaces with non-expensive evaluations. As we explained in section 6, the APRS algorithm is a heuristic-based one and it requires very little setup effort, making it the ideal choice when simplicity is valued and there is no specific need for more efficient but also more complex approaches. MOMSLS is considerably faster but also more likely to produce a greater ADRS.

When domain knowledge is available, the Multi-Objective MDP algorithm (MOMDP) [Beltrame and Nicolescu 2011] clearly shows better performance, both in term of fast convergence to a very small ADRS and quality Pareto set, in large design spaces with high cost evaluations.

When one cannot obtain or exploit domain knowledge, the choice is split between the very high-quality Response Surface Pareto Iterative Refinement (RESPIR) algorithm and the Ishibuchi-Murata MO Genetic Local Search (IMMOGLS) algorithm. Both these algorithms are suited for very large design spaces.

However, it is worth noting that the IMMOGLS algorithm usually requires a larger number of evaluations, therefore is not recommended when dealing with high cost simulation. Pareto Simulated Annealing (PSA) is a valid alternative to IMMOGLS when a larger ADRS is acceptable.

What we can conclude from Figure 8 is that algorithms with small set-up times (i.e. the ones in classes 1 and 2) are especially suitable for simple problems with relatively small design spaces and/or short simulation times. On the other hand, complex algo-

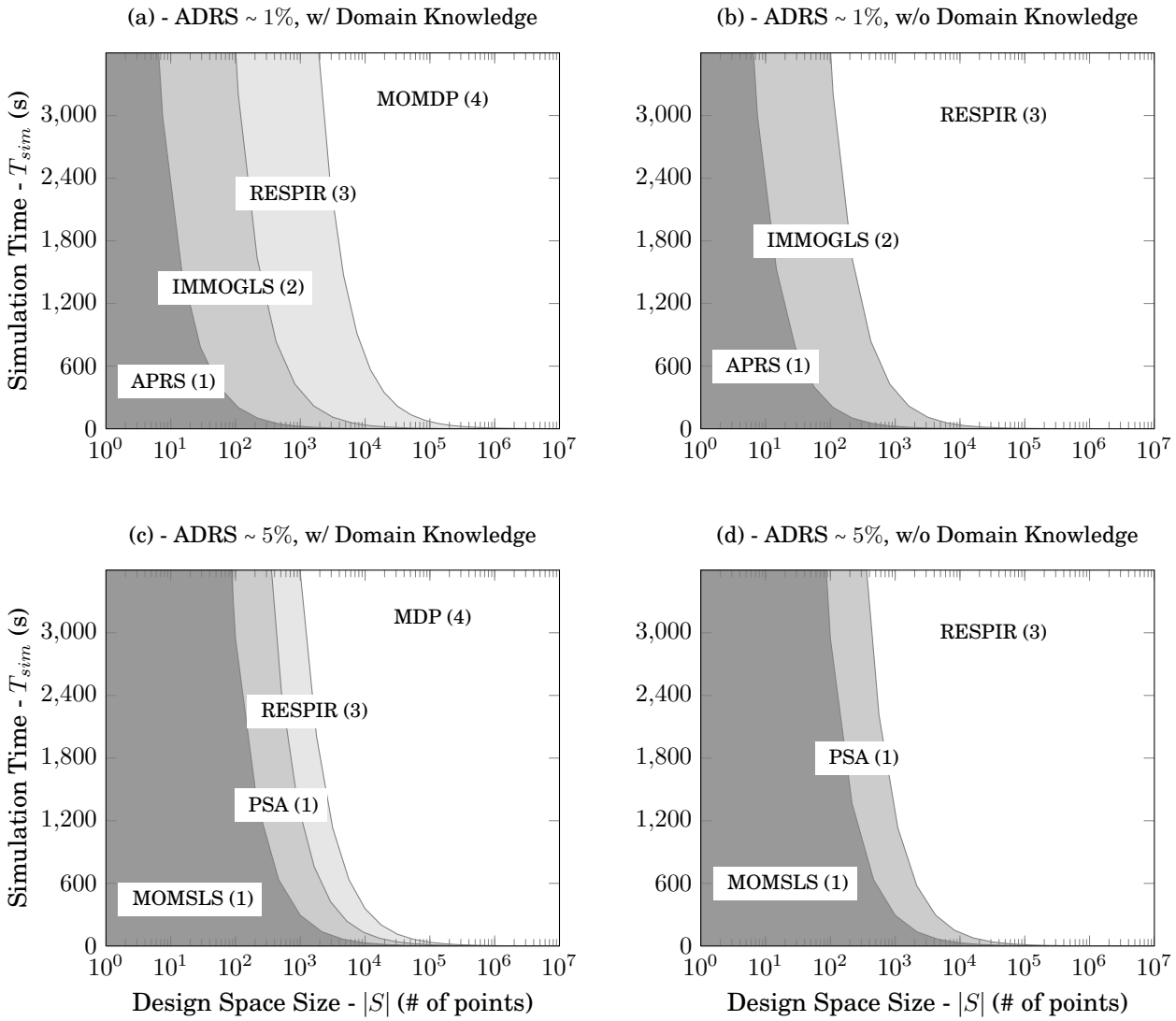

Fig. 8. Recommended algorithms for different design space size, simulation time and desired ADRS, whether domain knowledge is available or not. The number after each algorithm name indicates its class.

rithms in classes 3 and 4 always compensate for their longer configuration times when the exploration problem is difficult enough.

These recommendation are qualitative, but do take into account all the parameters shown in Section 6.

8. CONCLUSIONS

Concluding, this paper presented a classification and comparative analysis of fifteen of the best recent multi-objective design exploration algorithms. The algorithms were applied to the exploration of a multi-processor platform, and were compared for setup effort, number of evaluations, quality of the resulting approximate Pareto set, and scal-

ability. The results give guidelines on the choice of the proper algorithm according to the properties of the design space to be addressed. In particular, we have determined the most promising algorithms when considering design space size and evaluation effort.

REFERENCES

G. Beltrame, L. Fossati, and D. Sciuto. 2009. ReSP: A Nonintrusive Transaction-Level Reflective MPSoC Simulation Platform for Design Space Exploration. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 28, 12 (2009), 1857–1869.

G. Beltrame, L. Fossati, and D. Sciuto. 2010. Decision-Theoretic Design Space Exploration of Multiprocessor Platforms. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 29, 7 (2010), 1083–1095. DOI:http://dx.doi.org/10.1109/TCAD.2010.2049053

G. Beltrame and G. Nicolescu. 2011. A Multi-Objective Decision-Theoretic Exploration Algorithm for Platform-Based Design. In *Proc. of Design, Automation and Test in Europe (DATE)*. In press.

Carlos A. Coello. 2000. An updated survey of GA-based multiobjective optimization techniques. *ACM Comput. Surv.* 32, 2 (June 2000), 109–143. DOI:http://dx.doi.org/10.1145/358923.358929

Carlos A. Coello Coello. 1998. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems* 1 (1998), 269–308.

Piotr Czyzak and Adrezej Jaszkiewicz. 1998. Pareto simulated annealinga metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis* 7, 1 (1998), 34–47. DOI:http://dx.doi.org/10.1002/(SICI)1099-1360(199801)7:1⟨34::AID-MCDA161⟩3.0.CO;2-6

Kalyanmoy Deb and Tushar Goel. 2001. Controlled Elitist Non-dominated Sorting Genetic Algorithms for Better Convergence. In *Evolutionary Multi-Criterion Optimization*, Eckart Zitzler, Lothar Thiele, Kalyanmoy Deb, CarlosArtemio Coello Coello, and David Corne (Eds.). Lecture Notes in Computer Science, Vol. 1993. Springer Berlin Heidelberg, 67–81. DOI:http://dx.doi.org/10.1007/3-540-44719-9_5

Cagkan Erbas. 2006. *System-level Modelling and Design Space Exploration for Multiprocessor Embedded System-on-chip Architectures*. Amsterdam University Press. 156 pages.

Carlos M. Fonseca and Peter J. Fleming. 1995. An overview of evolutionary algorithms in multiobjective optimization. *Evol. Comput.* 3, 1 (March 1995), 1–16. DOI:http://dx.doi.org/10.1162/evco.1995.3.1.1

William Fornaciari, Donatella Sciuto, Cristina Silvano, and Vittorio Zaccaria. 2002. A Sensitivity-Based Design Space Exploration Methodology for Embedded Systems. *Design Automation for Embedded Systems* 7, 1 (2002), 7–33. DOI:http://dx.doi.org/{10.1023/A:1019791213967}

T. Givargis, F. Vahid, and J. Henkel. 2001. System-level exploration for Pareto-optimal configurations in parameterized systems-on-a-chip. In *Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on*. 25–30. DOI:http://dx.doi.org/{10.1109/ICCAD.2001.968593}

M.D. Hill and M.R. Marty. 2008. Amdahl's Law in the Multicore Era. *Computer* 41, 7 (2008), 33–38.

H. Ishibuchi and T. Murata. 1996. Multi-objective genetic local search algorithm. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*. 119–124. DOI:http://dx.doi.org/10.1109/ICEC.1996.542345

H. Ishibuchi and T. Murata. 1998. A multi-objective genetic local search algorithm and its application to flowshop scheduling. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 28, 3 (aug 1998), 392 –403. DOI:http://dx.doi.org/10.1109/5326.704576

Stanley Jaddoe and Andy D Pimentel. 2008. Signature-Based Calibration of Analytical System-Level Performance Models. In *Proceedings of the 8th international workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS '08)*. Springer-Verlag, Berlin, Heidelberg, 268278.

Andrzej Jaszkiewicz. 2004. A Comparative Study of Multiple-Objective Metaheuristics on the Bi-Objective Set Covering Problem and the Pareto Memetic Algorithm. *Annals of Operations Research* 131 (2004), 135–158. Issue 1-4. DOI:http://dx.doi.org/10.1023/B:ANOR.0000039516.50069.5b

Andrzej Jaszkiewicz and Grzegorz Dbrowski. 2005. MOMH: Multiple Objective Meta Heuristics. available at the web site http://home.gna.org/momh/. (2005).

Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. 1996. Reinforcement Learning: A Survey. *Journal of Articial Intelligence Research* 4 (1996), 237285. http://citeseer.ist.psu.edu/kaelbling96reinforcement.html

J. Kennedy and R. Eberhart. 1995. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, Vol. 4. 1942–1948 vol.4. DOI:http://dx.doi.org/10.1109/ICNN.1995.488968

Martin Lukasiewycz, Michael Glay, Christian Haubelt, and Jurgen Teich. 2008. Efficient symbolic multi-objective design space exploration. In *ASP-DAC '08: Proceedings of the 2008 Asia and South Pacific Design Automation Conference*. IEEE Computer Society Press, Seoul, Korea, 691–696.

Giovanni Mariani, Aleksandar Brankovic, Gianluca Palermo, Jovana Jovic, Vittorio Zaccaria, and Cristina Silvano. 2010. A correlation-based design space exploration methodology for multi-processor systems-on-chip. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*. 120–125.

R.T. Marler and J.S. Arora. 2004. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization* 26, 6 (2004), 369–395. DOI:http://dx.doi.org/10.1007/s00158-003-0368-6

G. Martin. 2006. Overview of the MPSoC design challenge. In *Design Automation Conference, 2006 43rd ACM/IEEE*. 274–279. DOI:http://dx.doi.org/{10.1109/DAC.2006.229245}

Kaisa Miettinen and Marko M. Mkel. 2002. On scalarizing functions in multiobjective optimization. *OR Spectrum* 24, 2 (May 2002), 193–213. DOI:http://dx.doi.org/10.1007/s00291-001-0092-9

S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. 2002. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. *SIGPLAN Not.* 37, 7 (2002), 18–27.

T. Okabe, Y. Jin, and B. Sendhoff. 2003. A critical survey of performance indices for multi-objective optimisation. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, Vol. 2. 878–885 Vol.2. DOI:http://dx.doi.org/10.1109/CEC.2003.1299759

G. Palermo, C. Silvano, and V. Zaccaria. 2008. Discrete Particle Swarm Optimization for Multi-objective Design Space Exploration. In *Digital System Design Architectures, Methods and Tools, 2008. DSD '08. 11th EUROMICRO Conference on*. 641–644. DOI:http://dx.doi.org/10.1109/DSD.2008.21

G. Palermo, C. Silvano, and V. Zaccaria. 2009. ReSPIR: A Response Surface-Based Pareto Iterative Refinement for Application-Specific Design Space Exploration. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 28, 12 (2009), 1816–1829. DOI:http://dx.doi.org/10.1109/TCAD.2009.2028681

Maurizio Palesi and Tony Givargis. 2002. Multi-objective design space exploration using genetic algorithms. In *CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign*. ACM, Estes Park, Colorado, 67–72. DOI:http://dx.doi.org/10.1145/774789.774804

P.G. Paulin and J.P. Knight. 1989. Algorithms for high-level synthesis. *Design & Test of Computers, IEEE* 6, 6 (1989), 18–31.

A.D. Pimentel, C. Erbas, and S. Polstra. 2006. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *Computers, IEEE Transactions on* 55, 2 (2006), 99–112. DOI:http://dx.doi.org/{10.1109/TC.2006.16}

Stuart J. Russell and Peter Norvig. 1995. *Artificial Intelligence: A Modern Approach* (1st ed.). Prentice Hall. 932 pages.

Paolo Serafini. 1994. Simulated Annealing for Multi Objective Optimization Problems. In *Multiple Criteria Decision Making*, G.H. Tzeng, H.F. Wang, U.P. Wen, and P.L. Yu (Eds.). Springer New York, 283–292. DOI:http://dx.doi.org/10.1007/978-1-4612-2666-6_29

D. Sheldon, F. Vahid, and S. Lonardi. 2007. Soft-core Processor Customization using the Design of Experiments Paradigm. In *DATE Conference, 2007*. 1–6.

S. N. Sivanandam and S. N. Deepa. 2007. *Introduction to Genetic Algorithms* (1st ed.). Springer Publishing Company, Incorporated.

N. Srinivas and Kalyanmoy Deb. 1994. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evol. Comput.* 2, 3 (Sept. 1994), 221–248. DOI:http://dx.doi.org/10.1162/evco.1994.2.3.221

Toktam Taghavi and Andy D. Pimentel. 2011. Design metrics and visualization techniques for analyzing the performance of MOEAs in DSE. In *ICSAMOS*. 67–76.

E.L. Ulungu, J. Teghem, P.H. Fortemps, and D. Tuyttens. 1999. MOSA method: a tool for solving multi-objective combinatorial optimization problems. *Journal of Multi-Criteria Decision Analysis* 8, 4 (1999), 221–236. DOI:http://dx.doi.org/10.1002/(SICI)1099-1360(199907)8:4⟨221::AID-MCDA247⟩3.0.CO;2-O

E. L. Ulungu and J. Teghem. 1994. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis* 3, 2 (1994), 83–104. DOI:http://dx.doi.org/10.1002/mcda.4020030204

Vittorio Zaccaria, Gianluca Palermo, F. Castro, Cristina Silvano, and Giovanni Mariani. 2010. Multicube Explorer: An Open Source Framework for Design Space Exploration of Chip Multi-Processors. In *2PARMA: Proceedings of the Workshop on Parallel Programming and Run-time Management Techniques for Many-core Architectures*. Hannover, Germany.

E. Zitzler, K. Deb, and L. Thiele. 1999a. *Comparison of Multiobjective Evolutionary Algorithms: Empirical Results (Revised Version)*. TIK Report 70. Computer Engineering and Networks Laboratory (TIK), ETH Zurich.

E. Zitzler, K. Deb, and L. Thiele. 1999b. Comparison of Multiobjective Evolutionary Algorithms on Test Functions of Different Difficulty. In *Genetic and Evolutionary Computation Conference (GECCO 1999): Bird-of-a-feather Workshop on Multi-criterion Optimization*.

E. Zitzler and L. Thiele. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *Evolutionary Computation, IEEE Transactions on* 3, 4 (nov 1999), 257 –271. DOI:http://dx.doi.org/10.1109/4235.797969

Received July 2013; revised October 2013; accepted December 2013