Optimization Strategies in Design Space Exploration

6

Jacopo Panerati, Donatella Sciuto, and Giovanni Beltrame

Abstract

This chapter presents guidelines to choose an appropriate exploration algorithm, based on the properties of the design space under consideration. The chapter describes and compares a selection of well-established multi-objective exploration algorithms for high-level design that appeared in recent scientific literature. These include heuristic, evolutionary, and statistical methods. The algorithms are divided into four sub-classes and compared by means of several metrics: their setup effort, convergence rate, scalability, and performance of the optimization. The common goal of these algorithms is the optimization of a multi-processor platform running a set of diverse software benchmark applications. Results show how the metrics can be related to the properties of a target design space (size, number of variables, and variable ranges) with a focus on accuracy, precision, and performance.

Acronyms

ADRS	Average Distance from Reference Set
ANN	Artificial Neural Network
DoE	Design of Experiments
DSE	Design Space Exploration
EA	Evolutionary Algorithm
GA	Genetic Algorithm
ILP	Integer Linear Program
MDP	Markov Decision Process
MPSoC	Multi-Processor System-on-Chip
NN	Neural Network

J. Panerati (✉) • G. Beltrame
Polytechnique Montréal, Montreal, QC, Canada
e-mail: jacopo.panerati@polymtl.ca; giovanni.beltrame@polymtl.ca

D. Sciuto
Politecnico di Milano, Milano, Italy
e-mail: donatella.sciuto@polimi.it

S. Ha, J. Teich (eds.), *Handbook of Hardware/Software Codesign*,
DOI 10.1007/978-94-017-7267-9_7

PSO	Particle Swarm Optimization
RSM	Response Surface Modeling
SA	Simulated Annealing
SoC	System-on-Chip

Contents

6.1 Introduction

Given a specific software application – or a class of software Applications – the parameters of a System-on-Chip (SoC) can be appropriately tuned to find the best trade-offs among the figures of merit (e.g., energy, area, and delay) deemed of interest by the designer. Design Space Exploration (DSE) is the tool by which the optimal *configuration* for a given system can be found.

This parameter tuning is fundamentally an optimization problem, which generally involves the maximization (or minimization) of multiple objectives. A consequence of having multiple objectives is that optimal solutions are, potentially, no longer unique. A set of *metrics* or *objective functions* are used to express the quality of a solution from different perspectives, also known as objectives. Since multi-objective optimization problems do not have a single optimal solution, solutions consist instead of several optima, i.e., the points that lie on the *Pareto curve* [11]. These are the optimal points that are non-dominated by any other point.

To find the Pareto curve for a specific platform, the designer has to evaluate all the possible configurations of the design space and characterize them in terms of objective functions. This approach is known as full or exhaustive search, and it is

often impractical due to the large number of points in a design space and/or due to the high cost associated with the evaluation of the objective functions (e.g., long simulation times).

Even today, Multi-Processor Systems-on-Chips (MPSoCs) platforms are often tuned according to designer experience or the non-systematic application of several algorithms found in literature. For example, classical heuristic algorithms (i.e., tabu search, simulated annealing, etc.) [28] are extremely common, as well as techniques able to reduce the design space size [22]. The advanced hybrid approaches in ▸ Chap. 7, "Hybrid Optimization Techniques for System-Level Design Space Exploration" combine metaheuristics and search algorithms to avoid large, unfeasible areas of the design space. All these approaches need to use simulation (or estimation) to assess the system-level metrics (i.e., the objective functions) of the very large number of configurations they evaluate. For a review of microarchitecture-level modeling and design methodologies for SoC, please refer to ▸ Chap. 27, "Microarchitecture-Level SoC Design".

The characteristics of a design space (e.g., its size or the time required to simulate one of its points) can certainly affect how different exploration algorithms perform in terms of time to convergence or accuracy of results. This chapter describes and compares 15 algorithms among the several ones that have been recently proposed in the scientific literature for automatic DSE and multi-objective optimization. These different approaches are dissimilar in terms of theoretical background, applicability conditions, and overall performance. Through rigorous analysis, the advantages and drawbacks of each method are uncovered, providing guidelines for their use in different contexts.

In the following, Sect. 6.2 describes a partitioning of the existing literature into four classes; Sect. 6.3 reviews the methodology of each one of the 15 algorithms that underwent experimental comparison; Sect. 6.4 presents the experimental setup, the framework used to compare these algorithms, and its results; Sect. 6.5 contains the discussion of these results as well as recommendations on the use of the algorithms; Sect. 6.6 lists some of the currently available open-source implementations of the algorithms; finally, Sect. 6.7 draws some concluding remarks.

6.2 Classification of Multi-objective DSE Strategies

The automation of DSE can be split into two sub-problems: (a) the identification of plausible candidate solutions (i.e., valid system configurations) and (b) the evaluation of the metrics of interest of these solutions, in order to select the optimal configurations.

Evolutionary Algorithms (EAs) in particular have widespread use in the area of design space exploration. EAs discriminate and select solutions using a combination of metrics called the fitness function. EAs are usually easy to apply and do not require detailed knowledge of the design space to be explored, and there is a strong theoretical background on how to assign fitness values for multi-objective problems [9].

To classify these methods, Coello [4] proposed to divide evolutionary approaches for multi-objective optimization into three sub-classes according to the way in which the fitness of individuals/solutions is computed: (1) algorithms using aggregating functions, (2) algorithms using non-aggregating but non Pareto-based approaches, and (3) Pareto-based approaches, such as the NSGA algorithm.

Broadening the scope of the analysis to also include design space exploration methods that are not based on EAs, four different classes of algorithms for problem (a) can be found in the literature:

- **Class 1: Heuristics and pseudo-random optimization approaches** attempt to reduce the design space under scrutiny and focus the exploration on regions of interest [10, 11]. Methodologies in this class often rely on full search or pseudo-random algorithms to explore the selected regions. Class 1 includes algorithms such as multi-agent optimization (e.g., Particle Swarm Optimization (PSO) [26]), Simulated Annealing (SA), tabu search, and operations research algorithms. Methodologies in this class often aim at drastically reducing the number of configurations to evaluate (e.g., from the product to the sum of the number of tunable parameters in [10]). One way to do so is, for example, to identify sub-spaces (called clusters) into the design search space that can be efficiently explored in an exhaustive fashion [11]. The global Pareto front can then be reconstructed from the Pareto-optimal configurations of each partition. Because of their simplicity, the exploration results of the algorithms in this class remain in many cases sub-optimal.
- **Class 2: Evolutionary algorithms.** EAs are the most common and widely used DSE algorithms, they apply random changes of a starting set of configurations to iteratively improve their Pareto set of solutions. Genetic Algorithms (GAs) belong to this category [28]. The major advantages of these algorithms are the very limited setup effort and the fact that they do not require any specific knowledge associated to the search space or the metrics used for the optimization. Unfortunately, these algorithms are not guaranteed to find the optimal solutions or even to converge to results within certain predefined quality bounds. In practice, however, they usually perform fairly well when they are allowed to run for a sufficiently large number of evaluations. Techniques in this class can also be combined with exact methods. For example, in [20], DSE is translated into a multi-objective 0–1 Integer Linear Program (ILP) problem and a pseudo-Boolean (PB) solver is used to constrain a GA within the feasible search space.
- **Class 3: Statistical approaches without domain knowledge.** Methodologies in this class extract a *metamodel* from the design space and use it to predict the next configurations to evaluate [20, 21, 27, 32]. Class 3 includes those methods that use statistics to guide their DSE. Many of the algorithms in this class exploit a methodology called *Design of Experiments (DoE)* [21, 27, 32] to characterize the sensitivity of the system to its parameters. DoE, in fact, allows to estimate the portion of the variance of each objective metric associated to the oscillations in a certain parameter. Heuristics or metamodels are then developed from this sensitivity analysis and used to tune the parameters and find the optimal configurations of the system. The work in [27] is an example of this

sort of approach. It uses DoE to define an initial set of experiments and create a preliminary estimate of the target design space. Then, the exploration is further refined iteratively using a technique called Response Surface Modeling (RSM). Statistical methods, such as DoE, are the defining characteristics of this class, as they allow to extract the maximum amount of information from the initial training sets of limited size. The generated metamodels can then be used both to find new candidate configurations as well as to evaluate them.

- **Class 4: Statistical approaches with domain knowledge.** These are techniques that use predefined rules and knowledge specific to a certain design space to find the most promising solutions [3]. The algorithms in Class 4 are characterized by the use of built-in domain knowledge to set up a probabilistic framework to guide the identification of new candidate solutions. The work in [3], in particular, combines decision theory with this kind of integrated knowledge. The DSE problem is remapped to a Markov Decision Process (MDP) [18, 30] whose solution is the sequence of changes to apply to the tunable parameters in order to minimize (or maximize) one of the objective performance metrics. The major advantage of this approach is that it greatly reduces the number of times in which the system requires to be simulated (i.e., only when uncertainty at a fork is too large to be managed with the embedded domain knowledge).

Later in this chapter, a selection of well-established algorithms from all the four classes and their configuration parameters are described. Then, their strengths and weaknesses are compared in the context of the optimization of a symmetric multi-processor platform. The results of this experiment provide guidelines for the selection of the right DSE and multi-objective optimization algorithm, given the characteristics of the design space.

The evaluation of the candidate solutions – problem (b) – can be tackled either using detailed simulation [3] or simpler predictive models [14], or also a combination of the two.

Okabe, Jin, and Sendhoff [25] reviewed several metrics for the evaluation of solutions to multi-objective optimization problems. Their conclusion is that no single metric is usually sufficient to quantify the quality of the Pareto set returned by a multi-objective optimization algorithm. Similar results are also found in [35].

In our analysis, four different metrics are used to evaluate the accuracy, distribution, and cardinality of the Pareto sets found by the optimization algorithms. Moreover, the performance of the algorithms under study is quantified as the number of evaluations needed to converge to a Pareto set.

6.3 Multi-objective DSE Algorithms

This section details 15 multi-objective optimization algorithms for DSE that are then experimentally compared in Sect. 6.4. For each algorithm, a brief description of inner working and its parameters are given. The reader should pay attention to the fact that the parameters mentioned here are not the tunable parameters of a computing platform (i.e., those that create the design space). The parameters

described in this section are the configuration parameters of each algorithm and they are important as they influence its performance and setup effort.

6.3.1 Heuristics and Pseudo-random Optimization Approaches

This section describes algorithms belonging to the first class defined in Sect. 6.2 in detail. Half of the algorithms described in this section (MOSA, PSA, and SMOSA) are based on SA. SA is essentially a local search characterized by the way in which local minima are avoided: solutions that do not improve the current one can be accepted with a given probability p, computed from a Boltzmann distribution (parameterized by a coefficient T, called temperature).

6.3.1.1 Adaptive Windows Pareto Random Search (APRS)

The APRS algorithm is one of two novel algorithms implemented in the Multicube Explorer framework [37]. APRS takes an initial set of design points as its approximate Pareto set, then it improves this solution by randomly picking new points from windows centered on the current points. The size of the windows is reduced according to two factors: time (i.e., number of iterations) and quality of the points they are created from.

6.3.1.2 Multi-objective Multiple Start Local Search (MOMSLS)

MOMSLS [16] is a heuristic method based on executing in parallel multiple local searches starting each one of them from a different initial point. Each local search itself is a simpler heuristic that iteratively refines an initial random solution by looking for better candidates in its neighborhood [30]. As a consequence, MOMSLS evaluates solutions in N different neighborhoods during each one of its iteration steps.

6.3.1.3 Multi-objective Particle Swarm Optimization (MOPSO)

PSO [19] is a biologically inspired heuristic search method that mimics the movement of a flock of birds. Candidate solutions are described by particles of the swarm, and, at each iteration, they move a maximum of the objective function updating their position according to a velocity vector (see Fig. 6.1) that is the linear combination of three components:

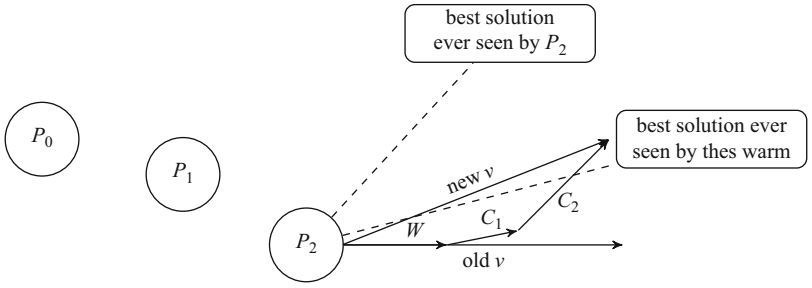

Fig. 6.1 Example of a velocity vector update for a particle belonging to a three-particle swarm

- their last velocity vector (weighted by an inertial factor, W);
- the direction toward the best (e.g., with the greatest objective metric) solution ever reached by the swarm (weighted by a social learning factor, C_1);
- the direction toward the best solution ever reached by the individual particle (weighted by a cognitive learning factor, C_2).

To adapt PSO to multi-objective optimization problems, MOPSO [26] replicates the PSO approach in N different swarms. These N swarms have, for objective function, the product of the multiple objective functions, each elevated by a random exponent. Moreover, in MOPSO, the inertial and social learning factors are set to 0. Particles are also forced to perform a random walk with a fixed probability p to avoid local minima.

In MOPSO implementations where the social learning factor is different from 0, the overall performance is also influenced by the way in which the swarm "leaders" are selected. The analysis performed in [23] shows that choosing as leaders those particles/solutions that contribute the most to the hyper-volume of the Pareto front usually results in the best performance.

6.3.1.4 Multi-objective Simulated Annealing (MOSA)

MOSA [36] is the multi-objective adaptation of the well-known SA technique. In [36], there are two proposed approaches for the translation of SA into the context of multi-objective optimization:

- probability scalarization, i.e., computing the acceptance/rejection probability of new solutions for each performance metric and their aggregation;
- criterion scalarization, i.e., the projection of the performance metrics into a single metric that is then used to compute the acceptance/rejection probability of the new solution.

The approach actually implemented in MOSA [36] and tested in Sect. 6.4 is the second one, while the first one is used by the two other SA-based algorithms presented in the following.

6.3.1.5 Pareto Simulated Annealing (PSA)

PSA [5] proposes two different criteria to scalarize the acceptance/rejection probability of a new solution and adapt simulated annealing to multi-objective problems:

- rule C says that the rejection probability p is proportional to the largest difference between the current and the new solution among all the performance metrics under evaluation;
- rule SL, instead, states that p is the weighted linear combination of the differences between the current and the new solution in all their performance metrics.

The weights used by rule SL to multiply each metric are also tuned at each iteration, depending on whether the most recently introduced solution brought a deterioration in that specific metric or not.

6.3.1.6 Serafini's Multiple Objective Simulated Annealing (SMOSA)

The work in [31] enriches the discussion on the rules that can be used to combine multiple performance metrics and apply SA to multi-objective optimization problems. In [31], a new complex rule is defined as the linear composition, with coefficients α and $(1-\alpha)$, of two simpler rules:

- rule P, i.e., the rejection probability p of a new solution is proportional to the product of the differences between the current and the new solution in all their performance metrics;
- and rule W, i.e., p is proportional to the smallest value among the differences between the current and the new solution in all their performance metrics.

6.3.2 Evolutionary Algorithms

This section describes algorithms belonging to the second class defined in Sect. 6.2 in detail. The majority of these approaches is composed by variations of traditional genetic algorithms. The basic way a GA works is by improving an initial set of solutions (often randomly chosen), called population, by computing new solutions as combinations of existing solutions X picked with a probability p proportional to their fitness $f(X)$ [33].

6.3.2.1 Multiple Objective Genetic Local Search (MOGLS)

MOGLS [12] combines a typical class 2 methodology with one from class 1, that is, genetic algorithms with a local search. Those algorithms that combine multiple search methodologies are often referred to as "hybrid approaches." Each fundamental iteration step of MOGLS is composed of two sub-steps:

- first, new solutions are generated using genetic operations;
- and then, a local search is performed in the neighborhoods of these new solutions.

6.3.2.2 Ishibuchi-Murata Multi-objective Genetic Local Search (IMMOGLS)

IMMOGLS [13] is another hybrid algorithm that combines a genetic algorithm and a local search, just like MOGLS. In a multi-objective minimization problem, a solution is said to be non-dominated with respect to a set of solutions, if none of the other solutions score lower in all of the metrics one wants to minimize (or maximize). The iteration step of IMMOGLS, as for MOGLS, consists of both genetic operations and a local search but with three peculiar aspects:

- the fitness function used by the GA is a linear combination of the optimization metrics and the weights are chosen randomly at each iteration;
- the local search is limited to a certain (random) number of neighbors k;
- at each iteration, the current population is purged of any dominated solutions (this is referred to as an "elitist strategy").

6.3.2.3 Non-dominated Sorting Genetic Algorithm (NSGA)

NSGA [34] is a very successful application of the genetic approach to the problem of multi-objective optimization. The main insight regarding NSGA is the way in which the fitness of solutions is computed. All the individual in the current population are assigned fitness values on the basis of non-domination. All non-dominated solutions are assigned the same fitness value.

6.3.2.4 Controlled Non-dominated Sorting Genetic Algorithm (NSGA-II)

In [6], NSGA-II – an evolution of NSGA – is introduced. NSGA-II has two main peculiar aspects:

- NSGA-II is an elite-preserving algorithm; this means that non-dominated solutions cannot ever be removed from the current population;
- the sorting of solutions by non-domination also reduces computational complexity.

Genetic algorithms rely on several different genetic operators (mutation, crossover, etc.) to create new solutions as shown in Fig. 6.2. Thanks to GAs robustness, even the random selection of these operators usually leads to quality performance [24].

6.3.2.5 Pareto Memetic Algorithm (PMA)

PMA [15] is another member of the family of hybrid algorithms that combines GAs local searches. PMA differs from MOGLS and IMMOGLS in the way it selects the solutions for the crossover genetic operation. These are not drawn from the current population but from another set of size T that is the results of sampling

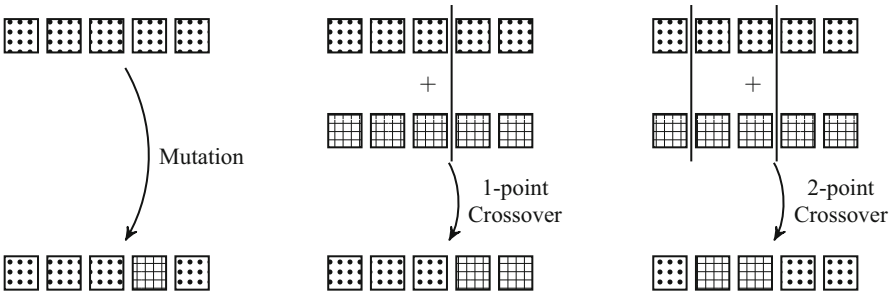

Fig. 6.2 Three of the most commonly used genetic operators: mutation, one-point, and two-point crossover

(with repetition) from the current population. The solutions that are chosen for recombination are the two with the best fitness in this new set.

6.3.2.6 Strength Pareto Evolutionary Algorithm (SPEA)

SPEA [38] is part of the larger family of heuristic search methods called evolutionary algorithms that also includes GAs. The three characteristic traits of SPEA are:

- all the non-dominated solutions are also stored in an auxiliary set/population;
- the fitness of a solution in the current population is determined only by comparison with the solutions in this auxiliary set [38];
- clustering is used to limit the size of the auxiliary population.

6.3.3 Statistical Approaches Without Domain Knowledge

This section describes algorithms belonging to the third class defined in Sect. 6.2 in detail.

6.3.3.1 Response Surface Pareto Iterative Refinement (ReSPIR)

ReSPIR [27] is a DSE tool that exploits two statistical/learning methodologies to infer the relationships between the tunable parameters of a system and its performance metrics. The benefit of this approach is that the number of evaluations needed to optimize a design is greatly reduced. These two pillars of ReSPIR are:

- DoE, a methodology that maximizes the information gained from a set of empirical trials;
- and RSMs, analytical representations of a performance metric reconstructed from the data. Several models can be applied for this problem: linear regression, Shepard-based interpolation, Artificial Neural Networks (ANNs), etc.

The main iteration step of ReSPIR consists of using DoE to define a set of experiments to perform, training the RSMs with the information collected, and finally producing an intermediate Pareto set.

6.3.4 Statistical Approaches with Domain Knowledge

This section describes algorithms belonging to the fourth class defined in Sect. 6.2 in detail.

6.3.4.1 Markov Decision Process Optimization (MDP)

The approach proposed by [3] is based on a framework for sequential decision making called Markov decision process. The components of an MDP are states, actions, stochastic transitions from state to state, and rewards (Fig. 6.3). The solution

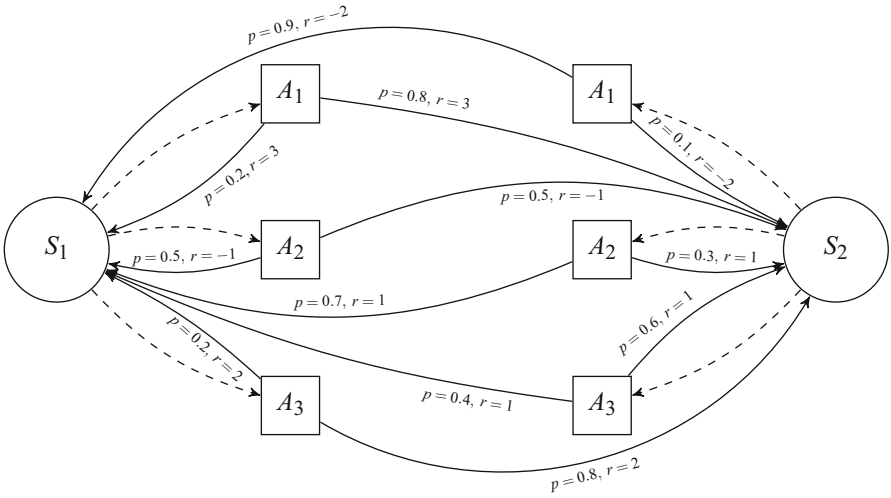

Fig. 6.3 An MDP with two states and three actions. Each arc going from an action A_x to a state S_y is associated with a probability p and a reward r

of an MDP is a strategy, i.e., the correct actions to perform in each one of the states to collect the largest amount of rewards [30] in a certain time horizon. In the formulation of [3], states are points in the design space (with their relative performance metrics), actions are changes in the tunable parameters, and rewards are improvements in the performance metrics. Stochastic transitions are initialized with uniform distribution and their likelihood are then refined throughout the execution of the algorithm. The main drawback of this approach is that it requires 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 setup time and cannot be used without thorough knowledge of the system to optimize.

6.3.4.2 Multi-objective Markov Decision Process (MOMDP)

MOMDP [1] is an improved version of MDP. The main difference lies in MOMDP novel exploration strategy. MDP, in fact, aggregated multiple performance metrics with a scalarizing function. By changing the value taken from a parameter called α (representing the weight(s) associated to each metric), it was possible to discover a Pareto curve for multiple separate objectives.

MOMDP, instead, 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.

6.4 Experimental Comparison

Most class 1 and class 2 algorithm implementations can be found in the Multiple Objective MetaHeuristics Library in C++ (MOMHLib++) [16]. Multicube Explorer (M3Explorer) [37] also 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, they are evaluated using the original source code.

6.4.1 Objectives

This experimental comparison has multiple aims. Its three most important aspects are:

- determine the effort required to configure each algorithm for a given design space and how the characteristics of the design space influence the choice of the most effective exploration algorithm;
- determine the number of evaluations required by each algorithm to obtain an approximate Pareto set that meets certain quality requirements;
- and, finally, quantify the quality of the resulting Pareto set found by each algorithm.

A qualitative comparison of the 15 algorithms is presented in Table 6.3. Each algorithm was tested in the context of the same design space: a symmetric multi-processor platform running three different applications, shown in Fig. 6.4.

6.4.2 Benchmark Applications

Three applications (listed in Table 6.1) were used for testing, more specifically, two large applications and a small benchmark, for which exhaustive search was possible. *ffmpeg*, 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 six-step FFT algorithm (*fft6*). All applications are data-parallel and are targeted toward 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 chosen in order to display the maximum variability in their behavior, as they use a different synchronization mechanisms and require very different evaluation times. For situations in which it is especially important to identify the optimal mapping of complex, multi-application workloads,

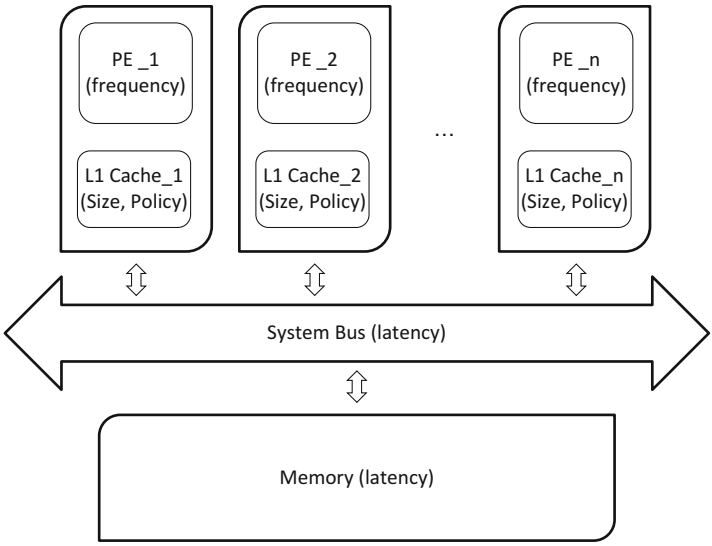

Fig. 6.4 The simulated multi-core processor architecture and its parameters (From [29])

Table 6.1 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	∼2 m	A parallel implementation of gzip
fft6	2.0	C/Fortran77	OpenMP	Barrier	∼30 s	Implementation of Bailey's 6-step fast Fourier transformation algorithm
ffmpeg	49.0.2	C	pthreads	semaphore	∼30 m	A fast video and audio converter

▸ Chap. 9, "Scenario-Based Design Space Exploration" explores thoroughly the domain of scenario-based DSE.

6.4.3 Target Computing Platform

The target platform consists of a collection of ARM9 cores with private caches and a shared memory (a comprehensive review of memory architectures and organizations can be found in ▸ Chap. 13, "Memory Architectures") interconnected by a simple system-bus model. Cache coherence is directory-based and implements the MESI protocol. The number of processing elements varies between one and eight. In the context of heterogeneous (e.g., big.LITTLE) multi-processor architectures, ▸ Chap. 8, "Architecture and Cross-Layer Design Space Exploration"

Table 6.2 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}

examines how cross-layer optimization and predictive models can further enhance the DSE process.

The ReSP [2] open-source simulation environment was used to perform the simulations, providing a set of configurable parameters, listed in Table 6.2. ReSP provides values for execution time and power consumption, which were used as the performance metrics for all optimization algorithms in our experiments. For more on energy optimization, power, and thermal constraints in DSE, please refer to the methodologies in ▸ Chap. 10, "Design Space Exploration and Run-Time Adaptation for Multicore Resource Management Under Performance and Power Constraints".

The platform was explored using the parameters listed in Table 6.2 with a resulting design space of 8640 points (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.), comparable with similar works (e.g., 6144 points in [32]) and such that 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 three benchmark applications was optimized ten times with each exploration algorithm ($N = 30$ executions for each algorithm).

6.4.4 Metrics to Evaluate Approximate Pareto Sets

According to [35], the quality of the result of a multi-objective optimization algorithm is twofold: (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.

Because of this reason, three metrics presented in [8] are used to compare the relative quality of the approximate Pareto sets obtained by the 15 algorithms under evaluation:

6.4.4.1 Average Distance from Reference Set

The Average Distance from Reference Set (ADRS) is used to compare the approximated Pareto sets with the best Pareto set (found combining the results of all

experiments). ADRS approximates the distance of the set under scrutiny from the Pareto-optimal front and should be minimized: an algorithm with low ADRS is very likely to have found a Pareto set that resembles the actual one.

As defined in [37], the ADRS between an approximate Pareto set Λ and a reference Pareto set Π is computed as:

$$\text{ADRS}(\Pi, \Lambda) = \frac{1}{|\Pi|} \sum_{-a \in \Pi} \left(\min_{-b \in \Lambda} \{\delta(-b, -a)\} \right) \tag{6.1}$$

where the δ function stands for:

$$\delta(-b, -a) = \max_{j=1,..,m} \left\{ 0, \frac{\phi_j(-a) - \phi_j(-b)}{\phi_j(-b)} \right\} \tag{6.2}$$

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

6.4.4.2 Non-uniformity

Non-uniformity measures how solutions are distributed in the design space. A good search algorithm should look at all the different regions of a design space with equal attention. 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 two consecutive points ($i = 1, .., |\bar{\Lambda}| - 1$), and $\hat{d}$ is the average value of all the d's, non-uniformity [8] can be computed as:

$$\sum_{i=1}^{|\bar{\Lambda}|-1} \frac{|d_i - \hat{d}|}{\sqrt{m}(|\bar{\Lambda}| - 1)} \tag{6.3}$$

6.4.4.3 Concentration

Concentration measures 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. It is important to observe that non-uniformity captures the behavior of a search algorithm in the design space, while concentration looks at points in the space of objective metrics.

Given a normalized Pareto set $\bar{\Lambda}$, where $\phi_i^{\min}$ is defined as $\min\{\phi_i(-a)\ s.t.\ -a \in \bar{\Lambda}\}$ and $\phi_i^{\max}$ is defined as $\max\{\phi_i(-a)\ s.t.\ -a \in \bar{\Lambda}\}$, concentration [8] can be computed as:

$$\prod_{i=1}^{m} \frac{1}{|\phi_i^{\max} - \phi_i^{\min}|} \tag{6.4}$$

6.4.5 Performance of the Algorithms Under Test

Having established the experimental context – that is, a set of benchmark applications and the target computing platform – and the performance metrics necessary to evaluate approximate Pareto sets, this section details the obtained results.

6.4.5.1 Initial Setup Effort and Parameter Sensitivity

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 perform. There is no common rule for the choice of each algorithm's parameters: these range from 4 to 12, and they can be anything from integers to the choice of an interpolation function. The selection of the parameters that are optimal for a specific optimization problem requires either human expertise or the meta-exploration (also known as parameter screening) of the algorithm parameters 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 6.3 qualitatively presents the tuning cost required by each algorithm.

Algorithms of classes 1 and 2 only require few parameters (such as population size, mutation factors, initial temperature, etc.), and they 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 (e.g., initial temperature for MOSA and NSGA-II) have no direct link to any characteristic of the design space and can be determined only by experience or guesswork, rendering the best

Table 6.3 A qualitative analysis of the chosen algorithms: setup effort, number of evaluations for 1% ADRS, number of Pareto points found, and scalability

Acronym	Class	Setup effort	Evaluations	Pareto points	Scalability
APRS [37]	1	⋆	⋆ ⋆ ⋆ ⋆ ⋆	⋆⋆	⋆
MOMSLS [16]	1	⋆	⋆ ⋆ ⋆	⋆	⋆ ⋆ ⋆
MOPSO [26]	1	⋆⋆	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆	⋆ ⋆ ⋆⋆
MOSA [36]	1	⋆ ⋆ ⋆	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆	⋆ ⋆ ⋆⋆
PSA [5]	1	⋆ ⋆ ⋆	⋆ ⋆ ⋆	⋆	⋆ ⋆ ⋆⋆
SMOSA [31]	1	⋆ ⋆ ⋆	⋆	⋆	⋆ ⋆ ⋆⋆
MOGLS [12]	2	⋆⋆	⋆ ⋆ ⋆	⋆ ⋆ ⋆	⋆ ⋆ ⋆⋆
IMMOGLS [13]	2	⋆⋆	⋆ ⋆ ⋆	⋆⋆	⋆ ⋆ ⋆⋆
NSGA [34]	2	⋆ ⋆ ⋆	⋆ ⋆ ⋆	⋆	⋆ ⋆ ⋆⋆
NSGA-II [6]	2	⋆ ⋆ ⋆	⋆ ⋆ ⋆⋆	⋆	⋆ ⋆ ⋆⋆
PMA [15]	2	⋆⋆	⋆ ⋆ ⋆	⋆⋆	⋆ ⋆ ⋆⋆
SPEA [38]	2	⋆⋆	⋆ ⋆ ⋆	⋆ ⋆ ⋆	⋆ ⋆ ⋆⋆
ReSPIR [27]	3	⋆ ⋆ ⋆⋆	⋆⋆	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆
MDP [3]	4	⋆ ⋆ ⋆ ⋆ ⋆	⋆	⋆	⋆⋆
MOMDP [1]	4	⋆ ⋆ ⋆ ⋆ ⋆	⋆	⋆ ⋆ ⋆	⋆ ⋆ ⋆

combination very difficult to obtain without screening and additional evaluations. For the comparison in this chapter, the best parameters were determined via screening, which required running several thousand evaluations.

Algorithms like APRS have a minimal setup effort as they do not require any special tuning and rely on pre-determined heuristics. It is worth noting that both class 1 and 2 algorithms do not guarantee convergence to the optimal Pareto set and require the user to 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 in class 3 demand a greater setup effort: the choice of proper metamodels for the design space requires some expertise and an initial screening (i.e., 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 the comparison in this chapter, the central composite design using a Neural Network (NN) interpolator was used, as suggested by the results in [27]. However, the NN produced results with a very high variance, with ADRS ranging from 0 to 160%, making the use of this interpolator impractical. The Shepard interpolation was found to be 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. The results of [27] were effectively replicated in the considered design space using a power of 16, which was found by parameter screening ($\sim 10^3$ evaluations).

Finally, algorithms in class 4 require to set bounds to the effects of parameter variations on the configuration's metrics. The quantification of these bounds is left to the designer's experience or can be determined via statistical modeling. This means detailed analysis of each design space and a large setup effort. The main difference between MDP 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 the design space under evaluation, hundreds of additional evaluations are required. MOMDP does not require this screening phase, 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.

6.4.5.2 Convergence Rate

For the comparison in this chapter, each algorithm's parameters were optimally tuned but the evaluations needed for screening and initial parameter estimation are **not** accounted for. Concerning ADRS, since exhaustive search is not possible, the reference Pareto set is the best Pareto set generated compounding all evaluations performed by all algorithms, which covered a sizable portion of the entire design space (around 30%).

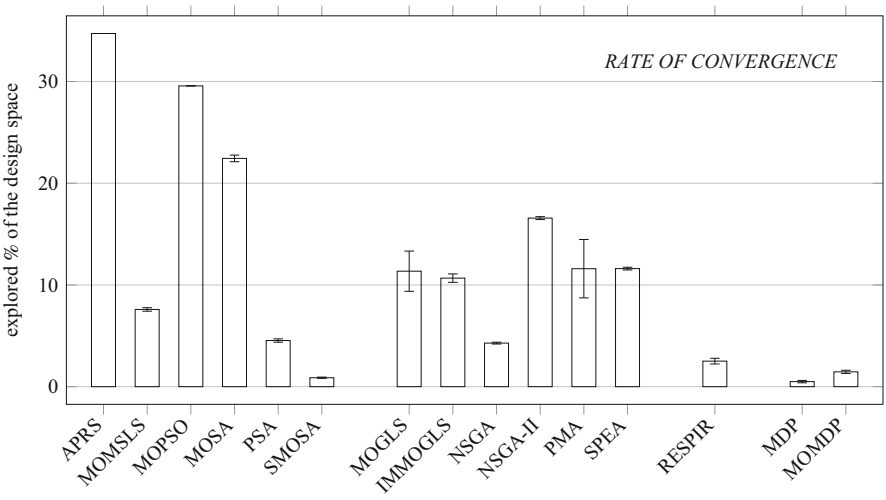

Fig. 6.5 The percentage of points in the design space evaluated by each algorithm for similar levels of accuracy, that is, ∼1% (From [29])

Figure 6.5 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 for all the algorithms to converge to the exact same quality result, and some algorithms show high variability. In fact, SMOSA and NSGA often do not converge to acceptable solutions. The final accuracy values obtained are shown in Fig. 6.6. Please note that the histograms and the error bars in Figs. 6.5, 6.6, 6.7, 6.8 and 6.9 show average values and standard deviations, respectively, for each algorithm over 30 experiments. No negative percentage actually resulted during the experiments.

Figure 6.5 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.

6.4.5.3 Quality of the Approximate Pareto Set

Figure 6.7 shows the number of Pareto points found by each algorithm, normalized by the average number found (for each benchmark).

The performance of most algorithms does not appear to show any statistically significant difference, with the exception of RESPIR, which finds, on average, 25% more points than all the other algorithms but with a slightly higher variance. Once

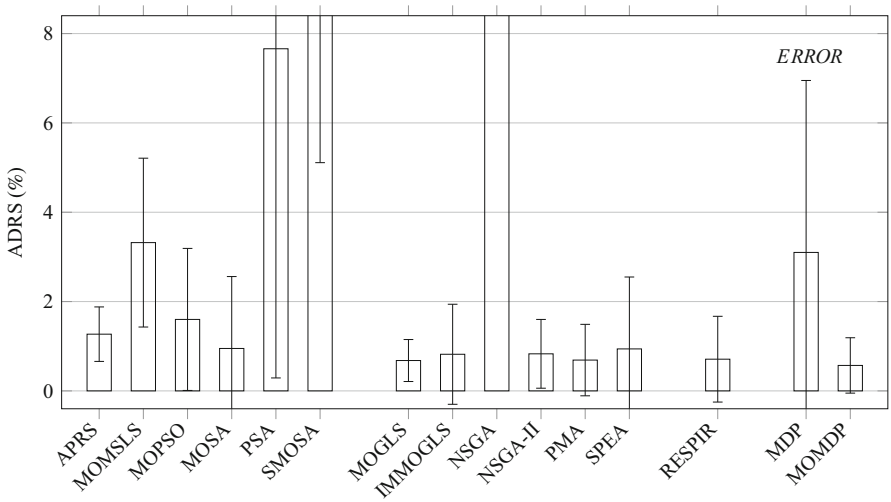

Fig. 6.6 Accuracy (ADRS) reached by each algorithm at convergence (From [29])

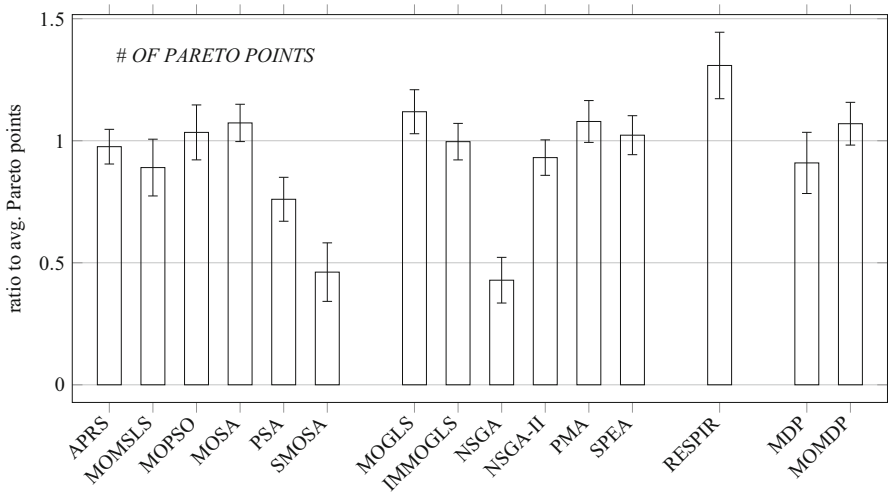

Fig. 6.7 The average number of points in the approximate Pareto set found by each algorithm, normalized with the average for each benchmark (From [29])

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 was found by RESPIR.

Concerning non-uniformity and concentration, all algorithms behave similarly, satisfyingly covering the design space and without concentrating on specific

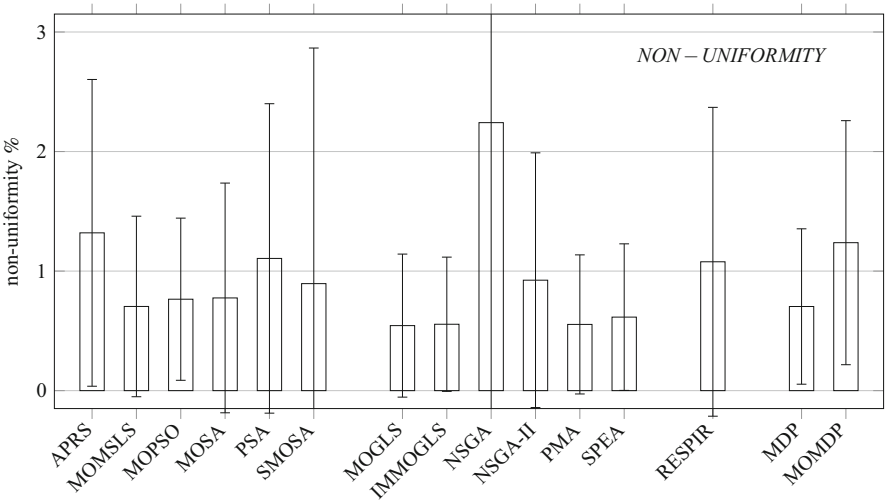

Fig. 6.8 The non-uniformity of the distribution of the points found in the approximate Pareto set found by each algorithm (From [29])

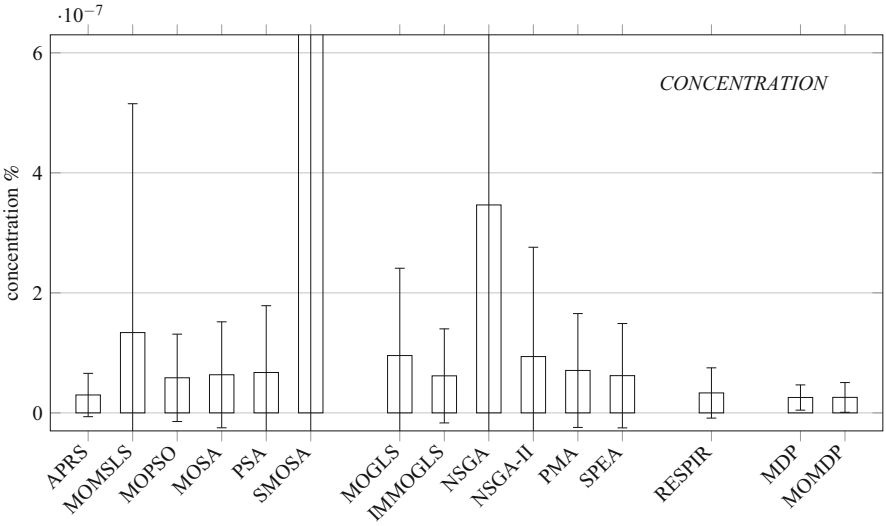

Fig. 6.9 The concentration of the points found in the approximate Pareto set found by each algorithm (From [29])

areas. Again, no statistically significant difference between the algorithms can be observed, with the only exception of SMOSA and NSGA, which produced the worst results as well as higher variance. Results are presented in Figs. 6.8, 6.9 and 6.10.

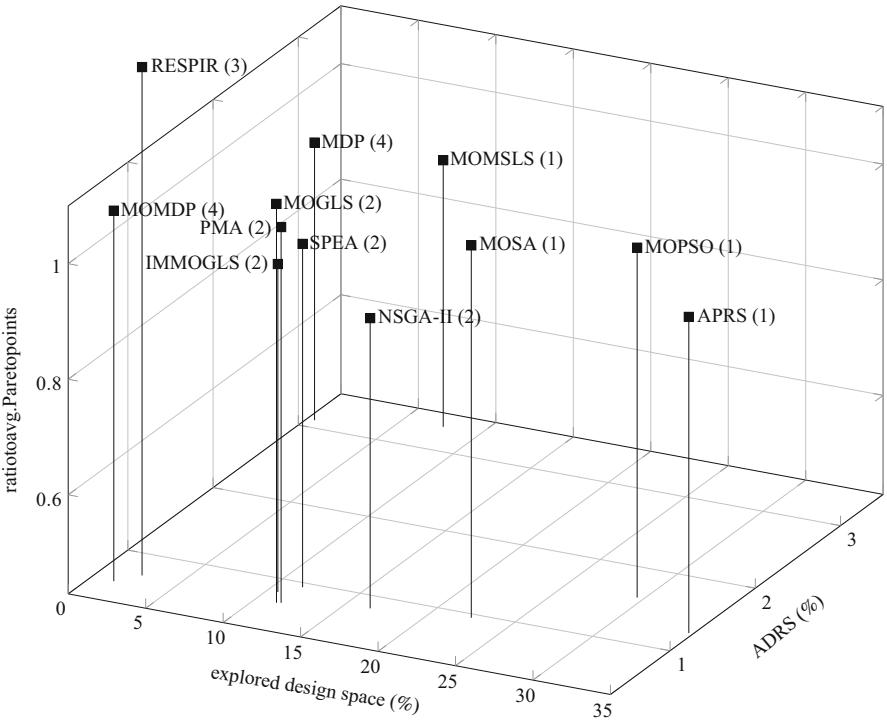

Fig. 6.10 3D representation of the algorithms performance showing % of the design space explored in order to reach convergence, the number of Pareto points (w.r.t the average number), and the ADRS error metric (NSGA, PSA, and SMOSA are omitted because of their large ADRS). The number after each algorithm name indicates the its class (from [29])

6.4.5.4 Scalability

The scalability column of Table 6.3 tells how an algorithm's performance degrades with the increasing 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, the size of the design space was progressively increased, starting with three parameters and adding the remaining three one by one.

Most algorithms of classes 1 and 2 are very scalable: the number of parameters – or the number of values per parameter – does not affect the overall result, even though the number of additional evaluations needed grows proportionally with the number of parameters (i.e., remains around ∼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 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 behavior. The number of evaluations for convergence remains ∼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 increases. While adding a parameter requires defining new bounds, adding new values comes without effort, and the number of evaluations needed increases less than linearly [3]. One drawback of MDP when compared to MOMDP is that it requires the estimation of an additional parameter (α; see [3]) when increasing the size of the design space, which might require additional evaluations.

6.5 Discussion

Selection of the best algorithm for a particular application is not an easy task, and it requires trading-off setup effort, scalability, expected number of simulations, and accuracy. Looking at the result from Sect. 6.4, one can draw some general guidelines according to the cost of each evaluation (in terms, e.g., of simulation time) and the size of the design space.

High evaluation costs make algorithms requiring a low number of simulations more appealing, even if they have a high upfront setup cost. On the contrary, if evaluations have moderate costs, one might want to trade-off a higher number of evaluations for a no-effort setup. Similarly, large design spaces favor scalable algorithms, while smaller spaces do not justify the extra work required to apply sophisticated algorithms.

In order to estimate the *actual* convergence time of an algorithm i, one must 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 setup time T_i^{setup} of the algorithm:

$$T_i^{\text{actual}} = (\nu_i \times |S| \times T_{\text{sim}}) + T_i^{\text{setup}} \tag{6.5}$$

The values of ν_i's are reported in Fig. 6.5. Regarding the T_i^{setup} values, the qualitative information in Table 6.3 was translated into a 10′- to 30-h range: observations showed 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 6.11 presents four plots: the size of the design space appears on the x axis while the time required by each simulation appears on the y axis. In each plot, areas are labeled with the name of the most suitable algorithm according to Eq. 6.5. 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

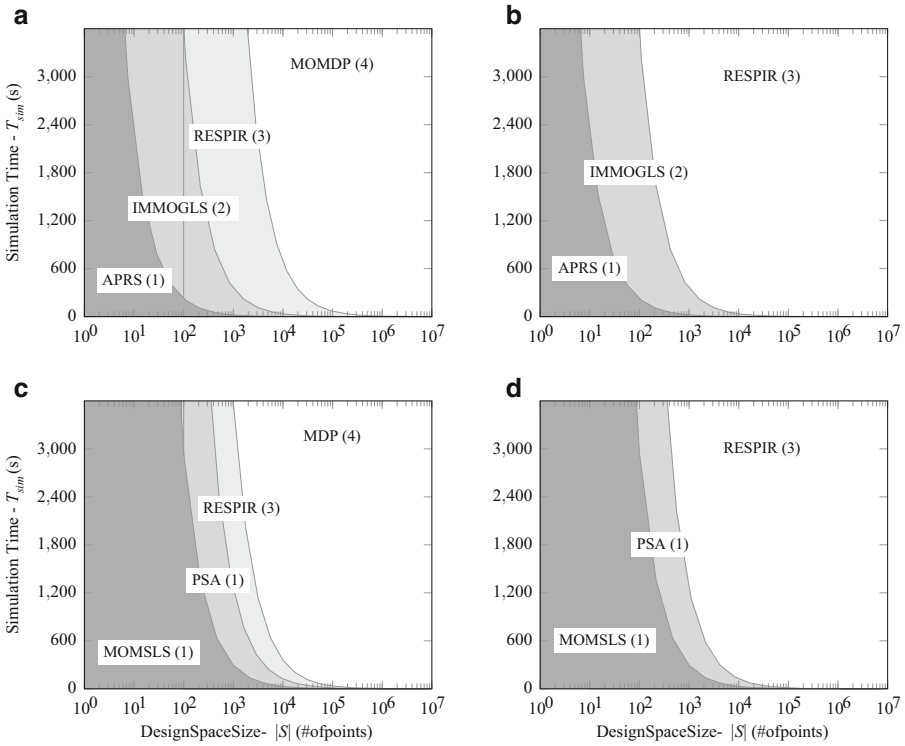

Fig. 6.11 Recommended algorithms for different design space size, simulation time and desired ADRS, whether domain knowledge is available or not. The number after each algorithm name indicates its class (From [29]). (**a**) ADRS $\sim$ 1%, w/ domain knowledge. (**b**) ADRS $\sim$ 1%, w/o domain knowledge. (**c**) ADRS $\sim$ 5%, w/ domain knowledge. (**d**) ADRS $\sim$ 5%, w/o domain knowledge

the most appealing solutions for small to medium design spaces with inexpensive evaluations. As explained in Sect. 6.4, APRS is a heuristic algorithm that requires very little setup effort, making it the ideal choice when simplicity is valued more than raw performance. MOMSLS is considerably faster but also more likely to produce a larger ADRS.

In large design spaces with high cost evaluations – if domain knowledge is available – the multi-objective MDP algorithm (MOMDP) [1] clearly grants better performance, both in terms of fast convergence to a very small ADRS and quality of the Pareto set.

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.

Table 6.4 Recommended algorithms for space size and evaluation effort

	Design space size w/ knowledge		
Eval cost	Small	Medium	Large
Low	APRS	PMA	IMMOGLS
Medium	PMA	RESPIR	RESPIR
High	RESPIR	MOMDP	MOMDP
	Design space size w/o knowledge		
Eval cost	Small	Medium	Large
Low	APRS	PMA	IMMOGLS
Medium	PMA	RESPIR	IMMOGLS
High	RESPIR	RESPIR	RESPIR

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 one can conclude from Fig. 6.11 is that algorithms with small setup 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 algorithms in classes 3 and 4 usually compensate for their longer configuration times when the exploration problem is sufficiently challenging.

These recommendations (summarized in Table 6.4) are qualitative, but do take into account all the parameters discussed in Sect. 6.4.

6.6 Existing Frameworks

All the algorithms presented in this chapter have an open-source implementation. There are many optimization tools that can be used or adapted for design space exploration. It is generally advisable to use an existing, well-tested implementation of one of these algorithms instead of going for the homebrew solution. The reason is that the most common open-source frameworks are used by many developers, and many of the issues and bugs have been found just by the sheer volume of users.

Most frameworks are available in the form of *libraries* to be used with a specific language or development environment. In the following, examples of existing frameworks are given, providing a short analysis of their strong and weak points.

6.6.1 jMetal

jMetal [7] is an object-oriented Java-based framework for multi-objective optimization with metaheuristics. It provides 12 different multi-objective algorithms, as well as some single-objective metaheuristics. jMetal is one of the most popular frameworks available and has a number of advantages:

- It is based on Java, that is, it is platform independent and the API is easily accessible by a programmer.
- It provides a graphical user interface and a set of test problems with quality indicators.
- It provides support for the parallel execution for a subset of its algorithms.

The main disadvantage of jMetal arises from one of its strong points: being written in Java, jMetal does not have optimal performance, and its parallel performance does not scale very well. Despite this, jMetal is one of the most comprehensive existing frameworks, and its utilization is recommended if the algorithm of interest is present in its library.

6.6.2 PaGMO/PyGMO

PaGMO (https://github.com/esa/pagmo/) is a C++ optimization framework initially developed by the European Space Agency for the optimization of interplanetary trajectories. The framework focuses on novel algorithms, parallelism, and performance. Compared to jMetal, PaGMO provides similar algorithms, but all of them are tuned for massively parallel execution. Therefore, PaGMO has an edge in terms of exploration performance. As jMetal, PaGMO comes with sample problems, metrics, and extension abilities. PaGMO also has Python bindings (PyGMO), which make it accessible with very simple scripts.

The main disadvantage of PaGMO is that despite having been in developing for a few years, it is not as user-friendly as jMetal: the installation is only from source, there is no graphical user interface, and its documentation incomplete. PaGMO is therefore recommended to the more experienced programmer or to the user with extreme need for performance.

6.6.3 MOHMLib++

The Multiple Objective MetaHeuristics Library in C++ (MOMHLib++) [16] is a C++ library providing 15 optimization algorithms. It is not as developed or maintained as jMetal or PaGMO, but it has few dependencies and it is very simple to use.

MOHMLib++ has no graphical user interface, no support for massively parallel execution, and no scripting language bindings. However, its small size and simplicity are usually very attractive to an unambitious developer looking to find a quick solution to an optimization or design exploration problem.

6.6.4 NASA

NASA (Non Ad-hoc Search Algorithm) [17] is not a collection of search algorithms but rather a DSE framework characterized by its modularity. Because of this reason, NASA can be used orthogonally and together with the strategies implemented, for

example, by MOHMLib++. The infrastructure of NASA is implemented in C++ and consists of several modules (including a search module, a feasibility checker, a simulator, and an evaluator) that can be extended or replaced by its user in plug-and-play fashion.

The main advantage of NASA is, indeed, its modularity. NASA allows the designer to fully decouple important functionalities (such as the choice of a search algorithm and the evaluation of multiple performance metrics) with the aid of only three *interface files*. Moreover, NASA is already integrated with two widely used system-level simulators – CASSE and Sesame – and it supports the parallel exploration of design space dimensions that are deemed independent by the designer. A shortcoming of NASA is the lack of pre-implemented search methodologies other than a proprietary GA, making the joint use of one of the previous frameworks a must.

6.7 Conclusions

Concluding, this chapter presented a classification and comparative analysis of 15 of the best and most recent multi-objective design exploration algorithms. The algorithms were applied to the exploration of a multi-processor platform and they were compared for setup effort, number of evaluations, quality of the resulting approximate Pareto set, and scalability. The obtained results were then used as guidelines for the choice of the algorithm best suited to the properties of a target design space. In particular, the experiments determined the most promising algorithms when considering design space size and evaluation effort. Finally, a list of reusable, open frameworks implementing DSE optimization strategies is provided.

References

1. Beltrame G, Nicolescu G (2011, in press) A multi-objective decision-theoretic exploration algorithm for platform-based design. In: Proceedings of design, automation and test in Europe (DATE)
2. Beltrame G, Fossati L, Sciuto D (2009) ReSP: a nonintrusive transaction-level reflective MPSoC simulation platform for design space exploration. IEEE Trans Comput Aided Des Integr Circuits Syst 28(12):1857–1869
3. Beltrame G, Fossati L, Sciuto D (2010) Decision-theoretic design space exploration of multiprocessor platforms. IEEE Trans Comput Aided Des Integr Circuits Syst 29(7):1083–1095. doi: 10.1109/TCAD.2010.2049053
4. Coello CA (2000) An updated survey of ga-based multiobjective optimization techniques. ACM Comput Surv 32(2):109–143. doi: 10.1145/358923.358929
5. Czyżżak P, Jaszkiewicz A (1998) Pareto simulated annealing–a metaheuristic technique for multiple-objective combinatorial optimization. J Multi-Criteria Decis Anal 7(1):34–47. doi: 10.1002/(SICI)1099-1360(199801)7:1<34::AID-MCDA161>3.0.CO;2-6
6. Deb K, Goel T (2001) Controlled elitist non-dominated sorting genetic algorithms for better convergence. In: Zitzler E, Thiele L, Deb K, Coello Coello C, Corne D (eds) Evolutionary multi-criterion optimization. Lecture notes in Computer Science, vol 1993. Springer, Heidelberg, pp 67–81. doi: 10.1007/3-540-44719-9_5

7. Durillo JJ, Nebro AJ (2011) jMetal: a java framework for multi-objective optimization. Adv Eng Softw 42:760–771
8. Erbas C (2006) System-level modelling and design space exploration for multiprocessor embedded system-on-chip architectures. Amsterdam University Press, Amsterdam
9. Fonseca CM, Fleming PJ (1995) An overview of evolutionary algorithms in multiobjective optimization. Evol Comput 3(1):1–16. doi: 10.1162/evco.1995.3.1.1
10. Fornaciari W, Sciuto D, Silvano C, Zaccaria V (2002) A sensitivity-based design space exploration methodology for embedded systems. Des Autom Embed Syst 7(1):7–33. doi: 10.1023/A:1019791213967
11. Givargis T, Vahid F, Henkel J (2001) System-level exploration for pareto-optimal configurations in parameterized systems-on-a-chip. In: 2001 IEEE/ACM international conference on computer aided design, ICCAD 2001, pp 25–30. doi: 10.1109/ICCAD.2001.968593
12. Ishibuchi H, Murata T (1996) Multi-objective genetic local search algorithm. In: Proceedings of IEEE international conference on evolutionary computation, pp 119–124. doi: 10.1109/ICEC.1996.542345
13. Ishibuchi H, Murata T (1998) A multi-objective genetic local search algorithm and its application to flowshop scheduling. IEEE Trans Syst Man Cybern C Appl Rev 28(3):392–403. doi: 10.1109/5326.704576
14. Jaddoe S, Pimentel AD (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, Heidelberg, pp 268–278
15. Jaszkiewicz A (2004) A comparative study of multiple-objective metaheuristics on the bi-objective set covering problem and the pareto memetic algorithm. Ann Oper Res 131:135–158. doi: 10.1023/B:ANOR.0000039516.50069.5b
16. Jaszkiewicz A, Dabrowski G (2005) MOMH: multiple objective meta heuristics. Available at the web site http://home.gna.org/momh/
17. Jia ZJ, Bautista T, Núñez A, Pimentel AD, Thompson M (2013) A system-level infrastructure for multidimensional MP-SoC design space co-exploration. ACM Trans Embed Comput Syst 13(1s):27:1–27:26. doi: 10.1145/2536747.2536749
18. Kaelbling LP, Littman ML, Moore AP (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285
19. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings IEEE international conference on neural networks, vol 4, pp 1942–1948. doi: 10.1109/ICNN.1995.488968
20. Lukasiewycz M, Glay M, Haubelt C, Teich J (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, pp 691–696
21. Mariani G, Brankovic A, Palermo G, Jovic J, Zaccaria V, Silvano C (2010) A correlation-based design space exploration methodology for multi-processor systems-on-chip. In: 2010 47th ACM/IEEE design automation conference (DAC), pp 120–125
22. Mohanty S, Prasanna VK, Neema S, Davis J (2002) Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. SIGPLAN Not 37(7):18–27
23. Nebro A, Durillo J, Coello C (2013) Analysis of leader selection strategies in a multi-objective particle swarm optimizer. In: 2013 IEEE congress on evolutionary computation (CEC), pp 3153–3160. doi: 10.1109/CEC.2013.6557955
24. Nebro AJ, Durillo JJ, Machín M, Coello Coello CA, Dorronsoro B (2013) A study of the combination of variation operators in the NSGA-II algorithm. In: Advances in artificial intelligence: 15th conference of the Spanish association for artificial intelligence, CAEPIA 2013, Madrid, 17–20 Sept 2013. Proceedings. Springer, Heidelberg, pp 269–278. doi: 10.1007/978-3-642-40643-0_28
25. Okabe T, Jin Y, Sendhoff B (2003) A critical survey of performance indices for multi-objective optimisation. In: The 2003 congress on, evolutionary computation, CEC '03, vol 2, pp 878–885. doi: 10.1109/CEC.2003.1299759

26. Palermo G, Silvano C, Zaccaria V (2008) Discrete particle swarm optimization for multi-objective design space exploration. In: 11th EUROMICRO conference on digital system design architectures, methods and tools, DSD'08, pp 641–644. doi: 10.1109/DSD.2008.21
27. Palermo G, Silvano C, Zaccaria V (2009) ReSPIR: a response surface-based pareto iterative refinement for application-specific design space exploration. IEEE Trans Comput Aided Des Integr Circuits Syst 28(12):1816–1829. doi: 10.1109/TCAD.2009.2028681
28. Palesi M, Givargis T (2002) Multi-objective design space exploration using genetic algorithms. In: CODES '02: Proceedings of the tenth international symposium on hardware/software codesign. ACM, Colorado, pp 67–72. doi: 10.1145/774789.774804
29. Panerati J, Beltrame G (2014) A comparative evaluation of multi-objective exploration algorithms for high-level design. ACM Trans Des Autom Electron Syst 19(2):15:1–15:22. doi: 10.1145/2566669
30. Russell SJ, Norvig P (1995) Artificial intelligence: a modern approach, 1st edn. Prentice Hall, Upper Saddle River
31. Serafini P (1994) Simulated annealing for multi objective optimization problems. In: Tzeng G, Wang H, Wen U, Yu P (eds) Multiple criteria decision making. Springer, New York, pp 283–292. doi: 10.1007/978-1-4612-2666-6_29
32. Sheldon D, Vahid F, Lonardi S (2007) Soft-core processor customization using the design of experiments paradigm. In: DATE conference, pp 1–6
33. Sivanandam SN, Deepa SN (2007) Introduction to genetic algorithms, 1st edn. Springer, Berlin/New York
34. Srinivas N, Deb K (1994) Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evol Comput 2(3):221–248. doi: 10.1162/evco.1994.2.3.221
35. Taghavi T, Pimentel AD (2011) Design metrics and visualization techniques for analyzing the performance of moeas in DSE. In: ICSAMOS, pp 67–76
36. Ulungu E, Teghem J, Fortemps P, Tuyttens D (1999) MOSA method: a tool for solving multiobjective combinatorial optimization problems. J Multi-Criteria Decis Anal 8(4):221–236. doi: 10.1002/(SICI)1099-1360(199907)8:4<221::AID-MCDA247>3.0.CO;2-O
37. Zaccaria V, Palermo G, Castro F, Silvano C, Mariani G (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
38. Zitzler E, Thiele L (1999) Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Trans Evol Comput 3(4):257–271. doi: 10.1109/4235.797969