Knowledge-Based Systems 87 (2015) 47–60

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Performance comparison of generational and steady-state asynchronous multi-objective evolutionary algorithms for computationally-intensive problems

Alexandru-Ciprian Zăvoianu [a,c,*], Edwin Lughofer [a], Werner Koppelstätter [c], Günther Weidenholzer [c], Wolfgang Amrhein [b,c], Erich Peter Klement [a,c]

[a] Department of Knowledge-based Mathematical Systems/Fuzzy Logic Laboratory Linz-Hagenberg, Johannes Kepler University of Linz, Austria
[b] Institute for Electrical Drives and Power Electronics, Johannes Kepler University of Linz, Austria
[c] LCM, Linz Center of Mechatronics, Linz, Austria

ARTICLE INFO

Article history:
Received 29 October 2014
Received in revised form 12 March 2015
Accepted 30 May 2015
Available online 5 June 2015

Keywords:
Evolutionary computation
Multi-objective optimization
Performance analysis
Master–slave parallelization
Steady-state evolution
Electrical drive design

ABSTRACT

In the last two decades, multi-objective evolutionary algorithms (MOEAs) have become ever more used in scientific and industrial decision support and decision making contexts the require an a posteriori articulation of preference. The present work is focused on a comparative analysis of the performance of two master–slave parallelization (MSP) methods, the canonical *generational* scheme and the *steady-state asynchronous* scheme. Both can be used to improve the convergence speed of multi-objective evolutionary algorithms that must use computationally-intensive fitness evaluation functions. Both previous and present experiments show that a correct choice for one or the other parallelization method can lead to substantial improvements with regard to the overall duration of the optimization process. Our main aim is to provide practitioners of MOEAs with a simple but effective method of deciding which MSP option is better given the particularities of the concrete optimization process. This in turn, would give the decision maker more time for articulating preferences (i.e., more flexibility). Our analysis is performed based on 15 well-known MOOP benchmark problems and two simulation-based industrial optimization processes from the field of electrical drive design. For the first industrial MOOP, when comparing with a preliminary study, applying the steady-state asynchronous MSP enables us to achieve an overall speedup (in terms of total wall-clock computation time) of $\approx 25\%$. For the second industrial MOOP, applying the steady-state MSP produces an improvement of $\approx 12\%$. We focus our study on two of the best known and most widely used MOEAs: the Non-dominated Sorting Genetic Algorithm II (NSGA-II) and the Strength Pareto Evolutionary Algorithm (SPEA2).

1. Introduction and state of the art

1.1. Motivation

Many real-world optimization problems usually arise in decision making contexts that involve several conflicting objectives (e.g. cost vs. quality, risk vs. return on investment) that should be simultaneously optimized. Problems falling within this class are referred to as *multi-objective optimization problems* (MOOPs in short). Generally, such problems do not have a single solution and "solving" them requires finding a set of non-dominated solutions called the *Pareto-optimal set* (in short, PS). Each solution (candidate) from this set is better than any other solution from the set with regard to at least one of the optimization objectives (i.e., no solution from this set is fully dominated by another solution). For many MOOPs, the Pareto-optimal set is unknown and/or infinite. Therefore, in most application domains, decision makers refer to the *Pareto non-dominated set* (in short, PN) which contains an arbitrarily fixed number of solutions that are able to provide a good approximation of the PS. The objective space representation of a Pareto non-dominated set is called the *Pareto front*.

In real-life scenarios, solving a MOOP is actually divided into two distinct stages:

– *the search stage* where the goal is to find solution candidates that are able to optimize (minimize) all the objectives of the MOOP (i.e., discover PNs that are as close as possible to the PS of the MOOP);

* Corresponding author at: Department of Knowledge-Based Mathematical Systems/Fuzzy Logic Laboratory Linz-Hagenberg, Johannes Kepler University of Linz, Austria. Tel.: +43 732 2468 4140; fax: +43 732 2468 4142.
E-mail address: ciprian.zavoianu@jku.at (A.-C. Zăvoianu).

http://dx.doi.org/10.1016/j.knosys.2015.05.029
0950-7051/

– *the decision making/articulation of preferences stage* where the goal is to determine the exact solution candidate(s) that incorporate(s) the best trade-offs in the decision maker's opinion.

The focus of this work lies exclusively with improving the general converge time of methods (evolutionary algorithms) that are commonly used to solve the *search stage* and find high-quality PNs. The motivation for this (and the general concept of a posteriori articulation of preference) is that once a clear and broad picture of all the existing trade-offs in the MOOP is available (i.e., a good Pareto front has been discovered), the decision maker has a lot of flexibility to tune the more subjective multi-criteria decision making part in order to select a very limited number of Pareto non-dominated solutions (sometimes just one) that will be con structed/implemented/applied in the real-life process modeled by the given MOOP.

Multi-objective evolutionary algorithms (MOEAs) have proven to be one of the most successful soft computing techniques for solving MOOPs [1–3]. This is because they are able to produce complete PNs over single runs. Like most stochastic methods, MOEAs are approximate methods that cannot guarantee finding the optimal solution set of the MOOP (i.e., the PS and the *true Pareto front* associated with it), but these algorithms are fairly robust and can find high quality non-dominated solution sets in reasonable time.

The main drawback of using MOEAs in practical applications is the fact that, in order to discover good solution sets, they usually require a large number of solutions to be evaluated during the optimization run. The issue can become particularly problematic for optimization problems that require very computationally-intensive fitness evaluation functions in order to compute objective or constraint values (e.g., consider physics-orientated simulation methods such as finite element methods, or the usage of software emulators in engineering design). In these cases, optimization runs can last for several days, as shown in [4] where MOEAs are used for the optimization of combustion in a diesel engine, or in [5] where MOEAs are applied for optimizing design parameters of electrical drives.

A very simple idea that displays immediate benefits in reducing the runtime of time intensive optimization runs is the parallelization and/or distribution of the MOEA run over a computer cluster or grid environment. There are several paradigms (architectural and/or conceptual models) of parallelizing a MOEA: master–slave, island, diffusion, hierarchical and hybrid models (please see Chapter 8 of [3] for an overview).

By far, the most straight forward and easiest to implement parallelization method for evolutionary algorithms is the **master–slave parallelization** (MSP) model: fitness evaluations are distributed between several slave nodes (computational units), while all the evolutionary operations (selection, crossover, mutation, etc.) are performed on a master node (computational unit). The MSP is suitable both for a generational approach, as well as for an asynchronous parallelization approach similar to the steady-state selection scheme described in [6]. The question of which of these two simple parallelization schemes is better, is an age old problem in the field of evolutionary computation. Very recently, Scott and De Jong started to analyze the problem more thoroughly [7]. In the present study we aim to offer some interesting insights into this matter from the general point of view of multi-objective evolutionary algorithms and from the particular perspective of trying to choose the best parallelization method when trying to optimize design parameters in electrical drive engineering. The findings of our analysis show that for two industrial problems, an overall computation time speedup of about 12 % and 25 % can be achieved by simply using an asynchronous MSP instead of a classical generational MSP (like the one applied in [5]). This improvement of the overall duration of the optimization seems to have no negative impact on the quality of the final solutions (i.e., of the PNs) the MOEAs are able to discover.

1.2. Basic concepts

When considering a computational process parallelized/distributed on a general master–slave architecture, one should distinguish among two types of tasks:

(i) **remote tasks** – very *time-intensive computations* that are performed on the slave nodes;
(ii) **sequential tasks** – include all the computations that must be performed on the master-node in order to *create*, *dispatch* and *retrieve* remote computation tasks.

It should be noted that in the case of most MOEAs, the "*create*" part of the sequential tasks includes applying typical genetic operators like (parent) selection, crossover, mutation, and survival for selection. Apart from these, in real-world master–slave parallelization setups for MOEAs, the duration of the sequential tasks is also affected by the fact that lengthy pre-evaluation steps must be performed locally (on the master node) for each generated individual, before dispatching the individual for remote fitness evaluation on the slave nodes. These pre-evaluation steps must be performed on the master node because of security concerns, software licensing issues, network configuration settings, etc. Whenever the average duration of the sequential tasks carried out on the master node is significant with regard to the average duration of the fitness evaluation tasks (i.e., the system displays a low **parallelization ratio**), the speed-up that can be achieved by employing a parallel/distributed architecture is affected (q.v. Amdahl's law).

Apart from the parallelization ratio, another aspect that must be considered refers to the **heterogeneity of the time-wise distributions of the** remote (i.e., **fitness evaluation** in the case of MOEAs) and sequential tasks. Although literature that focuses on the effects of fitness function time-wise heterogeneity on the MSP choice for MOEAs is scarce, a study by Yagoubi et al. from 2011 [4] indicates that, for MOOPs that display a heterogeneous (non-constant) time-wise fitness distribution, the steady state asynchronous parallelization is somewhat better in terms of convergence (Pareto quality and global run-time) than the generational approach. In their 2008 work [8], Durillo et al. also show evidence that applying a (synchronous) steady state approach when performing a MOEA run can bring improvements in terms of Pareto quality. The present research builds on these earlier findings, considerably extends and generalizes preliminary concepts and results reported in [9], and tries to *outline the main reasons that might influence the average performance* of the two MSP methods in the context of MOEAs.

Our main intention is to provide an analytical framework to help practitioners in this field to decide what is the most efficient parallelization option based on the particularities (achievable parallelization ratio, MOEA choice, MOEA parameterization, MOOP characteristics, etc.) of their concrete optimization scenarios. The comparison is focused on the very practical aspect of achievable Pareto quality/run-time performance. In other words, given a MOOP to solve, a MOEA to use and two very simple and fast-to-implement (master–slave) parallelization options, we want to know *which parallelization method is more likely to deliver the highest quality solution set in a pre-defined global run-time interval?*

In the next paragraphs we provide a compact description of the two MSP methods we consider in the present research and we explain why the parallelization choice can seriously impact the MOEA search (performance) behavior. Together with a motivation for our dual analysis of parallelization performance, in Section 2.3 we provide an outline of the rest of this work.