Theoretical Computer Science 551 (2014) 66–83

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Analysis of speedups in parallel evolutionary algorithms and $(1+\lambda)$ EAs for combinatorial optimization ☆

Jörg Lässig [a], Dirk Sudholt [b,*,1]

[a] Department of Computer Science, University of Applied Sciences Zittau/Görlitz, Germany
[b] Department of Computer Science, University of Sheffield, UK

ARTICLE INFO

Article history:
Received 13 March 2013
Received in revised form 12 September 2013
Accepted 29 June 2014
Available online 5 July 2014
Communicated by N. Krasnogor

Keywords:
Parallel evolutionary algorithms
Combinatorial optimization
Island model
Spatial structures
Offspring populations
Runtime analysis

ABSTRACT

Evolutionary algorithms are popular heuristics for solving various combinatorial problems as they are easy to apply and often produce good results. Island models parallelize evolution by using different populations, called islands, which are connected by a graph structure as communication topology. Each island periodically communicates copies of good solutions to neighboring islands in a process called migration. We consider the speedup gained by island models in terms of the parallel running time for problems from combinatorial optimization: sorting (as maximization of sortedness), shortest paths and Eulerian cycles. The results show in which settings and up to what degree evolutionary algorithms can be parallelized efficiently. Our results include offspring populations in $(1+\lambda)$ EAs as a special case. Potential speedups depend on many design choices such as the search operators, representations and fitness functions used on the islands, and also the parameters of the island model. In particular, we show that a natural instance for Eulerian cycles leads to exponential vs. logarithmic speedups, depending on the frequency of migration.

1. Introduction

1.1. Motivation

Evolutionary algorithms (EAs) are popular heuristics for various combinatorial problems as they often perform better than problem-specific algorithms with proven performance guarantees. They are easy to apply, even in cases where the problem is not well understood or when there is not enough time or expertise to design a problem-specific algorithm. They are a method of choice in black-box optimization, where no knowledge about the problem at hand is available and evaluating candidate solutions is the only way to learn about the problem. Another advantage is that EAs can be parallelized easily [2]. This is becoming more and more important, given the development in computer architecture and the rising number of CPU cores. Developing efficient parallel metaheuristics has emerged as a very active research area in recent years [3,2,4,5].

A simple way of using parallelization is to use so-called offspring populations: new solutions (offspring) are created and evaluated simultaneously on different processors. Island models use parallelization on a higher level. Subpopulations, called islands, which are connected by a graph structure, evolve independently for some time and periodically communicate copies

☆ An extended abstract with parts of the results and without proofs was presented at ISAAC 2011 [1].

* Corresponding author.

[1] Part of this work was done while the second named author was at the University of Birmingham, partially supported by EPSRC grant EP/D052785/1. The authors would like to thank the anonymous reviewers for their helpful comments.

http://dx.doi.org/10.1016/j.tcs.2014.06.037
0304-3975/

of good solutions to neighboring islands in a process called migration. Migration is typically performed every τ iterations, the parameter τ being known as *migration interval*.

Island models and similar parallel metaheuristics have been remarkably successful on a variety of applications. A very recent survey paper on parallel EAs and other parallel metaheuristics by Alba, Luque, and Nesmachnow [6] lists 130 successful high-impact applications of these algorithms to real-world problems in various domains: automation and robotics; bioinformatics; engineering design; hydraulic engineering; information processing, classification, and data mining; manufacturing and industrial applications; routing, logistics and vehicle planning; scheduling; software engineering and software development; telecommunications; energy and power network optimization; health and medicine; and many more.

Despite widespread applications and a long history of parallel EAs, the theory of these algorithms is lagging far behind. Present theoretical work concerns the study of the spread of information, or *takeover time*, in simplified models [7–11] as well as investigations of island models on constructed test functions [12–14]. Even the impact of the most basic parameters in island models on performance is not well understood [5]. It is agreed that more fundamental research is needed to get insight into when and how parallel EAs are effective [15,5].

1.2. Notion of speedup

One of the most pressing questions for the design and analysis of parallel EAs is what speedup can be obtained by using parallelization. There are many notions of "speedup" and, as pointed out in [6], to date there is no consent in the scientific community as to what is the best. Alba [16] provided a taxonomy: *strong speedup* compares a parallel EA against the best known sequential algorithm, whether this is an EA or not. The comparison is made according to the execution time, i.e. wall-clock time, and the speedup is defined as ratio of wall-clock times. In contrast, *weak speedup* compares a parallel EA against a sequential version of the same EA. The latter can mean comparing a parallel EA on λ processors against a single, panmictic population, which may have λ times the size of one processor's population; this is called *weak speedup versus panmixia*. Since this means comparing two very different algorithms, it makes more sense to compare a parallel EA with λ islands against an EA on a single island; this is called *weak orthodox speedup*.

The question addressed in this work is what speedup can be obtained by using multiple populations operating in parallel instead of one. This corresponds to the notion of *weak orthodox speedup* in island models.

Formally, for an island model with λ islands, the *parallel running time* T^{par} is defined as the number of generations until the first global optimum is evaluated. Let T_λ be the number of generations before the island model finds a global optimum for the first time. Then

$$T^{\text{par}} := T_\lambda.$$

The *sequential running time* T^{seq} is defined as the number of function evaluations until the first global optimum is evaluated. It thus captures the overall effort across all processors. It is formally defined as

$$T^{\text{seq}} := \lambda \cdot T_\lambda.$$

In both measures, T^{par} and T^{seq}, we allow ourselves to neglect the cost of the initialization as this only adds a fixed term to the running times.

The *(weak orthodox) speedup* is then defined as the ratio of expected running times of a single island and an island model with λ islands:

$$\mathrm{E}(T_1)/\mathrm{E}(T_\lambda).$$

Note that our notion of speedup is defined with regard to the number of generations. This makes sense for a theoretical investigation as we aim for results that are independent of the specific parallel hardware. Yet the execution time is determined by the nature and parameters of the parallel hardware. Our results can be transferred to specific parallel architectures using the following estimations.

When the algorithm is stopped once it reaches a global optimum, the execution time of an island model with λ islands can be estimated as

$$\mathrm{Exec}_\lambda = T_\lambda \cdot \left(\mathrm{Exec}_\lambda^{\text{gen}} + \mathrm{Exec}_\lambda^{\text{migr}}\right) \tag{1}$$

where $\mathrm{Exec}^{\text{gen}}$ is the average execution time for one generation (excluding migration) and $\mathrm{Exec}^{\text{migr}}$ is the average execution time for migration. The latter depends on the number of islands, the communication topology, the migration interval and the number of individuals migrated. For sequential algorithms ($\lambda = 1$) we have $\mathrm{Exec}_1^{\text{migr}} = 0$.

In the following, we consider speedups with regard to the number of generations only, ignoring the differences in $\mathrm{Exec}^{\text{migr}}$. This makes sense in settings where $\mathrm{Exec}^{\text{gen}} \gg \mathrm{Exec}^{\text{migr}}$; for instance, when fitness evaluations are so expensive that they dominate the execution time. Otherwise, the speedups stated here may be optimistic as the overhead induced by migration is ignored. However, with additional information about $\mathrm{Exec}^{\text{gen}}$ and $\mathrm{Exec}^{\text{migr}}$ and Eq. (1) our results can easily be transferred to a notion of weak orthodox speedup with regard to execution times.