# 46. Parallel Evolutionary Algorithms

Dirk Sudholt

Evolutionary algorithms (EAs) have given rise to many parallel variants, fuelled by the rapidly increasing number of CPU cores and the ready availability of computation power through GPUs and cloud computing. A very popular approach is to parallelize evolution in island models, or coarse-grained EAs, by evolving different populations on different processors. These populations run independently most of the time, but they periodically communicate genetic information to coordinate search. Many applications have shown that island models can speed up computation significantly, and that parallel populations can further increase solution diversity.

The aim of this book chapter is to give a gentle introduction into the design and analysis of parallel evolutionary algorithms, in order to understand how parallel EAs work, and to explain when and how speedups over sequential EAs can be obtained.

Understanding how parallel EAs work is a challenging goal as they represent interacting stochastic processes, whose dynamics are determined by several parameters and design choices. This chapter uses a theory-guided perspective to explain how key parameters affect performance, based on recent advances on the theory of parallel EAs. The presented results give insight into the fundamental working principles of parallel EAs, assess the impact of parameters and design choices on performance, and contribute to an informed design of effective parallel EAs.



Recent years have witnessed the emergence of a huge number of parallel computer architectures. Almost every desktop or notebook PC, and even mobile phones, come with several CPU cores built in. Also GPUs

have been discovered as a source of massive computation power at no extra cost. Commercial IT solutions often use clusters with hundreds and thousands of CPU cores and cloud computing has become an affordable and convenient way of gaining CPU power.

With these resources readily available, it has become more important than ever to design algorithms that can be implemented effectively in a parallel architecture. Evolutionary algorithms (EA) are popular general-purpose metaheuristics inspired by the natural evolution of species. By using operators like mutation, recombination, and selection, a multi-set of solutions – the *population* – is evolved over time. The hope is that this artificial evolution will explore vast regions of the search space and yet use the principle of *survival of the fittest* to generate good solutions for the problem at hand. Countless applications as well as theoretical results have demonstrated that these algorithms are effective on many hard optimization problems.

One of many advantages of EAs is that they are easy to parallelize. The process of artificial evolution can be implemented on parallel hardware in various ways. It is possible to parallelize specific operations, or to parallelize the evolutionary process itself. The latter approach has led to a variety of search algorithms called island models or cellular evolutionary algorithms. They differ from a sequential implementation in that evolution happens in a spatially structured network. Subpopulations evolve on different processors and good solutions are communicated between processors. The spread of information can be tuned easily via key parameters of the algorithm. A slow spread of information can lead to a larger diversity in the system, hence increasing exploration.

Many applications have shown that parallel EAs can speed up computation and find better solutions, compared to a sequential EA. This book chapter reviews the most common forms of parallel EAs. We highlight what distinguishes parallel EAs from sequential EAs. We also we make an effort to understand the search dynamics of parallel EA. This addresses a very hot topic since, as of today, even the impact of the most basic parameters of a parallel evolutionary algorithms are not well understood.

The chapter has a particular emphasis on theoretical results. This includes runtime analysis, or computational complexity analysis. The goal is to estimate the expected time until an EA finds a satisfactory solution for a particular problem, or problem class, by rigorous mathematical studies. This area has led to very fruitful results for general EAs in the last decade [46.1, 2]. Only recently have researchers turned to investigating parallel evolutionary algorithms from this perspective [46.3–7]. The results help to get insight into the search behavior of parallel EAs and how parameters and design choices affect performance. The presentation of these results is kept informal in order to make it accessible to a broad audience. Instead of presenting theorems and complete formal proofs, we focus on key ideas and insights that can be drawn from these analyses.

The outline of this chapter is as follows. In Sect. 46.1 we first introduce parallel models of evolutionary algorithms, along with a discussion of key design choices and parameters. Section 46.2 considers performance measures for parallel EAs, particularly notions for *speedup* of a parallel EA when compared to sequential EAs.

Section 46.3 deals with the spread of information in parallel EAs. We review various models used to describe how the number of *good* solutions increases in a parallel EA. This also gives insight into the time until the whole system is taken over by good solutions, the so-called *takeover time*.

In Sect. 46.4 we present selected examples where parallel EAs were shown to outperform sequential evolutionary algorithms. Drastic speedups were shown on illustrative example functions. This holds for various forms of parallelization, from independent runs to offspring populations and island models.

Section 46.5 finally reviews a general method for estimating the expected running time of parallel EAs. This method can be used to transfer bounds for a sequential EA to a corresponding parallel EA, in an automated fashion. We go into a bit more detail here, in order to enable the reader to apply this method by her-/himself. Illustrative example applications are given that also include problems from combinatorial optimization.

The chapter finishes with conclusions in Sect. 46.6 and pointers to further literature on parallel evolutionary algorithms.

## 46.1 Parallel Models

### 46.1.1 Master–Slave Models

There are many ways how to use parallel machines. A simple way of using parallelization is to execute operations on separate processors. This can concern variation operators like mutation and recombination as well as function evaluations. In fact, it makes most sense for function evaluations as these operations can be performed independently and they are often among the most expensive operations. This kind of architecture is known as *master–slave model*. One machine represents the master and it distributes the workload for executing operations to several other machines called slaves. It is well suited for the creation of offspring populations as offspring can be created and evaluated independently, after suitable parents have been selected.

The system is typically synchronized: the master waits until all slaves have completed their operations before moving on. However, it is possible to use asynchronous systems where the master does not wait for slaves that take too long.

The behavior of synchronized master–slave models is not different from their sequential counterparts. The implementation is different, but the algorithm – and therefore search behavior – is the same.

### 46.1.2 Independent Runs

Parallel machines can also be used to simulate different, independent runs of the same algorithm in parallel. Such a system is very easy to set up as no communication during the runtime is required. Only after all runs have been stopped, do the results need to be collected and the best solution (or a selection of different high-quality solutions) is output.

Alternatively, all machines can periodically communicate their current best solutions so that the system can be stopped as soon as a satisfactory solution has been found. As for master–slave models, this prevents us from having to wait until the longest run has finished.

Despite its simplicity, independent runs can be quite effective. Consider a setting where a single run of an algorithm has a particular *success probability*, i. e., a probability of finding a satisfactory solution within a given time frame. Let this probability be denoted $p$. By using several independent runs, this success probability can be increased significantly. This approach is commonly known as *probability amplification*.

The probability that in $\lambda$ independent runs no run is successful is $(1-p)^{\lambda}$. The probability that there is at least one successful run among these is, therefore,

$$1-(1-p)^{\lambda} \,. \tag{46.1}$$

Figure 46.1 illustrates this amplified success probability for various choices of $\lambda$ and $p$.

We can see that for a small number of processors the success probability increases almost linearly. If the number of processors is large, a saturation effect occurs. The benefit of using ever more processors decreases with the number of processors used. The point where saturation happens depends crucially on $p$; for smaller success probabilities saturation happens only with a fairly large number of processors.

Furthermore, independent runs can be set up with different initial conditions or different parameters. This is useful to effectively explore the parameter space and to find good parameter settings in a short time.

### 46.1.3 Island Models

Independent runs suffer from obvious drawbacks: once a run reaches a situation where its population has become stuck in a difficult local optimum, it will most likely remain stuck forever. This is unfortunate since other runs might reach more promising regions of the search space at the same time. It makes more sense to

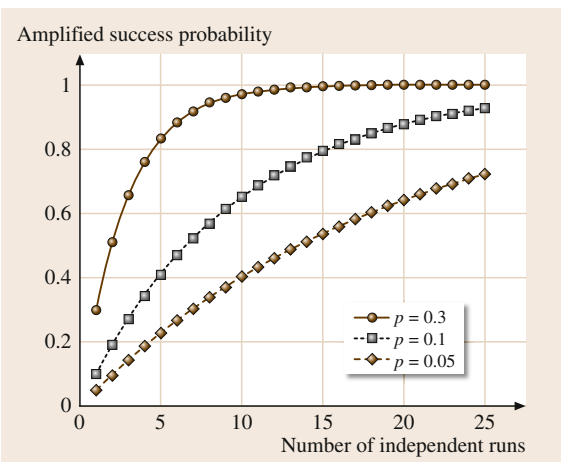


**Fig. 46.1** Plots of the amplified success probability $1-(1-p)^{\lambda}$ of a parallel system with $\lambda$ independent runs, each having success probability $p$

establish some form of communication between the different runs to coordinate search, so that runs that have reached low-quality solutions can join in on the search in more promising regions.

In *island models*, also called *distributed EAs*, the *coarse-grained model*, or the *multi-deme model*, the population of each run is regarded an island. One often speaks of islands as *subpopulations* that together form the population of the whole island model. Islands evolve independently as in the independent run model, for most of the time. However, periodically solutions are exchanged between islands in a process called *migration*.

The idea is to have a *migration topology*, a directed graph with islands as its nodes and directed edges connecting two islands. At certain points of time selected individuals from each island are sent off to neighbored islands, i. e., islands that can be reached by a directed edge in the topology. These individuals are called migrants and they are included in the target island after a further selection process. This way, islands can communicate and compete with one another. Islands that get stuck in low-fitness regions of the search space can be taken over by individuals from more successful islands. This helps to coordinate search, focus on the most promising regions of the search space, and use the available resources effectively. An example of an island model is given in Fig. 46.2. Algorithm 46.1 shows the general scheme of a basic island model.

*Algorithm 46.1 Scheme of an island model with migration interval $\tau$*

1: Initialize a population made up of subpopulations or islands, $P^{(0)} = \{P_1^{(0)}, \ldots, P_m^{(0)}\}$.
2: Let $t := 1$.
3: **loop**
4: **for** each island $i$ **do in parallel**
5: **if** $t$ mod $\tau = 0$ **then**
6: Send selected individuals from island $P_i^{(t)}$ to selected neighbored islands.
7: Receive immigrants $I_i^{(t)}$ from islands for which island $P_i^{(t)}$ is a neighbor.
8: Replace $P_i^{(t)}$ by a subpopulation resulting from a selection among $P_i^{(t)}$ and $I_i^{(t)}$.
9: **end if**
10: Produce $P_i^{(t+1)}$ by applying reproduction operators and selection to $P_i^{(t)}$.
11: **end for**
12: Let $t := t + 1$.
13: **end loop**

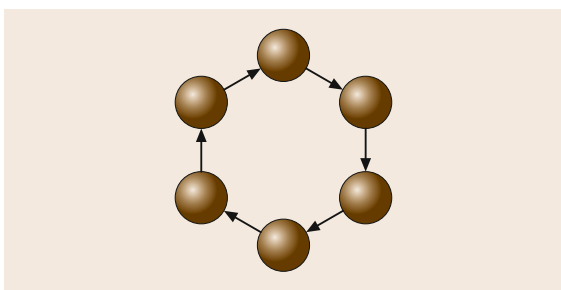

**Fig. 46.2** Sketch of an island model with six islands and an example topology

There are many design choices that affect the behavior of such an island model:

- *Emigration policy*. When migrants are sent, they can be removed from the sending island. Alternatively, copies of selected individuals can be emigrated. The latter is often called *pollination*. Also the selection of migrants is important. One might select the best, worst, or random individuals.
- *Immigration policy*. Immigrants can replace the worst individuals in the target population, random individuals, or be subjected to the same kind of selection used within the islands for parent selection or selection for replacement. Crowding mechanisms can be used, such as replacing the most similar individuals. In addition, immigrants can be recombined with individuals present on the island before selection.
- *Migration interval*. The time interval between migrations determines the speed at which information is spread throughout an island model. Its reciprocal is often called *migration frequency*. Frequent migrations imply a rapid spread of information, while rare migrations allow for more exploration. Note that a migration interval of $\infty$ yields independent runs as a special case.
- *Number of migrants*. The number of migrants, also called *migration size*, is another parameter that determines how quickly an island can be taken over by immigrants.
- *Migration topology*. Also the choice of the migration topology impacts search behavior. The topology can be a directed or undirected graph – after all, undirected graphs can be seen as special cases of directed graphs. Common topologies include unidirectional rings (a ring with directed edges

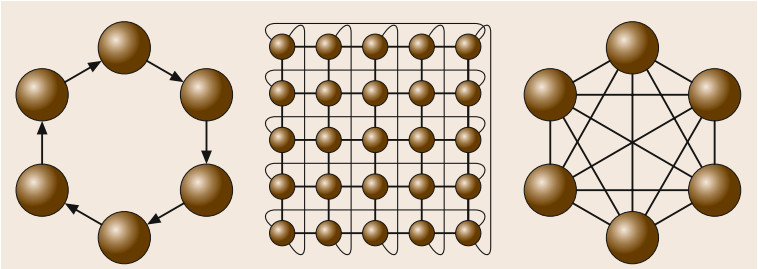

**Fig. 46.3** Sketches of common topologies: a unidirectional ring, a torus, and a complete graph. Other common topologies include bidirectional rings where all edges are undirected and grid graphs where the edges wrapping around the torus are removed

only in one direction), bidirectional rings, torus or grid graphs, hypercubes, scale-free graphs [46.8], random graphs [46.9], and complete graphs. Figure 46.3 sketches some of these topologies. An important characteristic of a topology $T = (V, E)$ is its *diameter*: the maximum number of edges on any shortest path between two vertices. Formally, $\mathrm{diam}(T) = \max_{u,v \in V} \mathrm{dist}(u, v)$, where $\mathrm{dist}(u, v)$ is the graph distance, the number of edges on a shortest path from $u$ to $v$. The diameter gives a good indication of the time needed to propagate information throughout the topology. Rings and torus graphs have large diameters, while hypercubes, complete graphs, and many scale-free graphs have small diameters.

Island models with non-complete topologies are also called *stepping stone models*. The impact of these design choices will be discussed in more detail in Sect. 46.3.

If all islands run the same algorithm under identical conditions, we speak of a *homogeneous island model*. *Heterogeneous island models* contain islands with different characteristics. Different algorithms might be used, different representations, objective functions, or parameters. Using heterogeneous islands might be useful if one is not sure what the best algorithm is for a particular problem. It also makes sense in the context of multiobjective optimization or when a diverse set of solutions is sought, as the islands can reflect different objective functions, or variations of the same objective functions, with an emphasis on different criteria.

*Skolicki* [46.10] proposed a two-level view of search dynamics in island models. The term *intra-island evolution* describes the evolutionary process that takes place within each island. On a higher level, *inter-island evolution* describes the interaction between different islands. He argues that islands can be regarded as individuals in a higher-level evolution. Islands compete with one another and islands can take over other islands, just like individuals can replace other individuals in a regular population. One conclusion is that with this perspective an island models looks more like a compact entity.

The two levels of evolution obviously interact with one another. Which level is more important is determined by the migration interval and the other parameters of the system that affect the spread of information.

### 46.1.4 Cellular EAs

Cellular EAs represent a special case of island models with a more fine-grained form of parallelization. Like in the island model we have islands connected by a fixed topology. Rings and two-dimensional torus graphs are the most common choice. The most striking characteristic is that each island only contains a single individual. Islands are often called *cells* in this context, which explains the term *cellular EA*. Each individual is only allowed to mate with its neighbors in the topology. This kind of interaction happens in every generation. This corresponds to a migration interval of 1 in the con-

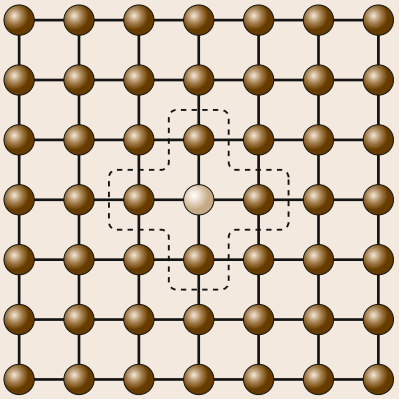

**Fig. 46.4** Sketch of a cellular EA on a $7 \times 7$ grid graph. The *dashed line* indicates the neighborhood of the highlighted cell

text of island models. Figure 46.4 shows a sketch of a cellular EA. A scheme of a cellular EA is given in Algorithm 46.2.

*Algorithm 46.2 Scheme of a cellular EA*

1: Initialize all cells to form a population $P^{(0)} = \{P_1^{(0)}, \ldots, P_m^{(0)}\}$. Let $t := 0$.
2: **loop**
3: **for** each cell $i$ **do in parallel**
4: Select a set $S_i$ of individuals from $P_i^{(t)}$ out of all cells neighbored to cell $i$.
5: Create a set $R_i$ by applying reproduction operators to $S_i$.
6: Create $P_i^{(t+1)}$ by selecting an individual from $\{P_i^{(t)}\} \cup R_i$.
7: **end for**
8: Let $t := t + 1$.
9: **end loop**

Cellular EAs yield a much more fine-grained system; they have therefore been called *fine-grained models*, *neighborhood models*, or *diffusion models*. The difference to island models is that no evolution takes place on the cell itself, i. e., there is no intra-island evolution. Improvements can only be obtained by cells interacting with one another. It is, however, possible that an island can interact with itself.

In terms of the two-level view on island models, in cellular EAs the intra-island dynamics have effectively been removed. After all, each island only contains a single individual. Fine-grained models are well suited for investigations of inter-island dynamics. In fact, the first runtime analyses considered fine-grained island models, where each island contains a single individual [46.4, 5]. Other studies dealt with fine-grained systems that use a migration interval larger than 1 [46.3, 6, 7].

For replacing individuals the same strategies as listed for island models can be used. All cells can be updated synchronously, in which case we speak of a *synchronous cellular EA*. A common way of implementing this is to create a new, temporary population. All parents are taken from the current population and new individuals are written into the temporary population. At the end of the process, the current population is replaced by the temporary population.

Alternatively, cells can be updated sequentially, resulting in an *asynchronous cellular EA*. This is likely to result in a different search behavior as individuals can mate with offspring of their neighbors. *Alba* et al. [46.11] define the following update strategies. The terms are tailored towards two-dimensional grids or torus graphs as they are inspired by cellular automata. It is, however, easy to adapt these strategies to arbitrary topologies:

- *Uniform choice*: the next cell to be updated is chosen uniformly at random.
- *Fixed line sweep*: the cells are updated sequentially, line by line in a grid/torus topology.
- *Fixed random sweep*: the cells are updated sequentially, according to some fixed order. This order is determined by a permutation of all cells. This permutation is created uniformly at random during initialization and kept throughout the whole run.
- *New random sweep*: this strategy is like fixed random sweep, but after each sweep is completed a new permutation is created uniformly at random.

A time step or generation is defined as the time needed to update $m$ cells, $m$ being the number of cells in the grid. The last three strategies ensure that within each time step each cell is updated exactly once. This yields a much more balanced treatment for all cells. With the uniform choice model is it likely that some cells must wait for a long time before being updated. In the limit, the waiting time for updates follows a Poisson distribution. Consider the random number of updates until the last cell has been updated at least once. This random process is known as the *coupon collector problem* [46.12, page 32], as it resembles the process of collecting coupons, which are drawn uniformly at random. A simple analysis shows that the expected number of updates until the last cell has been updated in the uniform choice model (or all coupons have been collected) equals

$$m \cdot \sum_{i=1}^{m} 1/i \approx m \cdot \ln(m) \,.$$

This is equivalent to

$$\sum_{i=1}^{m} 1/i \approx \ln m$$

time steps, which can be significantly larger than 1, the time for completing a sweep in any given order.

Cellular EAs are often compared to cellular automata. In the context of the latter, it is common practice to consider a two-dimensional grid and different neighborhoods. The neighborhood in Fig. 46.2 is called the *von Neumann neighborhood* or *Linear 5*. It includes the cell itself and its four neighbors along the directions north, south, west, and east. The *Moore neighborhood* or *Compact 9* in addition also contains the four cells to the north west, north east, south west, and south east. Also larger neighborhoods are common, containing cells that are further away from the center cell.

Note that using a large neighborhood on a two-dimensional grid is equivalent to considering a graph where, starting with a torus graph, for each vertex edges to nearby vertices have been added. We will, therefore, in the remainder of this chapter stick to the common notion of neighbors in a graph (i. e., vertices connected by an edge), unless there is a good reason not to.

### 46.1.5 A Unified Hypergraph Model for Population Structures

*Sprave* [46.13] proposed a unified model for population structures. It is based on hypergraphs; an extension of graphs where edges can connect more than two vertices. We present an informal definition to focus on the ideas; for formal definitions we refer to [46.13]. A hypergraph contains a set of vertices and a collection of hyperedges. Each hyperedge is a non-empty set of vertices. Two vertices are neighbored in the hypergraph if there is a hyperedge that contains both vertices. Note that the special case where each hyperedge contains two different vertices results in an undirected graph.

In Sprave's model each vertex represents an individual. Hyperedges represent the set of possible parents for each individual. The model unifies various common population models:

- *Panmictic populations*: for panmictic populations we have a set of vertices $V$ and there is a single hyperedge that equals the whole vertex set. This reflects the fact that in a panmictic population each individual has all individuals as potential parents.
- *Island models with migration*: if migration is understood in the sense that individuals are removed, the set of potential parents for an individual contains all potential immigrants as well as all individuals from its own island, except for those that are being emigrated.
- *Island models with pollination*: if pollination is used, the set of potential parents contains all immigrants and all individuals on its own island.
- *Cellular EAs*: For each individual, the potential parents are its neighbors in the topology.

In the case of coarse-grained models, the hypergraph may depend on time. More precisely, we have different sets of potential parents when migration is used, compared to generations without migration. Sprave considers this by defining a dynamic population structure: instead of considering a single, fixed hypergraph, we consider a sequence of hypergraphs over time.

### 46.1.6 Hybrid Models

It is also possible to combine several of the above approaches. For instance, one can imagine an island model where each island runs a cellular EA to further promote diversity. Or one can think of hierarchical island models where islands are island models themselves. In such a system it makes sense that the inner-layer island models use more frequent migrations than the outer-layer island model. Island models and cellular EAs can also be implemented as master–slave models to achieve a better speedup.

## 46.2 Effects of Parallelization

An obvious effect of parallelization is that the computation time can be reduced by using multiple processors. This section describes performance measured that can be used to define this speedup. We also consider beneficial effects of using parallel EAs that can lead to superlinear speedups.

### 46.2.1 Performance Measures for Parallel EAs

The computation time of a parallel EA can be defined in various ways. It makes sense to use wall-clock time as the performance measure as this accounts for the

overhead by parallelization. Under certain conditions, it is also possible to use the number of generations or function evaluations. This is feasible if these measures reflect the real running time in an adequate way, for instance if the execution of a generation (or a function evaluation) dominates the computational effort, including the effort for coordinating different machines. It is also feasible if one can estimate the overhead or the communication costs separately.

We consider settings where an EA is run until a certain goal is fulfilled. Goals can be reaching a global or local optimum or reaching a certain minimum fitness. In such a setting the goal is fixed and the running time of the EA can vary. This is in contrast to setups where the running time is fixed to a predetermined number of generations and then the quality or accuracy of the obtained solutions is compared. As *Alba* pointed out [46.14], performance comparisons of parallel and sequential EAs only make sense if they reach the same accuracy. In the following, we focus on the former setting where the same goal is used.

Still, defining speedup formally is far from trivial. It is not at all clear against what algorithm a parallel algorithm should be compared. However, this decision is essential to clarify the meaning of speedup. Not clarifying it, or using the wrong comparison, can easily yield misleading results and false claims. We present a taxonomy inspired by *Alba* [46.14], restricted to cases where a fixed goal is given:

- *Strong speedup*: the parallel run time of a parallel algorithm is compared against the sequential run time of the *best known sequential algorithm*. It was called *absolute speedup* by *Barr* and *Hickman* [46.15]. This measure captures in how far parallelization can improve upon the best known algorithms. However, it is often difficult to determine the best sequential algorithm. Most researchers, therefore, do not use strong speedup [46.14].
- *Weak speedup*: the parallel run time of an algorithm is compared against *its own sequential run time*. This gives rise to two subcases where the notion of its own sequential run time is made precise:
  - *Single machine/panmixia*: the parallel EA is compared against a canonical, panmictic version of it, running on a single machine. For instance, we might compare an island model with $m$ islands against an EA running a single island. Thereby, the EA run on all islands is the same in both cases.
  - *Orthodox*: the parallel EA running on $m$ machines is compared against the same parallel EA running on a single machine. This kind of speedup was called *relative speedup* by *Barr* and *Hickman* [46.15].

In the light of these essential differences, it is essential for researchers to clarify their notion of speedup.

Having clarified the comparison, we can now define the speedup and other measures. Let $T_m$ denote the time for $m$ machines to reach the goal. Let $T_1$ denote the time for a single machine, where the algorithm is chosen according to one of the definitions of speedup defined above.

The idea is to consider the ratio of $T_m$ and the time for a single machine, $T_1$, as speedup. However, as we are dealing with randomized algorithms, $T_1$ and $T_m$ are random variables and so the ratio of both is a random variable as well. It makes more sense to consider the ratio of expected times for both the parallel and the sequential algorithm as speedup

$$s_m = \frac{\mathrm{E}(T_1)}{\mathrm{E}(T_m)} \,.$$

Note that $T_1$ and $T_m$ might have very dissimilar probability distributions. Even when both are re-scaled appropriately to obtain the best possible match between the two, they might still have different shapes and different variances. In some cases it might make sense to consider the median or other statistics instead of the expectation.

According to the speedup $s_m$ we distinguish the following cases:

- *Sublinear speedup*: if $s_m < m$ we speak of a *sublinear speedup*. This implies that the total computation time across all machines is larger than the total computation time of the single machine (assuming no idle times in the parallel algorithm).
- *Linear speedup*: the case $s_m = m$ is known as *linear speedup*. There, the parallel and the sequential algorithm have the same total time. This outcome is very desirable as it means that parallelization does not come at a cost. There is no noticeable overhead in the parallel algorithm.
- *Superlinear speedup*: if $s_m > m$ we have a *superlinear speedup*. The total computation time of the parallel algorithm is even smaller than that of the single machine. This case is considered in more detail in the following section.

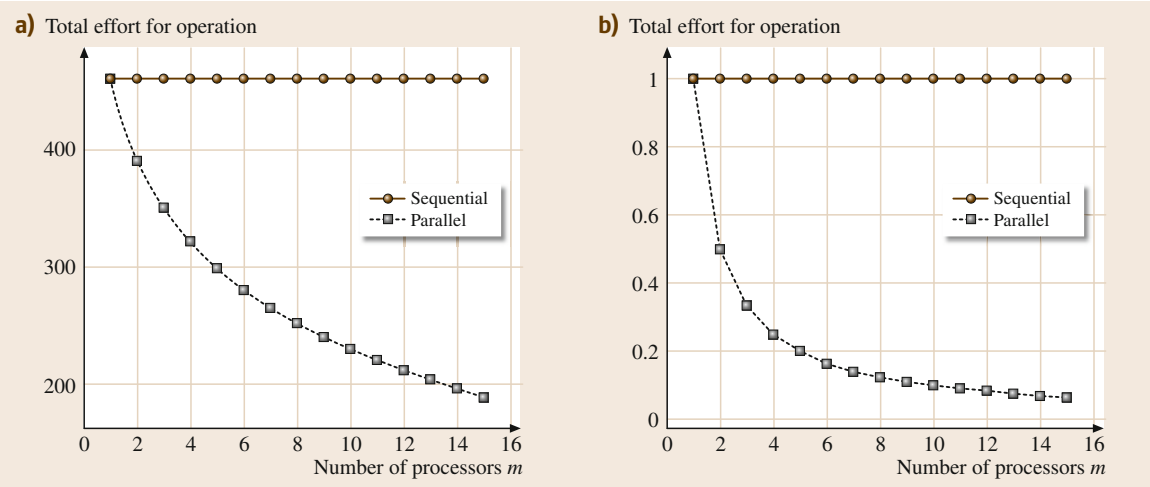


**Fig. 46.5a,b** Total effort for executing an operation on a single, panmictic population of size $\mu = 100$ (sequential algorithm) and a parallel algorithm with $m$ processors and $m$ subpopulations of size $\mu/m = 100/m$ each. The effort on a population of size $n$ is assumed to be $n \ln n$ **(a)** and $n^2$ **(b)**. Note that no overhead is considered for the parallel algorithm

Speedup is the best known measure, but not the only one used regularly. For the sake of completeness, we mention other measures. The *efficiency* is a normalization of the speedup

$$e_m = \frac{s_m}{m} .$$

Obviously, $e_m = 1$ is equivalent to a linear speedup. Lower efficiencies correspond to sublinear speedups, higher ones to superlinear speedups.

Another measure is called *incremental efficiency* and it measures the speedup when moving from $m-1$ processors to $m$ processors

$$\mathrm{ie}_m = \frac{(m-1)\cdot \mathrm{E}(T_{m-1})}{m \cdot \mathrm{E}(T_m)} .$$

There is also a generalized form where $m-1$ is replaced by $m' < m$ in the above formula. This reflects the speedup when going from $m'$ processors to $m$ processors.

## 46.2.2 Superlinear Speedups

At first glance superlinear speedups seem astonishing. How can a parallel algorithm have a smaller total computation time than a sequential counterpart? After all, parallelization usually comes with significant overhead that slows down the algorithm. The existence of superlinear speedups has been discussed controversially in the literature. However, there are convincing reasons why a superlinear speedup might occur.

*Alba* [46.14] mentions physical sources as one possible reason. A parallel machine might have more resources in terms of memory or caches. When moving from a single machine to a parallel one, the algorithm might – purposely or not – make use of these additional resources. Also, each machine might only have to deal with smaller data packages. It might be that the smaller data fits into the cache while this was not the case for the single machine. This can make a significant performance difference.

When comparing a single panmictic population against smaller subpopulations, it might be easier to deal with the subpopulations. This holds even when the total population sizes of both systems are the same. In particular, a parallel system has an advantage if operations need time which grows faster than linearly with the size of the (sub)population.

We give two illustrative examples. Compare a single panmictic population of size $\mu$ with $m$ subpopulations of size $\mu/m$ each. Some selection mechanisms, like ranking selection, might have to sort the individuals in the population according to their fitness. In a straightforward implementation one might use well-known sorting algorithms such as (randomized) *QuickSort*, *MergeSort*, or *HeapSort*. All of these are known to take time $\Theta(n \ln n)$ for sorting $n$ elements, on average. Let us disregard the hidden constant and the randomness of

randomized *QuickSort* and assume that the time is precisely $n \ln n$.

Now the effort of sorting the panmictic population is $\mu \ln \mu$. The total effort for sorting $m$ populations of size $\mu/m$ each is

$$m \cdot \mu/m \cdot \ln(\mu/m) = \mu \cdot \ln(\mu/m) = \mu \ln(\mu) - \mu \cdot \ln(m) .$$

So, the parallel system executes this operation faster, with a difference of $\mu \cdot \ln(m)$ time steps in terms of the total computation time.

This effect becomes more pronounced the more expensive operations are used (with respect to the population size). Assume that some selection mechanism or diversity mechanism is used, which compares every individual against every other one. Then the effort for the panmictic population is roughly $\mu^2$ time steps. However, for the parallel EA and its subpopulations the total effort would only be

$$m \cdot (\mu/m)^2 = \mu^2/m .$$

This is faster than the panmictic EA by a factor of $m$.

The above two growth curves are actually very typical running times for operations that take more than linear time. A table with time bounds for common selection mechanisms can be found in *Goldberg* and *Deb* [46.16]. Figure 46.5 shows plots for the total effort in both scenarios for a population size of $\mu = 100$. One can see that even with a small number of processors the total effort decreases quite significantly. To put this into perspective, most operations require only linear time. Also the overhead by parallelization was not accounted for. However, the discussion gives some hints as to why the execution time for smaller subpopulations can decrease significantly in practice.

## 46.3 On the Spread of Information in Parallel EAs

In order to understand how parallel EAs work, it is vital to get an idea on how quickly information is propagated. The spread of information is the most distinguishing aspect of parallel EAs, particularly distributed EAs. This includes island models and cellular EAs. Many design choices can tune the speed at which information is transmitted: the topology, the migration interval, the number of migrants, and the policies for emigration and immigration.

### 46.3.1 Logistic Models for Growth Curves

Many researchers have turned to investigating the selection pressure in distributed EAs in a simplified model. Assume that in the whole system we only have two types of solutions: current best individuals and worse solutions. No variation is used, i. e., we consider EAs using neither mutation nor crossover. The question is the following. Using only selection and migration, how long does it take for the best solutions to take over the whole system? This time, starting from a single best solution, is referred to as *takeover time*.

It is strongly related to the study of *growth curves*: how the number of best solutions increases over time. The takeover time is the first point of time at which the number of best solutions has grown to the whole population.

Growth curves are determined by both inter-island dynamics and intra-island dynamics: how quickly current best solutions spread in one island's population, and how quickly they populate neighbored islands, until the whole topology is taken over. Both dynamics are linked: intra-island dynamics can have a direct impact on inter-island dynamics as the fraction of best individuals can decide how many (if any) best individuals emigrate.

For intra-island dynamics one can consider results on panmictic EAs. Logistic curves have been proposed and found to fit simulations of takeover times very well for common selection schemes [46.16]. These curves are defined by the following equation. If $P(t)$ is the proportion of best individuals in the population at time $t$, then

$$P(t) = \frac{1}{1 + \left(\frac{1}{P(0)} - 1\right) e^{-at}} ,$$

where $a$ is called the growth coefficient. One can see that the proportion of best individuals increases exponentially, but then the curve saturates as the proportion approaches 1.

*Sarma* and *De Jong* [46.17] considered growth curves in cellular EAs. They presented a detailed empirical study of the effects of the neighborhood size and

the shape of the neighborhood for different selection schemes. They showed that logistic curves as defined above can model the growth curves in cellular EAs reasonably well.

*Alba* and *Luque* [46.18] proposed a logistic model called LOG tailored towards distributed EAs with periodic migration. If $\tau$ denotes the migration interval and $m$ is the number of islands, then

$$P_{\mathrm{LOG}}(t) = \sum_{i=0}^{m-1} \frac{1/m}{1 + a \cdot e^{-b(t-\tau \cdot i)}} \,.$$

In this model $a$ and $b$ are adjustable parameters. The model counts subsequent increases of the proportion of best individuals during migrations. However, it does not include any information about the topology and the authors admit that it only works appropriately on the ring topology [46.19, Section 4.2]. They, therefore, present an even more detailed model called TOP, which includes the diameter $\mathrm{diam}(T)$ of the topology $T$.

$$P_{\mathrm{TOP}}(t) = \sum_{i=0}^{\mathrm{diam}(T)-1} \frac{1/m}{1 + a \cdot e^{-b(t-\tau \cdot i)}} + \frac{m - \mathrm{diam}(T)/m}{1 + a \cdot e^{-b(t-\tau \cdot \mathrm{diam}(T))}} \,.$$

Simulations show that this model yields very accurate fits for ring, star, and complete topologies [46.19, Section 4.3].

*Luque* and *Alba* [46.19, Section 4.3] proceed by analyzing the effect of the migration interval and the number of migrants. With a large migration interval, the growth curves tend to make jumps during migration and flatten out quickly to form plateaus during periods without migration. The resulting curves look like step functions, and the size of these steps varies with the migration interval.

Varying the number of migrants changes the slope of these steps. A large number of migrants has a better chance of transmitting best individuals than a small number of migrants. However, the influence of the number of migrants was found to be less drastic than the impact of the migration interval. When a medium or large migration frequency is used, the impact of the number of migrants is negligible [46.19, Section 4.5]. The same conclusion was made earlier by *Skolicki* and *De Jong* [46.20].

Luque and Alba also presented experiments with a model based on the *Sprave's* hypergraph formulation of distributed EAs [46.13]. This model gave a better fit than the simple logistic model LOG, but it was less accurate than the model TOP that included the diameter.

For the sake of completeness, we also mention that *Giacobini* et al. [46.21] proposed an improved model for asynchronous cellular EAs, which is not based on logistic curves.

### 46.3.2 Rigorous Takeover Times

*Rudolph* [46.22, 23] rigorously analyzed takeover times in panmictic populations, for various selection schemes. In [46.22] he also dealt with the probability that the best solution takes over the whole population; this is not evident for non-elitistic algorithms. In [46.23] *Rudolph* considered selection schemes made elitistic by undoing the last selection in case the best solution would become extinct otherwise. Under this scheme the expected takeover time in a population of size $\mu$ is $O(\mu \log \mu)$.

In [46.24] *Rudolph* considered spatially structured populations in a fine-grained model. Each population has size 1, therefore vertices in the migration topology can be identified with individuals. Migration happens in every generation. Assume that initially only one vertex $i$ in the topology is a best individual. If in every generation each non-best vertex is taken over by the best individual in its neighborhood, then the takeover time from vertex $i$ equals

$$\max_{j \in V} \mathrm{dist}(i,j) \,,$$

where $V$ is the set of vertices and $\mathrm{dist}(i,j)$ denotes the graph distance, the number of edges on a shortest path from $i$ to $j$.

Rudolph defines the takeover time in a setting where the initial best solution has the same chance of evolving at every vertex. Then

$$\frac{1}{|V|} \sum_{i \in V} \max_{j \in V} \mathrm{dist}(i,j)$$

is the expected takeover time if, as above, best solutions are always propagated to their neighbors with probability 1. If this probability is lower, the expected takeover time might be higher. The above formula still represents a lower bound. Note that in non-elitist EAs it is possible that all best solutions might get lost, leading to a positive extinction probability [46.24].

Note that $\max_{j \in V} \mathrm{dist}(i,j)$ is bounded by the *diameter* of the topology. The diameter is hence a trivial lower bound on the takeover times. *Rudolph* [46.24]

conjectures that the diameter is more important than the selection mechanism used in the distributed EA.

In [46.25] the author generalizes the above arguments to coarse-grained models. Islands can contain larger populations and migration happens with a fixed frequency. In his model the author assumes that in each island new best individuals can only be generated by immigration. Migration always communicates best individuals. Hence, the takeover time boils down to a deterministic time until the last island has been reached, plus a random component for the time until all islands have been taken over completely.

*Rudolph* [46.25] gives tight bounds for unidirectional rings, based on the fact that each island with a best individual will send one such individual to each neighbored island. Hence, on the latter island the number of best individuals increases by 1, unless the island has been taken over completely. For more dense topologies he gives a general upper bound, which may not be tight for all graphs. If there is an island that receives best individuals from $k > 1$ other islands, the number of best individuals increases by $k$. (The number $k$ could even increase over time.) It was left as an open problem to derive more tight bounds for interesting topologies other than unidirectional rings.

Other researchers followed up on Rudolph's seminal work. *Giacobini* et al. [46.26] presented theoretical and empirical results for the selection pressure on ring topologies, or linear cellular EAs. *Giacobini* et al. [46.27] did the same for toroidal cellular EAs. In particular, they considered takeover times for asynchronous cellular EAs, under various common update schemes. Finally, *Giacobini* et al. investigated growth curves for small-world graphs [46.9].

The assumption from Rudolph's model that only immigration can create new best individuals is not always realistic. If standard mutation operators are used, there is a constant probability of creating a clone of a selected parent simply by not flipping any bits. This can lead to a rapid increase in the number of high-fitness individuals.

This argument on the takeover of good solutions in panmictic populations has been studied as part of rigorous runtime analyses of population-based EAs. *Witt* [46.28] considered a simple $(\mu + 1)$ EA with uniform parent selection, standard bit mutations, no crossover, and cut selection at the end of the generation. From his work it follows that good solutions take over the population in expected time $O(\mu \log \mu)$. More precisely, if currently there is at least one individual with current best fitness $i$, then after $O(\mu \log \mu)$ generations all individuals in the population will have fitness $i$ at least.

*Sudholt* [46.29, Lemma 2] extended these arguments to a $(\mu + \lambda)$ EA and proved an upper bound of $O(\mu/\lambda \cdot \log \mu + \log \mu)$. Note that, in contrast to other studies of takeover times, both results apply *real* EAs that actually use mutation. Extending these arguments to distributed EAs is an interesting topic for future work.

### 46.3.3 Maximum Growth Curves

Now, we consider inter-island dynamics in more detail. Assume for simplicity that intra-island takeover happens quickly: after each migration transmitting at least one best solution, the target island is completely taken over by best solutions before the next migra-

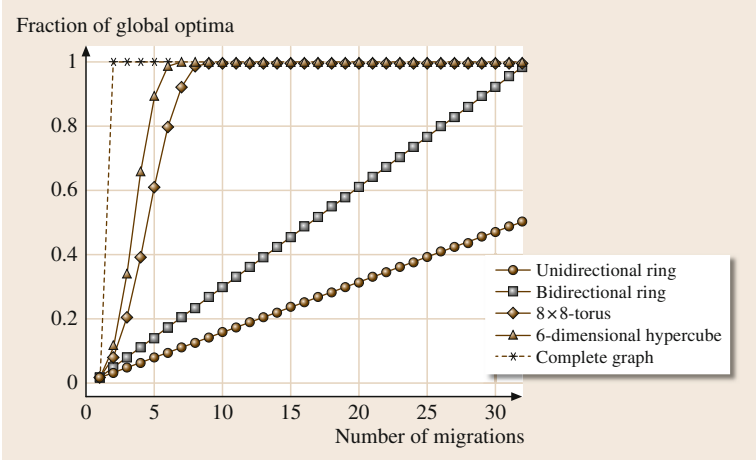


**Fig. 46.6** Plots of growth curves in an island model with 64 islands. We assume that in between two migrations all islands containing a current best solution completely take over all neighbored islands in the topology. Initially, one island contains a current best solution and all other islands are worse. The *curves* show the fraction of current best solutions in the system for different topologies: a unidirectional ring, a bidirectional ring, a square torus, a hypercube, and a complete graph

tion. We start with only one island containing a best solution, assuming that all individuals on this island are best solutions. We call such an island an optimal island. If migrants are not subject to variation while emigrating or immigrating, we will always select best solutions for migration and, hence, successfully transmit best solutions.

These assumptions give rise to a deterministic spread of best solutions: after each migration, each optimal island will turn all neighbored islands into optimal islands. This is very similar to *Rudolph's* model [46.25], but it also accounts for a rapid intra-island takeover in between migrations.

We consider growth curves on various graph classes: unidirectional and bidirectional rings, square torus graphs, hypercubes, and complete graphs. Figure 46.6 shows these curves for all these graphs on 64 vertices. The torus graph has side lengths $8 \times 8$. The hypercube has dimension 6. Each vertex has a label made of 6 bits. All possible values for this bit string are present in the graph. Two vertices are neighbored if their labels differ in exactly one bit.

For the unidirectional ring, after $i-1$ migrations we have exactly $i$ optimal islands, if $i \leq m$. The growth curve is, therefore, linear. For the bidirectional ring information spreads twice as fast as it can spread in two directions. After $i-1$ migrations we have $2i-1$ optimal islands if $2i-1 \leq m$.

The torus allows communication in two dimensions. After one migration there are $1+4=5$ optimal islands. After two migrations this number is $1+4+8$, and after three migrations it is $1+4+8+12$. In general, after $i-1$ migrations we have

$$1+\sum_{j=1}^{i-1} 4j = 1+2i(i-1) = 1+2i^2-2i$$

optimal islands, as long as the optimal islands can freely spread out in all four directions, north, south, west, and east. At some point the ends of the region of optimal islands will meet, i. e., the northern tip meets the southern one and the same goes for west and east. Afterwards, we observe regions of non-optimal islands that constantly shrink, until all islands are optimal. The growth curve for the torus is hence quadratic at first and then it starts to saturate. The deterministic growth on torus graphs was also considered in [46.30].

For the hypercube, we can without loss of generality assume that the initial optimal island has a label containing only zeros. After one migration all islands whose label contains a single one become optimal. After two migrations the same holds for all islands with two ones, and so on. The number of optimal islands after $i$ migrations in a $d$-dimensional hypercube (i. e., $m = 2^d$) is hence $\sum_{j=0}^{i} \binom{d}{j}$. This number is close to $d^j$ during the first migrations and then at some point starts to saturate. The complete graph is the simplest one to analyze here as it will be completely optimal after one migration.

These arguments and Fig. 46.6 show that the growth curves can depend tremendously on the migration topology. For sparse topologies like rings or torus graphs, in the beginning the growth is linear or quadratic, respectively. This is much slower than the exponential growth observed in logistic curves. Furthermore, for the ring there is no saturation; linear curves are quite dissimilar to logistic curves.

This suggests that logistic curves might not be the best models for growth curves across all topologies. The plots by *Luque* and *Alba* [46.19, Section 4.3] show a remarkably good overall fit for their TOP model. However, this might be due to the optimal choice of the parameters $a$ and $b$ and the fact that logistic curves are easily adaptable to various curves of roughly similar shape. We believe that it is possible to derive even more accurate models for common topologies, based on results by *Giacobini* et al. [46.9, 26, 27]. This is an interesting challenge for future work.

### 46.3.4 Propagation

So far, we have only considered models where migration always successfully transmits best individuals. For non-trivial selection of emigrants, this is not always given. Also if crossover is used during migration, due to disruptive effects migration is not always successful. If we consider randomized migration processes, things become more interesting.

*Rowe* et al. [46.31] considered a model of *propagation* in networks. Consider a network where vertices are either informed or not. In each round, each informed vertex tries to inform each of its neighbors. Every such trial is successful with a given probability $p$, and then the target island becomes informed. These decisions are made independently. Note that an uninformed island might obtain a probability larger than $p$ of becoming informed, in case several informed islands try to inform it. The model is inspired by models from epidemiology; it can be used to model the spread of a disease.

The model of propagation of information directly applies to our previous setting where the network is the migration topology and $p$ describes the probability of successfully migrating a current best solution. Note that when looking for estimations of growth curves and upper bounds on the takeover time, we can assume that $p$ is a lower bound on the actual probability of a successful transmission. Then the model becomes applicable to a broader range of settings, where islands can have different transmission probabilities.

On some graphs like unidirectional rings, we can just multiply our growth curves by $p$ to reflect the expected number of optimal islands after a certain time. It then follows that the time for taking over all $m$ islands is by a factor of $1/p$ larger than in the previous, deterministic model.

However, this reasoning does not hold in general. Multiplying the takeover time in the deterministic setting by $1/p$ does not always give the expected takeover time in the random model. Consider a star graph (or *hub*), where initially only the center vertex is informed. In the deterministic case $p = 1$, the takeover time is clearly 1. However, if $0 < p < 1$, the time until the last vertex is informed is given by the maximum of $n-1$ independent geometric distributions with parameter $p$. For constant $p$, this time is of order $\Theta(\log n)$, i. e., the time until the last vertex is informed is much larger than the expected time for any specific island to be informed.

*Rowe* et al. [46.31] presented a detailed analysis of hubs. They also show how to obtain a general upper bound that holds for all graphs. For every graph $G$ with $n$ vertices and diameter $\mathrm{diam}(G)$ the expected takeover time is bounded by

$$O\left(\frac{\mathrm{diam}(G)+\log n}{p}\right).$$

Both terms $\mathrm{diam}(G)$ and $\log n$ make sense. The diameter describes what distance needs to be overcome in order to inform all vertices in the network. The factor $1/p$ gives the expected time until a next vertex is informed, assuming that it has only one informed neighbor. We also obtain $\mathrm{diam}(G)$ (without a factor $1/p$) as a lower bound on the takeover time. The additive term $+\log n$ is necessary to account for a potentially large variance, as seen in the example for star graphs.

If the diameter of the graph is at least $\Omega(\log n)$, we can drop the $+\log n$-term in the asymptotic bound, leading to an upper bound of $O(\mathrm{diam}(G)/p)$.

Interestingly, the concept of propagation also appears in other contexts. When solving shortest paths problems in graphs, metaheuristics like evolutionary algorithms [46.32–34] and ant colony optimization (ACO) [46.35, 36] tend to propagate shortest paths through the graph. In the single-source shortest paths problem (SSSP) one is looking for shortest paths from a source vertex to all other vertices of the graph. The EAs and ACO algorithms tend to find shortest paths first for vertices that are *close* to the source, in a sense that their shortest paths only contain few edges. If these shortest paths are found, it enables the algorithm to find shortest paths for vertices that are further away.

When a shortest paths to vertex $u$ is found and there is an edge $\{u, v\}$ in the graph, it is easy to find a shortest path for $v$. In the case of evolutionary algorithms, an EA only needs to assign $u$ as a predecessor of $v$ on the shortest path in a lucky mutation in order to find a shortest path to $v$. In the case of ACO, pheromones enable an ant to follow pheromones between the source and $u$, and so it only has to decide to travel between $u$ and $v$ to find a shortest path to $v$, with good probability.

*Doerr* et al. [46.34, Lemma 3] used tail bounds to prove that the time for propagating shortest paths with an EA is highly concentrated. If the graph has diameter $\mathrm{diam}(G) \geq \log n$, the EA with high probability finds all shortest paths in time $O(\mathrm{diam}(G)/p)$, where $p = \Theta(n^{-2})$ in this case. This result is similar to the one obtained by *Rowe* et al. [46.31]; asymptotically, both bounds are equal. However, the result by *Doerr* et al. [46.33] also allows for conclusions about growth curves.

*Lässig* and *Sudholt* [46.6, Theorem 3] introduced yet another argument for the analysis of propagation times. They considered *layers* of vertices. The $i$-th layer contains all vertices that have shortest paths of at most $i$ edges, and that are not on any smaller layer. They bound the time until information is propagated throughout all vertices of a layer. This is feasible since all vertices in layer $i$ are informed with probability at least $p$ if all vertices in layers $1, \ldots, i-1$ are informed. If $n_i$ is the number of vertices in layer $i$, the time until the last vertex in this layer is informed is $O(n_i \cdot \log n_i)$. This gives a bound for the total takeover time of $O(\mathrm{diam}(G) \cdot \ln(en/\mathrm{diam}(G)))$. For small ($\mathrm{diam}(G) = O(1)$) or large ($\mathrm{diam}(G) = \Omega(n)$) diameters, we get the same asymptotic bound as before. For other values it is slightly worse.

However, the layering of vertices allows for inclusion of intra-island effects. Assume that the transmis-

sion probability $p$ only applies once islands have been taken over (to a significantly large degree) by best individuals. This is a realistic setting as with only a single best individual the probability of selecting it for emigration (or pollination, to be precise) might be very small. If all islands need time $T_{\text{intra}}$ in order to reach this stage after the first best individual has reached the island, we obtain an upper bound of

$$O(\text{diam}(G) \cdot \ln(en/\text{diam}(G))) + \text{diam}(G) \cdot T_{\text{intra}}$$

for the takeover time.

## 46.4 Examples Where Parallel EAs Excel

Parallel EAs have been applied to a very broad range of problems, including many NP-hard problems from combinatorial optimization. The present literature is immense; already early surveys like the one by *Alba* and *Troya* [46.37] present long lists of applications of parallel EAs. Further applications can be found in [46.38–40]. Research on and applications of parallel metaheuristics has increased in recent years, due to the emergence of parallel computer architectures.

*Crainic* and *Hail* [46.41] review applications of parallel metaheuristics, with a focus on graph coloring, partitioning problems, covering problems, Steiner tree problems, satisfiability problems, location and network design, as well as the quadratic assignment problems with its famous special cases: the traveling salesman problem and vehicle routing problems. *Luque* and *Alba* [46.19] present selected applications for natural language tagging, the design of combinatorial logic circuits, the workforce planning problem, and the bioinformatics problem of assembling DNA fragments.

The literature is too vast to be reviewed in this section. Also, for many hard practical problems it is often hard to determine the effect that parallelization has on search dynamics. The reasons behind the success of parallel models often remain elusive. We follow a different route and describe theoretical studies of evolutionary algorithms where parallelization was proven to be helpful. This concerns illustrative toy functions as well as problems from combinatorial optimization. All following settings are well understood and allow us to gain insights into the effect of parallelization. We consider parallel variants of the most simple evolutionary algorithm called $(1+1)$ evolutionary algorithm, shortly $(1+1)$ EA. It is described in Algorithm 46.3 and it only uses mutation and selection in a population containing just one current search point. We are interested in the *optimization time*, defined as the number of generations until the algorithm first finds a global optimum. Unless noted otherwise, we consider pseudo-Boolean optimization: the search space contains all bit strings of length $n$ and the task is to maximize a function $f: \{0,1\}^n \to \mathbb{R}$. We use the common notation $x = x_1 \ldots x_n$ for bit strings.

*Algorithm 46.3* $(1+1)$ *EA for maximizing* $f: \{0,1\}^n \to \mathbb{R}$

1: Initialize $x \in \{0,1\}^n$ uniformly at random.
2: **loop**
3: Create $x'$ by copying $x$ and flipping each bit independently with probability $1/n$.
4: **if** $f(x') \geq f(x)$ **then** $x := x'$.
5: **end loop**

The presentation in this section is kept informal. For theorems with precise results, including all preconditions, we refer to the respective papers.

### 46.4.1 Independent Runs

Independent runs prove useful if the running time has a large variance. The reason is that the optimization time equals the time until *the fastest* run has found a global optimum.

The variance can be particularly large in the case when the objective function yields local optima that are very hard to overcome. Bimodal functions contain two local optima, and typically only one is a global optimum. One such example was already analyzed theoretically in the seminal runtime analysis paper by *Droste* et al. [46.42].

We review the analysis of a similar function that leads to a simpler analysis. The function *TwoMax* was considered by *Friedrich* et al. [46.43] in the context of diversity mechanisms. It is a function of unitation: the fitness only depends on the number of bits set to 1. The function contains two symmetric slopes that increase linearly with the distance to $n/2$. Only one of these slopes leads to a global optimum. Formally, the function

is defined as the maximum of $OneMax := \sum_{i=1}^{n} x_i$ and its symmetric cousin $ZeroMax := \sum_{i=1}^{n} (1-x_i)$, with an additional fitness bonus for the all-ones bit string

$$TwoMax(x) := \max \left\{ \sum_{i=1}^{n} x_i, \sum_{i=1}^{n} (1-x_i) \right\} + \prod_{i=1}^{n} x_i \,.$$

See Fig. 46.7 for a sketch.

The $(1+1)$ EA reaches either a local optimum or a global optimum in expected time $O(n \log n)$. Due to the perfect symmetry of the function on the remainder of the search space, the probability that this is the global optimum is exactly $1/2$. If a local optimum is reached, the $(1+1)$ EA has to flip all bits in one mutation in order to reach the global optimum. The probability for this event is exactly $n^{-n}$.

The authors consider deterministic crowding [46.43] in a population of size $\mu$ as a diversity mechanism. It has the same search behavior as $\mu$ independent runs of the $(1+1)$ EA, except that the running time is counted in a different way. Their result directly transfers to this parallel model. The only assumption is that the number of independent runs is polynomially bounded in $n$.

The probability of finding a global optimum after $O(n \log n)$ generations of the parallel system is amplified to $1-2^{-\mu}$. This means that only with probability $2^{-\mu}$ we arrive at a situation where the parallel EA needs to escape from a local optimum. When all $m$ islands are in this situation, the probability that at least one island makes this jump in one generation is at most

$$1-(1-n^{-n})^m = \Theta(m \cdot n^{-n}) \,,$$

where the last equality holds since $m$ is asymptotically smaller than $n^n$.

This implies that the expected number of generations of a parallel system with $m$ independent runs is

$$O(n \log n) + 2^{-m} \cdot \Theta\left(\frac{n^n}{m}\right) \,.$$

We can see from this formula that the number of runs $m$ has an immense impact on the expected running time. Increasing the number of runs by 1 decreases the second summand by more than a factor of 2. The speedup is, therefore, exponential, up to a point where the running time is dominated by the first term $O(n \log n)$. Note in particular that $\log(n^n) = n \log n$ processors are sufficient to decrease the expected running time to $O(n \log n)$.

This is a very simple example of a superlinear speedup, with regard to the optimization time.

The observed effects also occur in combinatorial optimization. *Witt* [46.44] analyzed the $(1+1)$ EA on the NP-hard *PARTITION* problem. The task can be regarded as scheduling on two machines: given a sequence of jobs, each with a specific effort, the goal is to distribute the jobs on two machines to that the largest execution time (the makespan) is minimized.

On worst-case instances the $(1+1)$ EA has a constant probability of getting stuck in a bad local optimum. The expected time to find a solution with a makespan of less than $(4/3-\varepsilon) \cdot \mathrm{OPT}$ is $n^{\Omega(n)}$, where $\varepsilon > 0$ is an arbitrary constant and OPT is the value of the optimal solution.

However, if the $(1+1)$ EA is lucky, it can, indeed, achieve a good approximation of the global optimum. Assume we are aiming at a solution with a makespan of at most $(1+\varepsilon) \cdot \mathrm{OPT}$, for some $\varepsilon > 0$ we can choose. Witt's analysis shows that then $2^{(e \log e + e) \cdot \lceil 2/\varepsilon \rceil \ln(4/\varepsilon) + O(1/\varepsilon)}$ parallel runs output a solution of this quality with probability at least $3/4$. (This probability can be further amplified quite easily by using more runs.) Each run takes time $O(n \ln(1/\varepsilon))$. The parallel model represents what is known as a *polynomial-time randomized approximation scheme* (PRAS). The desired approximation quality $(1+\varepsilon)$ can be specified, and if $\varepsilon$ is fixed, the total computation time is bounded by a polynomial in $n$. This was the first example that parallel runs of a randomized search heuristics constitute a PRAS for an NP-hard problem.

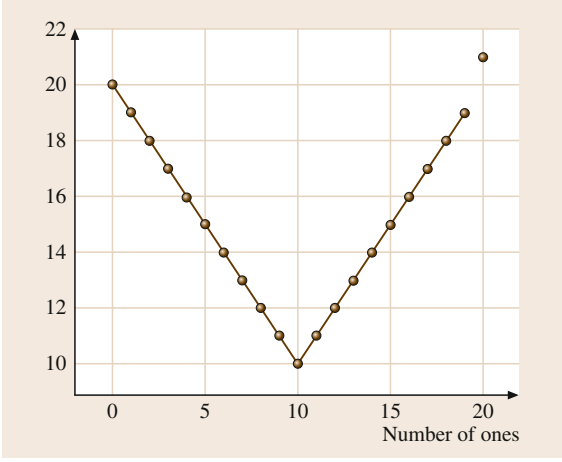


**Fig. 46.7** Plots of the bimodal function *TwoMax* as defined in [46.43]

### 46.4.2 Offspring Populations

Using offspring populations in a master–slave architecture can decrease the parallel running time and lead to a speedup. We will discuss this issue further in Sect. 46.5 as offspring populations are very similar to island models on complete topologies. For now, we present one example where offspring populations decrease the optimization time very drastically.

*Jansen* et al. [46.45] compared the $(1+1)$ EA against a variant $(1+\lambda)$ EA that creates $\lambda$ offspring in parallel and compares the current search point against the best offspring. They constructed a function SufSamp where offspring populations have a significant advantage. We refrain from giving a formal definition, but instead describe the main ideas. The vast majority of all search points tend to lead an EA towards the start of a path through the search space. The points on this path have increasing fitness, thus encouraging an EA to follow it. All points outside the path are worse, so the EA will stay on the path.

The path leads to a local optimum at the end. However, the function also includes a number of smaller paths that branch off the main path, see Fig. 46.8. All these paths lead to global optima, but they are difficult to discover. This makes a difference between the $(1+1)$ EA and the $(1+\lambda)$ EA for sufficiently large $\lambda$. The $(1+1)$ EA typically follows the main path without discovering the smaller paths branching off. At the end of the main path it thus becomes stuck in a local optimum. The analysis in [46.45] shows that the $(1+1)$ EA needs superpolynomial time, with high probability.

Contrarily, the $(1+\lambda)$ EA performs a more thorough search as it progresses on the main path. The many offspring tend to discover at least one of the smaller branches. The fitness on the smaller branches is larger than the fitness of the main path, so the EA will move away from the main path and follow a smaller path. It then finds a global optimum in polynomial time, with high probability.

Interestingly, this construction can be easily adapted to show an opposite result. We replace the local optimum at the end of the main path by a global optimum and replace all global optima at the end of the smaller branches by local optima. This yields another function SufSamp′, also shown in Fig. 46.8. By the same reasoning as above, the $(1+\lambda)$ EA will become stuck and the $(1+1)$ EA will find a global optimum in polynomial time, with high probability.

While the example is clearly constructed and artificial, it can be seen as a cautionary tale. The reader might be tempted to think that using offspring populations instead of creating a single offspring can never increase the number of *generations* needed to find the optimum. After all, evolutionary search with offspring population is more intense and improvements can be found more easily. As we focus on the number of generations (and do not count the effort for creating $\lambda$ offspring), it is tempting to claim that offspring populations are never disadvantageous.

The second example shows that this claim – however obvious it may seem – does not hold for general problem classes. Note that this statement is also implied by the well-known *no free lunch theorems* [46.46], but the above results are much stronger and more concrete.

### 46.4.3 Island Models

The examples so far have shown that a more thorough search – by independent runs or increased sampling of offspring – can lead to more efficient running times. *Lässig* and *Sudholt* [46.3] presented a first example where communication makes the difference between exponential and polynomial running times, in a typical run. They constructed a family of problems called $\mathrm{LOLZ}_{n,z,b,\ell}$ where a simple island model finds the optimum in polynomial time, with high probability. This holds for a proper choice of the migration interval and any migration topology that is not too sparse. The islands run $(1+1)$ EAs, hence the island model resembles a fine-grained model.

Contrarily, both a panmictic population as well as independent islands need exponential time, with high probability. This shows that the weak speedup versus panmixia is superlinear, even exponential (when considering speedups with respect to the typical running

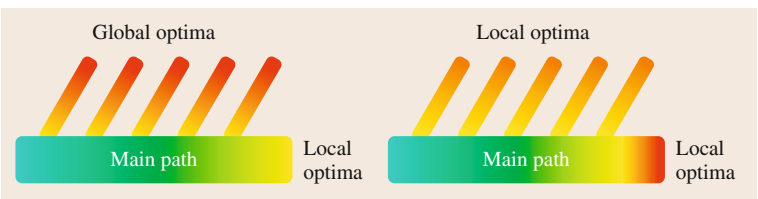


**Fig. 46.8** Sketches of the functions SufSamp (*left*) and SufSamp′ (*right*). The fitness is indicated by the *color*

**Table 46.1** Examples of solutions for the function LOLZ with four blocks and $z = 3$, along with their fitness values. All blocks have to be optimized from left to right. The sketch shows in bold all bits that are counted in the fitness evaluation. Note how in $x_3$ in the third block only the first $z = 3$ zeros are counted. Further 0-bits are ignored. The only way to escape from this local optimum is to flip all $z$ 0-bits in this block simultaneously

| | | | | | |
|---|---|---|---|---|---|
| $x_1$ | **1111**0011 | 11010100 | 11010110 | 01011110 | $\text{LOLZ}(x_1) = 4$ |
| $x_2$ | **11111111** | **11**010100 | 11010110 | 01011110 | $\text{LOLZ}(x_2) = 10$ |
| $x_3$ | **11111111** | **11111111** | **000**00110 | 01011110 | $\text{LOLZ}(x_3) = 19$ |

time instead of the expected running time). Unlike previous examples, it also shows that more sophisticated means of parallelization can be better than independent runs.

The basic idea of this construction is as follows. An EA can increase the fitness of its current solutions by gathering a prefix of bits with the same value. Generally, a prefix of $i$ leading ones yields the same fitness as a prefix of $i$ leading zeros. The EA has to make a decision whether to collect leading ones (LOs) or leading zeros (LZs). This not only holds for the $(1 + 1)$ EA but also for a (not too large) panmictic population as genetic drift will lead the whole population to either leading ones or leading zeros.

In the beginning, both decisions are symmetric. However, after a significant prefix has been gathered, symmetry is broken: after the prefix has reached a length of $z$, $z$ being a parameter of the function, only leading ones lead to a further fitness increase. If the EA has gone for leading zeros, it becomes stuck in a local optimum. The parameter $z$ determines the difficulty of escaping from this local optimum.

This construction is repeated on several blocks of the bit string that need to be optimized one-by-one. Each block has length $\ell$. Only if the right decision towards the leading ones is made on the first block, can the block be filled with further leading ones. Once the first block contains only leading ones, the fitness depends on the prefix in the second block, and a further decision between leading ones and leading zeros needs to be made. Figure 46.1 illustrates the problem definition.

So, the problem requires an EA to make several decisions in succession. The number of blocks, $b$, is another parameter that determines how many decisions need to be made. Panmictic populations will sooner or later make a wrong decision and become stuck in some local optimum. If $b$ is not too small, the same holds for independent runs.

However, an island model can effectively communicate the right decisions on blocks to other islands. Islands that have become stuck in a local optimum can be taken over by other islands that have made the correct decision. These dynamics make up the success of the island model as it can be shown to find global optima with high probability. A requirement is, though, that the migration interval is carefully tuned so that migration only transmits the right information. If migration happens before the symmetry between leading ones and leading zeros is broken, it might be that islands with leading zeros take over islands with leading ones. *Lässig* and *Sudholt* [46.3] give sufficient conditions under which this does not happen, with high probability.

An interesting finding is also how islands can regain independence. During migration, genetic information about future blocks is transmitted. Hence, after migration all islands contain the same genotype on future blocks. This is a real threat as this dependence might imply that all islands make the same decision after moving on to the next block. Then all diversity would be lost.

However, under the conditions given in [46.3] there is a period of independent evolution following migration, before any island moves on to a new block. During this period of independence, the genotypes of future blocks are subjected to random mutations, independently for each island. The reader might think of moving particles in some space. Initially, all bits are in the same position. However, then particles start moving around randomly. Naturally, they will spread out and separate from one another. After some time the distribution of particles will resemble a uniform distribution. In particular, an observer would not be able to distinguish whether the positions of particles were obtained by this random process or by simply drawing them from a uniform distribution.

The same effect occurs with bits of future blocks; after some time all bits of a future block will be indistinguishable from a random bit string. This shows that independence can not only be gained by independent runs, but also by periods of independent evolution. One could say that the island model combines the advantages of two worlds: independent evolution and selection pressure through migration. The island model

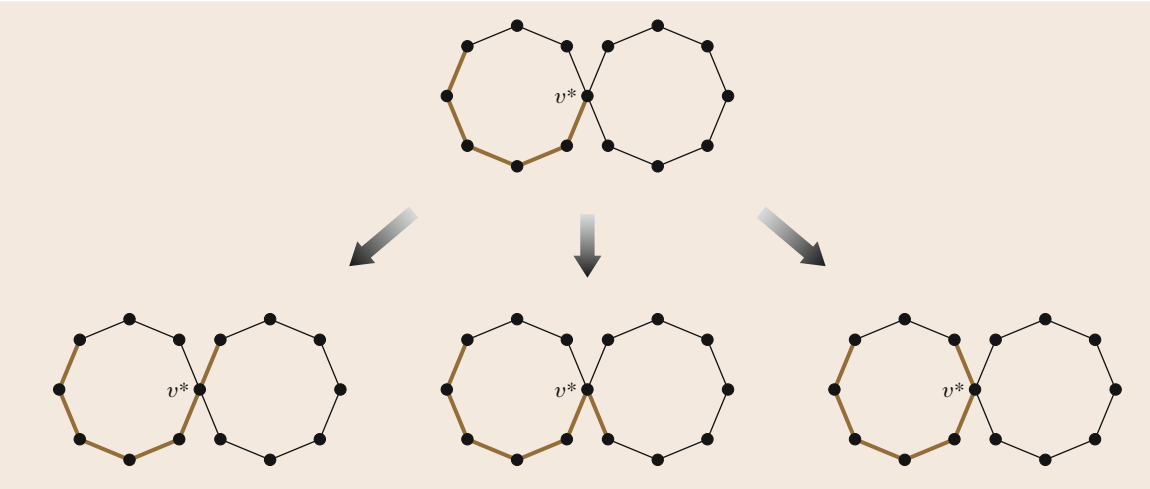


**Fig. 46.9** Sketch of the graph $G'$. The *top* shows a configuration where a decision at $v^*$ has to be made. The three configurations *below* show the possible outcomes. All these transitions occur with equal probability, but only the one *on the bottom right* leads to a solution where rotations are necessary

is only successful because it can use both migration and periods of independent evolution.

The theoretical results [46.3] were complemented by experiments in [46.47]. The aim was to look at what impact the choice of the migration topology and the choice of the migration interval have on performance, regarding the function LOLZ. The theoretical results made a statement about a broad class of dense topologies, but required a very precise migration interval. The experiments showed that the island model is far more robust with respect to the migration interval than suggested by theory.

Depending on the migration interval, some topologies were better than others. The topologies involved were a bidirectional ring, a torus with edges wrapping around, a hypercube graph, and the complete graph. We considered the success rate of the island model, stopping it as soon as all islands had reached local or global optima. We then performed statistical tests comparing these success rates. For small migration intervals, i. e., frequent migrations, sparse topologies were better than dense ones. For large migration intervals, i. e., rare migrations, the effect was the opposite. This effect was expected; however, we also found that the torus was generally better than the hypercube. This is surprising, as both have a similar density. Table 46.2 shows the ranking obtained for commonly used topologies.

Superlinear speedups with island models also occur in simpler settings. *Lässig* and *Sudholt* [46.6] also considered island models for the Eulerian cycle problem. Given an undirected Eulerian graph, the task is to find a Eulerian cycle, i. e., a traversal of the graph on which each edge is traversed exactly once. This problem can be solved efficiently by tailored algorithms, but it served as an excellent test bed for studying the performance of evolutionary algorithms [46.48–51].

Instead of bit strings, the problem representation by *Neumann* [46.48] is based on permutations of the edges of the graph. Each such permutation gives rise to a *walk*: starting with the first edge, a walk is the longest sequence of edges such that two subsequent edges in the permutation share a common vertex. The walk encoded by the permutation ends when the next edge does not share a vertex with the current one. A walk that contains all edges represents a Eulerian cycle. The length of the walk gives the fitness of the current solution.

*Neumann* [46.48] considered a simple instance that consists of two cycles of equal size, connected by one common vertex $v^*$ (Fig. 46.9). The instance is interesting as it represents a worst case for the time until an

**Table 46.2** Performance comparison according to success rates for commonly used migration topologies. The notion $A \prec B$ means that topology $A$ has a significantly smaller success rate than topology $B$

| Migration interval | Ranking |
|---|---|
| Small migration intervals | $K_\mu \prec$ hypercube $\prec$ torus $\prec$ ring |
| Medium migration intervals | hypercube $\prec K_\mu \prec$ ring $\prec$ torus |
| High migration intervals | ring $\prec$ torus $\prec$ hypercube $\prec K_\mu$ |

improvement is found. This is with respect to randomized local search (RLS) working on this representation. RLS works like the $(1+1)$ EA, but it only uses local mutations. As the mutation operator it uses jumps: an edge is selected uniformly at random and then it is moved to a (different) target position chosen uniformly at random. All edges in between the two positions are shifted accordingly.

On the considered instance RLS typically starts constructing a walk within one of these cycles, either by appending edges to the end of the walk or by prepending edges to the start of the walk. When the walk extends to $v^*$ for the first time, a decision needs to be made. RLS can either extend the walk to the opposite cycle, Fig. 46.9. In this case, RLS can simply extend both ends of the walk until a Eulerian cycle is formed. The expected time until this happens is $\Theta(m^3)$, where $m$ denotes the number of edges.

However, if another edge in the same cycle is added at $v^*$, the walk will evolve into one of the two cycles that make up the instance. It is not possible to add further edges to the current walk, unless the current walk starts and ends in $v^*$. However, the walk can be rotated so that the start and end vertex of the walk is moved to a neighbored vertex. Such an operation takes expected time $\Theta(m^2)$. Note that the fitness after a rotation is the same as before. Rotations that take the start and end closer to $v^*$ are as likely as rotations that move it away from $v^*$. The start and end of the walk hence performs a fair random walk, and $\Theta(m^2)$ rotations are needed on average in order to reach $v^*$. The total expected time for rotating the cycle is hence $\Theta(m^4)$.

Summarizing, if RLS makes the right decision then expected time $\Theta(m^3)$ suffices in total. However, if rotations become necessary the expected time increases to $\Theta(m^4)$. Now consider an island model with $m$ islands running RLS. If islands evolve independently for at least $\tau \geq m^3$ generations, all mentioned decisions are made independently, with high probability. The probability of making a wrong decision is $1/3$, hence with $m$ islands the probability that all islands make the wrong decision is $3^{-m}$. The expected time can be shown to be

$$\Theta(m^3 + 3^{-m} \cdot m^4) \,.$$

The choice $m := \log_3 m$ yields an expectation of $\Theta(m^3)$, and every value up to $\log_3 m$ leads to a superlinear speedup, asymptotically speaking. Technically, the speedup is even exponential.

Interestingly, this good performance only holds if migration is used rarely, or if independent runs are used. If migration is used too frequently, the island model rapidly loses diversity. If $T$ is any strongly connected topology and $\mathrm{diam}(T)$ is its diameter, we have the following. If

$$\tau \cdot \mathrm{diam}(T) \cdot m = O(m^2) \,,$$

then there is a constant probability that the island that first arrives at a decision at $v^*$ propagates this solution throughout the whole island model, before any other island can make an improvement. This results in an expected running time of $\Omega(m^4/\log(m))$. This is almost $\Theta(m^4)$, even for very large numbers of islands. The speedup is, therefore, logarithmic at best, or even worse. This natural example shows that the choice of the migration interval can make a difference between exponential and logarithmic speedups.

### 46.4.4 Crossover Between Islands

It has long been known that island models can also be useful in the context of crossover. Crossover usually requires a good diversity in the population to work properly. Due to the higher diversity between different islands, compared to panmixia, recombining individuals from different islands is promising.

*Watson* and *Jansen* [46.52] presented and analyzed a royal road function for crossover: a function where crossover drastically outperforms mutation-based evolutionary algorithms. In contrast to previous theoretically studied examples [46.53–57], their goal was to construct a function with a clear building-block structure. In order to prove that a GA was able to assemble all building blocks, they resorted to an island model with a very particular migration topology. In their *single-receiver model* all islands except one evolve independently. Each island sends its migrants to a designated island called the *receiver* (Fig 46.10). This way, all sending islands are able to evolve the right building blocks, and the receiver is used to assemble all these building blocks to obtain the optimum.

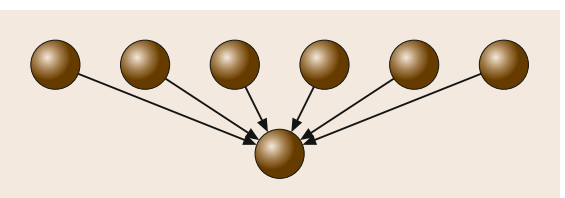

**Fig. 46.10** The topology for *Watson* and *Jansen's* single-receiver model (after [46.52])

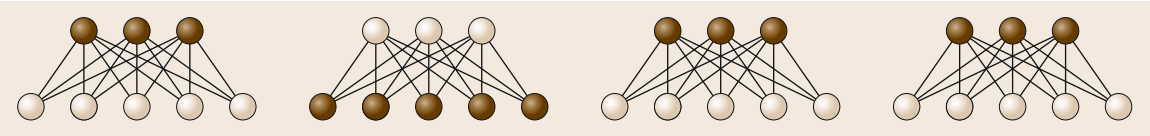

**Fig. 46.11** Vertex cover instance with bipartite graphs. The *brown vertices* denote selected vertices. In this configuration *the second component* shows a locally optimal configuration while all other components are globally optimal

This idea was picked up later on by *Neumann* et al. [46.7] in a more detailed study of crossover in island models. We describe parts of their results, as their problem is more illustrative than the one by Watson and Jansen. The former authors considered instances of the NP-hard *Vertex cover* problem. Given an undirected graph, the goal is to select a subset of vertices such that each vertex is either selected or neighbored to a selected vertex. We say that vertices are *covered* if this property holds for them. The objective is to minimize the number of selected vertices. The problem has a simple and natural binary representation where each bit indicates whether a corresponding vertex is selected or not.

Prior work by *Oliveto* et al. [46.58] showed that evolutionary algorithms with panmictic populations even fail on simply structured instance classes like copies of bipartite graphs. An example is shown in Fig. 46.11. Consider a single bipartite graph, i. e., two sets of vertices such that each vertex in one set is connected to every vertex in the other set. If both sets have different sizes, the smaller set is an optimal Vertex cover. The larger set is another Vertex cover. It is, in fact, a non-optimal local optimum which is hard to overcome: the majority of bits has to flip in order to escape. If the instance consists of several independent copies of bipartite graphs, it is very likely that a panmictic EA will evolve a locally optimal configuration on at least one of the bipartite graphs. Then the algorithm fails to find a global optimum.

Island models perform better. Assume the topology is the single-receiver model. In each migration a 2-point crossover is performed between migrants and the individual on the target island. All islands have population size 1 for simplicity. We also assume that the bipartite subgraphs are encoded in such a way that each subgraph forms one block in the bit string. This is a natural assumption as all subgraphs can be clearly identified as building blocks. In addition, *Jansen* et al. [46.59] presented an automated way of encoding graphs in a crossover-friendly way, based on the degrees of vertices.

The analysis in [46.7] shows the following. Assume that the migration interval is at least $\tau \geq n^{1+\varepsilon}$ for some positive constant $\varepsilon > 0$. This choice implies that all islands will evolve to configurations where all bipartite graphs are either locally optimal or globally optimal. With probability $1 - e^{-\Omega(m)}$ we have that for each bipartite graph at least a constant fraction of all sender islands will have the globally optimal configuration.

All that is left to do for the receiver island is to rely on crossover combining all present good building blocks. As two-point crossover can select one block from an immigrant and the remainder from the current solution on the receiver island, all good building blocks have a good chance to be obtained. The island model finds a global optimum within a polynomial number of generations, with probability $1 - e^{-\Omega(\min\{n^{\varepsilon/2}, m\})}$.

## 46.5 Speedups by Parallelization

### 46.5.1 A General Method for Analyzing Parallel EAs

We now finally discuss a method for estimating the speedup by parallelization. Assume that, instead of running a single EA, we run an island model where each island runs the same EA. The question is by how much the expected optimization time (i. e., the number of generations until a global optimum is found) decreases, compared to the single, panmictic EA. Recall that this speedup is called weak orthodox speedup [46.14].

In the following we sometimes speak of the expected *parallel* optimization time to emphasize that we are dealing with a parallel system. If the number of islands and the population size on each island is fixed, we can simply multiply this time by a fixed factor to obtain the expected number of function evaluations.

*Lässig* and *Sudholt* [46.4] presented a method for estimating the expected optimization time of island models. It combines growth curves with a well-known method for the analysis of evolutionary algorithms. The *fitness-level method* or *method of f-based partitions* [46.60] is a simple, yet powerful technique. The idea is to partition the search space into non-empty sets $A_1, A_2, \ldots, A_m$ such that the following holds:

- for each $1 \leq i < m$ each search point in $A_i$ has a strictly worse fitness than each search point in $A_{i+1}$ and
- $A_m$ contains all global optima.

The described ordering with respect to the fitness $f$ is often denoted

$$A_1 <_f A_2 <_f \cdots <_f A_m \,.$$

Note that $A_m$ can also be redefined towards containing all search points of some desired quality if the goal is not global optimization.

We say that a population-based algorithm $\mathcal{A}$ (including populations of size 1) is in $A_i$ or *on fitness level i* if the best search point in the population is in $A_i$. Now, assume that we know that $s_i$ is a lower bound on the probability that the algorithm finds a solution in $A_{i+1} \cup \cdots \cup A_m$ if it is currently in $A_i$. Then the reciprocal $1/s_i$ is an upper bound on the expected time until this event happens. If the algorithm is *elitist* (i. e., it never loses the current best solution), then it will never decrease its current fitness level. A sufficient condition for finding an optimal solution is that all sets $A_1, A_2, \ldots, A_{m-1}$ are left in the described manner at least once. This implies the following bound on the expected optimization time.

*Theorem 46.1 Wegener [46.60]*
Consider an elitist EA and assume a fitness-level partition $A_1 <_f \cdots <_f A_m$ where $A_m$ is the set of global optima. Let $s_i$ be a lower bound for the probability that in one generation the EA finds a search point in $A_{i+1} \cup \cdots \cup A_m$ if the best individual in the parent population is in $A_i$. Then the expected optimization time is bounded by

$$\sum_{i=1}^{m-1} \frac{1}{s_i} \,.$$

The above bound applies to all elitist algorithms. It is generally applicable and often quite versatile, as we can freely choose the partition $A_1, \ldots, A_m$. The challenge is to find such a partition and to find corresponding probability bounds $s_1, \ldots, s_{m-1}$ for finding improvements. Many papers have shown that this method – applied explicitly or implicitly – yields tight bounds on the expected optimization time of EAs for various problems [46.32, 42, 48]. It can also be used as part of a more general analysis [46.61, 62].

We are being pessimistic in assuming that every fitness level has to be left. In reality, several fitness levels might be skipped. The fitness-level method often yields good bounds if not too many levels are skipped, and if the probability bounds $s_i$ are good estimates for the real probabilities of finding a better fitness-level set. Note that the lower bound $s_i$ must apply regardless of the precise search point(s) in $A_i$ present in the population, hence we need to consider the worst-case probability of escaping from $A_i$.

Nevertheless, the fitness-level method often yields tight bounds. *Sudholt* [46.63] recently developed a lower-bound method based on fitness levels, which in each case shows that the upper bound is tight. Also, *Lehre* [46.64] recently presented an extension of the method to non-elitist algorithms. Asymptotically, the same bound as in Theorem 46.1 applies, if some additional conditions on the selection pressure and the population size are fulfilled. For the sake of simplicity, we focus on elitist algorithms in the following.

If $s_i$ denotes the probability of a single offspring finding an improvement, this probability can be increased by using $\lambda$ offspring in parallel. We have already seen in Sect. 46.1 how $\lambda$ independent trials can increase or amplify a success probability $p$ to $1-(1-p)^\lambda$. The same reasoning applies to the probability $s_i$ for finding an improvement on the current best level. Figure 46.1 has shown how this probability increases with the number of trials. Figure 46.12 shows how the expected time for having a success decreases with the number of offspring. In fact, the curves in Fig. 46.12 are just reciprocals of those in the previous Fig. 46.1.

Figure 46.12 shows that the speedup can be close to linear (in a strict, non-asymptotic sense), especially for low success probabilities. As the probability of increasing the current fitness level $i$ is at least $1-(1-s_i)^\lambda$, we obtain the following.

*Theorem 46.2*
Consider an elitist EA creating $\lambda$ offspring independently in each generation. Assume a fitness-level partition $A_1 <_f \cdots <_f A_m$, where $A_m$ is the set of global optima. Let $s_i$ be a lower bound for the probability that

in one generation a single offspring finds a search point in $A_{i+1} \cup \cdots \cup A_m$ if the best individual in the parent population is in $A_i$. Then the expected optimization time is bounded by

$$\sum_{i=1}^{m-1} \frac{1}{1-(1-s_i)^\lambda} \leq m-1+\frac{1}{\lambda}\sum_{i=1}^{m-1}\frac{1}{s_i}\,.$$

Note that the first bound for $\lambda = 1$ reproduces the previous upper bound from Theorem 46.1. For the second bound we used

$$\frac{1}{1-(1-s_i)^\mu} \leq 1+\frac{1}{\mu}\cdot\frac{1}{s_i}\,, \tag{46.2}$$

where the inequality was proposed by Jon Rowe (personal communication, 2011); it can be proven by a simple induction.

Our estimate of the probability for an improvement increases with the number of islands on the current best fitness level. In a spatially structured EA these growth curves are non-trivial. Especially with a sparse migration topology, information about the current best fitness level is typically propagated quite slowly. The increased exploration slows down exploitation. Still, even sparse topologies lead to drastically improved upper bounds, when compared to the simple bound for a sequential EA from Theorem 46.1. The precise bounds crucially depend on the particular topology.

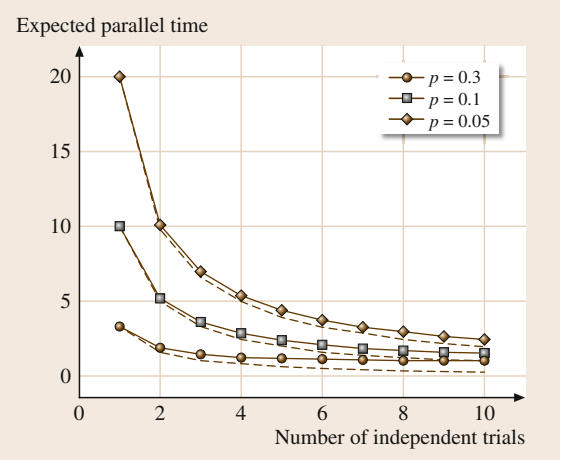


**Fig. 46.12** Plots of the expected parallel time until an offspring population of size $\lambda$ has a success, if each offspring independently has a success probability of $p$. The *dashed lines* indicate a perfect linear speedup

We first consider a setting where migration always transmits the current best fitness level and migration occurs in every generation. It is possible to adapt the results to account for larger migration intervals. One way of doing this is to redefine $s_i$ to represent a lower bound of finding an improvement in a time period between migrations. Then we obtain an upper bound on the expected number of migrations. For the sake of simplicity, we only consider the case $\tau = 1$ in the following.

The following theorem was presented in *Lässig* and *Sudholt* [46.6]; it is a refined special case of previous results [46.4]. The main proof idea is to combine the investigation of growth curves with the consideration of amplified success probabilities.

*Theorem 46.3 Lässig and Sudholt [46.6]*
Consider an island model with $\mu$ islands where each island runs an elitist EA. In every iteration each island sends copies of its best individual to all neighbored islands (i. e., $\tau = 1$). Each island incorporates the best out of its own individuals and its immigrants.

For every partition $A_1 <_f \cdots <_f A_m$ if $s_i$ is a lower bound for the probability that in one generation an island in $A_i$ finds a search point in $A_{i+1} \cup \cdots \cup A_m$ then the expected parallel optimization time is bounded by:

1. $2\sum_{i=1}^{m-1}\frac{1}{s_i^{1/2}}+\frac{1}{\mu}\sum_{i=1}^{m-1}\frac{1}{s_i}$ for every unidirectional ring (a ring with edges in one direction) or any other strongly connected topology,
2. $3\sum_{i=1}^{m-1}\frac{1}{s_i^{1/3}}+\frac{1}{\mu}\sum_{i=1}^{m-1}\frac{1}{s_i}$ for every undirected grid or torus graph with side lengths at least $\sqrt{\mu}\times\sqrt{\mu}$,
3. $m-1+\frac{1}{\mu}\sum_{i=1}^{m-1}\frac{1}{s_i}$ for the complete topology $K_\mu$.

Note that the bound for the complete topology $K_\mu$ is equal to the upper bound for offspring populations, Theorem 46.2. This makes sense as an island model with a complete topology propagates the current best fitness level like an offspring population.

All bounds in Theorem 46.3 consist of two additive terms. The second term

$$\frac{1}{\mu}\sum_{i=1}^{m-1}\frac{1}{s_i}$$

represents a perfect linear speedup, compared to the upper bound from Theorem 46.1. The larger we choose the number of islands $\mu$, the smaller this term becomes. The first additive term is related to the growth curves of the current best fitness level in the island model. The

denser the topology, the faster information is spread, and the smaller this term becomes. Note that it is independent of $\mu$. It can be regarded as the term limiting the degree of parallelizability. We can increase the number of islands in order to decrease the second term

$$\frac{1}{\mu}\sum_{i=1}^{m-1}\frac{1}{s_i}\,,$$

but we cannot decrease the first term by changing $\mu$.

This allows for immediate conclusions about cases where we obtain an asymptotic linear speedup over a single-island EA. For all choices of $\mu$ where the second term is asymptotically no smaller than the first term, the upper bound is smaller than the upper bound from Theorem 46.1 by a factor of order $\mu$. This is an asymptotic linear speedup if the upper bound from Theorem 46.1 is asymptotically tight. (If it is not, we can only compare upper bounds for a sequential and a parallel EA.)

We illustrate this with a simple and well-known test function from pseudo-Boolean optimization. The algorithm considered is an island model where each island runs a $(1+1)$ EA; the island model is also called parallel $(1+1)$ EA. The function

$$\mathrm{LO}(x) := \sum_{i=1}^{n}\prod_{j=1}^{i} x_j \quad (\textit{LeadingOnes})$$

counts the number of leading ones in the bit string. We choose the canonic partition where $A_i$ contains all search points with fitness $i$, i. e., $i$ leading ones. For any set $A_i$, $0 \le i \le n-1$ we use the following lower bound on the probability for an improvement.

An improvement occurs if the first 0-bit is flipped from 0 to 1 and no other bit flips. The probability of flipping the mentioned 0-bit is $1/n$ as each bit is flipped independently with probability $1/n$. The probability of not flipping any other bit is $(1-1/n)^{n-1}$. We use the common estimate $(1-1/n)^{n-1} \ge 1/e$, where $e = \exp(1) = 2.718\ldots$, so the probability of an improvement is at least $s_i \ge 1/(en)$ for all $0 \le i \le n-1$. Plugging this into Theorem 46.3, the second term is $\frac{1}{\mu}\cdot en^2$ for all bounds. The first terms are

$$2n\cdot(en)^{1/2} = 2e^{1/2}n^{3/2}$$

for the ring,

$$3n\cdot(en)^{1/3} = 3e^{1/3}n^{4/3}$$

for the torus, and $n$ for the complete graph, respectively.

For the ring, choosing $\mu = O(n^{1/2})$ islands results in an expected parallel time of $O(\frac{1}{\mu}\cdot n^2)$ as the second term is asymptotically not smaller than the first one. This is asymptotically smaller by a factor of $1/\mu$ than the expected optimization time of a single $(1+1)$ EA, $\Theta(n^2)$ [46.42]. Hence, each choice of $\mu$ up to $\mu = O(n^{1/2})$ gives a linear speedup. For the torus we obtain a linear speedup for $\mu = O(n^{2/3})$ in the same fashion. For the complete graph this even holds for $\mu = O(n)$. One can see here that the island model can decrease the expected parallel running time by significant polynomial factors.

Table 46.3 lists expected parallel optimization time bounds for several well-known pseudo-Boolean functions. The above analysis for LO generalizes to all unimodal functions. A function is called unimodal here if every non-optimal search point has a better Hamming neighbor, i. e., a better search point can be reached by flipping exactly one specific bit. $\mathrm{ONEMAX}(x) = \sum_{i=1}^{n} x_i$ counts the number of ones, hence modeling a simple hill climbing task. Finally, $\mathrm{Jump}_k$ [46.42] is a multimodal function of tunable difficulty. An EA

**Table 46.3** Upper bounds for expected parallel optimization times (number of generations) for the $(1+1)$ EA and the corresponding island model with $\mu$ islands in pseudo-Boolean optimization. The last but one column is for any unimodal function with $d$ function values. The number of function evaluations in the island model is larger than the number of generations by a factor of $\mu$

| Algorithm | ONEMAX | LO | Unimodal, $d$ values | $\mathrm{Jump}_k$, $k \ge 3$ |
|---|---|---|---|---|
| $(1+1)$ EA | $O(n\log n)$ [46.42] | $O(n^2)$ [46.42] | $O(nd)$ | $O(n^k)$ [46.42] |
| Island model on ring | $O\left(n+\frac{n\log n}{\mu}\right)$ | $O\left(n^{3/2}+\frac{n^2}{\mu}\right)$ | $O\left(dn^{1/2}+\frac{dn}{\mu}\right)$ | $O\left(n^{k/2}+\frac{n^k}{\mu}\right)$ |
| Island model on torus | $O\left(n+\frac{n\log n}{\mu}\right)$ | $O\left(n^{4/3}+\frac{n^2}{\mu}\right)$ | $O\left(dn^{1/3}+\frac{dn}{\mu}\right)$ | $O\left(n^{k/3}+\frac{n^k}{\mu}\right)$ |
| Island model on $K_\mu/(1+\mu)$ EA | $O\left(n+\frac{n\log n}{\mu}\right)$ | $O\left(n+\frac{n^2}{\mu}\right)$ | $O\left(d+\frac{dn}{\mu}\right)$ | $O\left(n+\frac{n^k}{\mu}\right)$ |

typically has to make a *jump* by flipping $k$ bits simultaneously, where $2 \le k \le n$. The $(1+1)$ EA has an expected optimization time of $\Theta(n^k)$, hence growing rapidly with increasing $k$.

One can see that the island model leads to drastically reduced parallel optimization times. This particularly holds for problems where improvements are hard to find.

We remark that *Lässig* and *Sudholt* [46.4] also considered parallel EAs where migration is not always successful in transmitting information about the current best fitness level. This includes the case where crossover is used during migration and crossover has a certain probability of being disruptive. We do obtain upper bounds on the expected optimization time if we know a lower bound $p^+$ on the probability of a successful transmission. The bounds depend on $p^+$; the degree of this dependence is determined by the topology. For simplicity we only focus on the deterministic case here.

### 46.5.2 Speedups in Combinatorial Optimization

The techniques are also applicable in combinatorial optimization. We review two examples here, presented in [46.6]. *Scharnow* et al. [46.32] considered the classical sorting problem as an optimization problem: given a sequence of $n$ distinct elements from a totally ordered set, sorting is the problem of maximizing sortedness. Without loss of generality the elements are $1,\dots,n$; then the aim is to find the permutation $\pi_{\text{opt}}$ such that $(\pi_{\text{opt}}(1),\dots,\pi_{\text{opt}}(n))$ is the sorted sequence.

The search space is the set of all permutations $\pi$ on $1,\dots,n$. Two different operators are used for mutation. An exchange chooses two indices $i \neq j$ uniformly at random from $\{1,\dots,n\}$ and exchanges the entries at positions $i$ and $j$. A jump chooses two indices in the same fashion. The entry at $i$ is put at position $j$ and all entries in between are shifted accordingly. For instance, a jump with $i=2$ and $j=5$ would turn $(1,2,3,4,5,6)$ into $(1,3,4,5,2,6)$.

The $(1+1)$ EA draws $S$ according to a Poisson distribution with parameter $\lambda = 1$ and then performs $S+1$ elementary operations. Each operation is either an exchange or a jump, where the decision is made independently and uniformly for each elementary operation. The resulting offspring replaces its parent if its fitness is not worse. The fitness function $f_{\pi_{\text{opt}}}(\pi)$ describes the sortedness of $(\pi(1),\dots,\pi(n))$. As in [46.32], we consider the following measures of sortedness:

- $\text{INV}(\pi)$ measures the number of pairs $(i,j)$, $1 \le i < j \le n$, such that $\pi(i) < \pi(j)$ (pairs in correct order),
- $\text{HAM}(\pi)$ measures the number of indices $i$ such that $\pi(i) = i$ (elements at the correct position),
- $\text{LAS}(\pi)$ equals the largest $k$ such that $\pi(i_1) < \dots < \pi(i_k)$ for some $i_i < \dots < i_k$ (length of the longest ascending subsequence),
- $\text{EXC}(\pi)$ equals the minimal number of exchanges (of pairs $\pi(i)$ and $\pi(j)$) to sort the sequence, leading to a minimization problem.

The expected optimization time of the $(1+1)$ EA is $\Omega(n^2)$ and $O(n^2 \log n)$ for all fitness functions. The upper bound is tight for LAS, and it is believed to be tight for INV, HAM, and EXC as well [46.32]. Theorem 46.3 yields the following. For INV, all topologies guarantee a linear speedup only in case $\mu = O(\log n)$ and the bound $O(n^2 \log n)$ for the $(1+1)$ EA is tight. The other functions allow for linear speedups up to $\mu = O(n^{1/2} \log n)$ (ring), $\mu = O(n^{2/3} \log n)$ (torus), and $\mu = O(n \log n)$ ($K_\mu$), respectively (again assuming tightness, otherwise up to a factor of $\log n$). Note how the results improve with the density of the topology. HAM, LAS, and EXC yield much better guarantees for the island model than INV. This is surprising as there is no visible performance difference for a single $(1+1)$ EA. Theorem 46.3 yields the following results also shown in Tab. 46.4

**Table 46.4** Upper bounds for expected parallel optimization times for the $(1+1)$ EA and the corresponding island model with $\mu$ islands for sorting $n$ objects

| Algorithm | INV | HAM, LAS, EXC |
|---|---|---|
| $(1+1)$ EA | $O(n^2 \log n)$ [46.32] | $O(n^2 \log n)$ [46.32] |
| Island model on ring | $O\left(n^2 + \frac{n^2 \log n}{\mu}\right)$ | $O\left(n^{3/2} + \frac{n^2 \log n}{\mu}\right)$ |
| Island model on torus | $O\left(n^2 + \frac{n^2 \log n}{\mu}\right)$ | $O\left(n^{4/3} + \frac{n^2 \log n}{\mu}\right)$ |
| Island model on $K_\mu/(1+\mu)$ EA | $O\left(n^2 + \frac{n^2 \log n}{\mu}\right)$ | $O\left(n + \frac{n^2 \log n}{\mu}\right)$ |

**Table 46.5** Worst-case expected parallel optimization times for the $(1+1)$ EA and the corresponding island model with $\mu$ islands for the SSSP on graphs with $n$ vertices and $m$ edges. The value $\ell$ is the maximum number of edges on any shortest path from the source to any vertex and $\ell^* := \max\{\ell, \ln n\}$. The second lines show a range of $\mu$-values yielding a linear speedup, apart from a factor $\ln(en/\ell)$

| Algorithm | Vertex-based mutation [46.32] | Edge-based mutation [46.65] |
|---|---|---|
| $(1+1)$ EA | $\Theta(n^2\ell^*)$ [46.34] | $\Theta(m\ell^*)$ [46.65] |
| Island model on ring | $O\left(n^{3/2}\ell^{1/2} + \frac{n^2\ell\ln(en/\ell)}{\mu}\right)$ $\longrightarrow \mu = O\left((n\ell)^{1/2}\right)$ | $O\left(m^{1/2}n^{1/2}\ell^{1/2} + \frac{m\ell\ln(en/\ell)}{\mu}\right)$ $\longrightarrow \mu = O\left((m/n\cdot\ell)^{1/2}\right)$ |
| Island model on torus | $O\left(n^{4/3}\ell^{1/3} + \frac{n^2\ell\ln(en/\ell)}{\mu}\right)$ $\longrightarrow \mu = O\left((n\ell)^{2/3}\right)$ | $O\left(m^{1/3}n^{2/3}\ell^{1/3} + \frac{m\ell\ln(en/\ell)}{\mu}\right)$ $\longrightarrow \mu = O\left((m/n\cdot\ell)^{2/3}\right)$ |
| Island model on $K_\mu/(1+\mu)$ EA | $O\left(n + \frac{n^2\ell\ln(en/\ell)}{\mu}\right)$ $\longrightarrow \mu = O(n\ell)$ | $O\left(n + \frac{m\ell\ln(en/\ell)}{\mu}\right)$ $\longrightarrow \mu = O(m/n\cdot\ell)$ |

An explanation is that INV leads to $\binom{n}{2}$ non-optimal fitness levels that are quite easy to overcome. HAM, LAS, and EXC have only $n$ non-optimal fitness levels that are more difficult. For a single EA both settings are equally difficult, leading to asymptotically equal expected times (assuming all upper bounds are tight). However, the latter setting is easier to parallelize than the former as it is easier to amplify small success probabilities.

We also consider parallel variants of the $(1+1)$ EA for the single source shortest path problem (SSSP) [46.32]. An SSSP instance is given by an undirected connected graph with vertices $\{1,\dots,n\}$ and a distance matrix $D = (d_{ij})_{1\le i,j\le n}$, where $d_{ij} \in \mathbb{R}_0^+ \cup \{\infty\}$ defines the length value for given edges from node $i$ to node $j$. We are searching for shortest paths from a node $s$ (without loss of generality $s = n$) to each other node $1 \le i \le n-1$.

A candidate solution is represented as a *shortest paths tree*, a tree rooted at $s$ with directed shortest paths to all other vertices. We define a search point $x$ as vector of length $n-1$, where position $i$ describes the predecessor node $x_i$ of node $i$ in the shortest path tree. Note that infeasible solutions are possible if the predecessors do not encode a tree. An elementary mutation chooses a vertex $i$ uniformly at random and replaces its predecessor $x_i$ by a vertex chosen uniformly at random from $\{1,\dots,n\}\setminus\{i, x_i\}$. We call this a vertex-based mutation. *Doerr* et al. [46.65] proposed an edge-based mutation operator. An edge is chosen uniformly at random, and the edge is made a predecessor edge for its end node.

The $(1+1)$ EA uses either vertex-based mutations or edge-based ones. It creates an offspring using $S$ elementary mutations, where $S$ is chosen according to a Poisson distribution with $\lambda = 1$. The result of an offspring is accepted in case no distance to any vertex has gotten worse.

Applying Theorem 46.3 along with a layering argument as described at the end of Sect. 46.3.4 yields the bounds on the expected parallel optimization time shown in Table 46.5.

The upper bounds for the island models with constant $\mu$ match the expected time of the $(1+1)$ EA if $\ell = O(1)$ or $\ell = \Omega(n)$ as then $\ell\ln(en/\ell) = \Theta(\ell^*)$. In other cases, the upper bounds are off by a factor of $\ln(en/\ell)$. Table 46.5 also shows a range of $\mu$-values for which the speedup is linear (if $\ell = O(1)$ or $\ell = \Omega(n)$) or almost linear, that is, when disregarding the $\ln(en/\ell)$ term.

Note how the possible speedups significantly increase with the density of the topology. The speedups also depend on the graph instance and the maximum number of edges $\ell$ on any shortest path. For a single $(1+1)$ EA edge-based mutations are more effective than vertex-based mutations [46.65]. Island models with edge-based mutations cannot be paral-

**Table 46.6** Asymptotic bounds for expected parallel running times and expected sequential running times for the parallel $(1+1)$ EA with adaptive population models

| | Scheme | Sequential | Parallel |
|---|---|---|---|
| ONEMAX | A | $\Theta(n\log n)$ | $O(n\log n)$ |
| | B | $\Theta(n\log n)$ | $O(n)$ |
| LO | A | $\Theta(n^2)$ | $\Theta(n\log n)$ |
| | B | $\Theta(n^2)$ | $O(n)$ |
| Unimodal $f$ | A | $O(dn)$ | $O(d\log n)$ |
| with $d$ $f$-values | B | $O(dn)$ | $O(d+\log n)$ |
| $\text{Jump}_k$ | A | $O(n^k)$ | $O(n\log n)$ |
| with $k \ge 2$ | B | $O(n^k)$ | $O(n + k\log n)$ |

lelized as effectively for sparse graphs as those with vertex-based mutations if the graph is sparse, i. e., $m = o(n^2)$. Then the number of islands that guarantees a linear speedup is smaller for edge-based mutations than for vertex-based mutations. The reason is that with a more efficient mutation operator there is less potential for further speedups with a parallel EA.

### 46.5.3 Adaptive Numbers of Islands

Theorem 46.3 presents a powerful tool for determining the number of islands that give an asymptotic linear speedup. However, it would be even more desirable to have an adaptive system that automatically finds the ideal number of islands throughout the run.

In [46.5] *Lässig* and *Sudholt* proposed and analyzed two simple adaptive schemes for choosing the number of islands. Both schemes check whether in the current generation some island has found an improvement over the current best fitness in the system. If no island has found an improvement, the number of islands is doubled. This can be implemented, for instance, by copying each island. New processors can be allocated to host these islands in large clusters or by using cloud computing.

If some island has found an improvement, the number of islands is reduced by removing selected islands from the system and de-allocating resources. Both schemes differ in the way they decrease the number of islands. The first scheme, simply called Scheme A, only keeps one island containing a current best solution. Scheme B halves the number of islands. Both schemes use complete topologies, so all remaining islands will contain current best individuals afterwards.

Both mechanisms lead to optimal speedups in many cases. Doubling the number of islands may seem aggressive, but the analysis shows that the probability of allocating far more islands than necessary is very very small. The authors considered the expected sequential optimization time, defined as the number of function evaluations, to measure the total effort over time. With both schemes it is guaranteed that the expected sequential time does not exceed the simple bound for a sequential EA from Theorem 46.1, asymptotically. The expected parallel times on each fitness level can, roughly speaking, be replaced by their logarithms.

The following is a slight simplification of results in [46.5].

*Theorem 46.4 Lässig and Sudholt [46.5]*
Given an $f$-based partition $A_1, \ldots, A_m$ and lower bounds $s_1, \ldots, s_{m-1}$ on the probability of a single island finding an improvement, the expected sequential times for island models using a complete topology and either Scheme A or Scheme B are bounded by

$$3 \sum_{i=1}^{m-1} \frac{1}{s_i} .$$

If each set $A_i$ contains only a single fitness value then also the expected parallel time is bounded by

$$4 \sum_{i=1}^{m-1} \log\left(\frac{2}{s_i}\right) .$$

Actually, for Scheme A we can obtain slightly better constants than the ones stated in Theorem 46.4. However, with a more detailed analysis one can show that Scheme B can perform much better than Scheme A. *Lässig*'s and *Sundholt*'s work [46.5] contains a more refined upper bound for Scheme B. We only show a special case where the fitness levels become increasingly harder. Then it makes sense to only halve the number of islands when an improvement is found, instead of resetting the number of islands to 1.

*Theorem 46.5 Lässig and Sudholt [46.5]*
Given an $f$-based partition $A_1, \ldots, A_m$, where each set $A_i$ contains only a single fitness value and for the probability bounds it holds $s_1 \geq s_2 \geq \cdots \geq s_{m-1}$. Then the expected parallel running time for an island model using a complete topology and Scheme B is bounded by

$$3(m-2) + \log\left(\frac{1}{s_{m-1}}\right) .$$

Example applications for a parallel $(1+1)$ EA in Table 46.6 show that Scheme B can automatically lead to the same speedups as when using an optimal number of islands. This holds for ONEMAX, LO, and the general bound for unimodal functions. For $\text{Jump}_k$ it also holds in the most relevant cases, when $k = O(n/\log n)$, as then the expected parallel time is $O(n)$.

We conclude that simply doubling or halving the number of islands represents a simple and effective mechanism for finding optimal parameters adaptively.

## 46.6 Conclusions

Parallel evolutionary algorithm can effectively reduce computation time and at the same time lead to an increased exploration and better diversity, compared to sequential evolutionary algorithms.

We have surveyed various forms of parallel EAs, from independent runs to island models and cellular EAs. Different lines of research have been discussed that give insight into the working principles behind parallel EAs. This includes the spread of information, growth curves for current best solutions, and takeover times.

A recurring theme was the possible speedup that can be achieved with parallel EAs. We have elaborated on the reasons why superlinear speedups are possible in practice. Rigorous runtime analysis has given examples where parallel EAs excel over sequential algorithms, with regard to the number of generations or the number of function evaluations until a global optimum is found. The final section has covered a method for estimating the expected parallel optimization time of island models. The method is easy to apply as we can automatically transfer existing analyses for sequential EAs to a parallel version thereof. Examples have been given for pseudo-Boolean optimization and combinatorial optimization. The results have also led to the discovery of a simple, yet surprisingly powerful adaptive scheme for choosing the number of islands.

There are many possible avenues for future work. In the light of the development in computer architecture, it is important to develop parallel EAs that can run effectively on many cores. It also remains a crucial issue to increase our understanding of how design choices and parameters affect the performance of parallel EAs. Rigorous runtime analysis has emerged recently as a new line of research that can give novel insights in this respect and opens new roads. The present results should be extended towards further algorithms, further problems, and more detailed cost models that reflect the costs for communication in parallel architectures. It would also be interesting to derive further rigorous results on takeover times in settings where propagation through migration is probabilistic. Finally, it is important to bring theory and practice together in order to create synergetic effects between the two areas.

### 46.6.1 Further Reading

This book chapter does not claim to be comprehensive. In fact, parallel evolutionary algorithms represent a vast research area with a long history. Early variants of parallel evolutionary algorithms were developed, studied, and applied more than 20 years ago. We, therefore, point the reader to references that may complement this chapter. *Paz* [46.66] presented a review of early literature and the history of parallel EAs. The survey by *Alba* and *Troya* [46.37] contains detailed overviews of parallel EAs and their characteristics.

This chapter does not cover implementation details of parallel evolutionary algorithms. We refer to the excellent survey by *Alba* and *Tomassini* [46.38]. This survey also includes an overview of the theory of parallel EAs. The emphasis is different from this chapter and it can be used to complement this chapter.

*Tomassini's* text book [46.67] describes various forms of parallel EAs like island models, cellular EAs, and coevolution. It also presents many mathematical and experimental results that help understand how parallel EAs work. Furthermore, it contains an appendix dealing with the implementation of parallel EAs.

The book edited by *Alba* et al. [46.39] takes a broader scope on parallel models that also include parallel evolutionary multiobjective optimization and parallel variants of swarm intelligence algorithms like particle swarm optimization and ant colony optimization. The book contains a part on parallel hardware as well as a number of applications of parallel metaheuristics.

*Alba*'s edited book on parallel metaheuristics [46.40] has an even broader scope. It covers parallel variants of many common metaheuristics such as genetic algorithms, genetic programming, evolution strategies, ant colony optimization, estimation-of-distribution algorithms, scatter search, variable-neighborhood search, simulated annealing, tabu search, greedy randomized adaptive search procedures (GRASPs), hybrid metaheuristics, multiobjective optimization, and heterogeneous metaheuristics.

The most recent text book was written by *Luque* and *Alba* [46.19]. It provides an excellent introduction into the field, with hands-on advice on how to present results for parallel EAs. Theoretical models of selection pressure in distributed GAs are presented. A large part of the book then reviews selected applications of parallel GAs.

## References


46.1 P.S. Oliveto, J. He, X. Yao: Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results, Int. J. Autom. Comput. **4**(3), 281–293 (2007)

46.2 F. Neumann, C. Witt: *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity* (Springer, Berlin, Heidelberg 2010)

46.3 J. Lässig, D. Sudholt: The benefit of migration in parallel evolutionary algorithms, Proc. Genet. Evol. Comput. Conf. (GECCO 2010) (ACM, New York 2010) pp. 1105–1112

46.4 J. Lässig, D. Sudholt: General scheme for analyzing running times of parallel evolutionary algorithms, 11th Int. Conf. Parallel Probl. Solving Nat. (PPSN 2010) (Springer, Berlin, Heidelberg 2010) pp. 234–243

46.5 J. Lässig, D. Sudholt: Adaptive population models for offspring populations and parallel evolutionary algorithms, Proc. 11th Workshop Found. Genet. Algorithms (FOGA 2011) (ACM, Berlin, Heidelberg 2011) pp. 181–192

46.6 J. Lässig, D. Sudholt: Analysis of speedups in parallel evolutionary algorithms for combinatorial optimization, 22nd Int. Symp. Algorithms Comput. (ISAAC '11) (Springer, Berlin, Heidelberg 2011) pp. 405–414

46.7 F. Neumann, P.S. Oliveto, G. Rudolph, D. Sudholt: On the effectiveness of crossover for migration in parallel evolutionary algorithms, Proc. Genet. Evol. Comput. Conf. (GECCO 2011) (ACM, New York 2011) pp. 1587–1594

46.8 M. De Felice, S. Meloni, S. Panzieri: Effect of topology on diversity of spatially-structured evolutionary algorithms, Proc. 13th Annu. Genet. Evol. Comput. Conf. (GECCO '11) (2011) pp. 1579–1586

46.9 M. Giacobini, M. Tomassini, A. Tettamanzi: Takeover time curves in random and small-world structured populations, Proc. Genet. Evol. Comput. Conf. (GECCO '05) (ACM, New York 2005) pp. 1333–1340

46.10 Z. Skolicki: An Analysis of Island Models in Evolutionary Computation, Ph.D. Thesis (George Mason University, Fairfax 2000)

46.11 E. Alba, M. Giacobini, M. Tomassini, S. Romero: *Comparing Synchronous and Asynchronous Cellular Genetic Algorithms, Parallel Problem Solving from Nature VII* (Springer, Berlin, Heidelberg 2002) pp. 601–610

46.12 M. Mitzenmacher, E. Upfal: *Probability and Computing* (Cambridge Univ. Press, Cambridge 2005)

46.13 J. Sprave: A unified model of non-panmictic population structures in evolutionary algorithms, Proc. 1999 Congr. Evol. Comput. (IEEE, Bellingham 1999) pp. 1384–1391

46.14 E. Alba: Parallel evolutionary algorithms can achieve super-linear performance, Inf. Process. Lett. **82**(1), 7–13 (2002)

46.15 R.S. Barr, B.L. Hickman: Reporting computational experiments with parallel algorithms: Issues, measures, and experts' opinion, ORSA J. Comput. **5**(1), 2–18 (1993)

46.16 D.E. Goldberg, K. Deb: A comparatative analysis of selection schemes used in genetic algorithms. In: *Foundations of Genetic Algorithms*, ed. by G.J.E. Rawlins (Morgan Kaufmann, Burlington 1991) pp. 69–93

46.17 J. Sarma, K. De Jong: An analysis of local selection algorithms in a spatially structured evolutionary algorithm, Proc. 7th Int. Conf. Genet. Algorithms (Morgan Kaufmann, Burlington 1997) pp. 181–186

46.18 E. Alba, G. Luque: Growth curves and takeover time in distributed evolutionary algorithms, Proc. Genet. Evol. Comput. Conf. (Springer, Berlin, Heidelberg 2004) pp. 864–876

46.19 G. Luque, E. Alba: *Parallel Genetic Algorithms – Theory and Real World Applications*, Studies in Computational Intelligence, Vol. 367 (Springer, Berlin, Heidelberg 2011)

46.20 Z. Skolicki, K.A. De Jong: The influence of migration sizes and intervals on island models, Proc. Genet. Evol. Comput. Conf. (GECCO '05) (ACM, New York 2005) pp. 1295–1302

46.21 M. Giacobini, E. Alba, M. Tomassini: Selection intensity in asynchronous cellular evolutionary algorithms, Proc. Genet. Evol. Comput. Conf. (GECCO '03) (Springer, Berlin, Heidelberg 2003) pp. 955–966

46.22 G. Rudolph: Takeover times and probabilities of non-generational selection rules, Proc. Genet. Evol. Comput. Conf. (GECCO '00) (Morgan Kaufmann, Burlington 2000) pp. 903–910

46.23 G. Rudolph: Takeover times of noisy non-generational selection rules that undo extinction, Proc. 5th Int. Conf. Artif. Neural Nets Genet. Algorithms (ICANNGA 2001) (Springer, Berlin, Heidelberg 2001) pp. 268–271

46.24 G. Rudolph: On takeover times in spatially structured populations: Array and ring, Proc. 2nd Asia-Pac. Conf. Genet. Algorithms Appl. (Global-Link Publishing, Hong Kong 2000) pp. 144–151

46.25 G. Rudolph: Takeover time in parallel populations with migration, Proc. 2nd Int. Conf. Bioinspired Optim. Methods Appl. (BIOMA 2006), ed. by B. Filipic, J. Silc (2006) pp. 63–72

46.26 M. Giacobini, M. Tomassini, A. Tettamanzi: Modelling selection intensity for linear cellular evolutionary algorithms, Proc. 6th Int. Conf. Artif. Evol., Evol. Artif. (Springer, Berlin, Heidelberg 2003) pp. 345–356

46.27 M. Giacobini, E. Alba, A. Tettamanzi, M. Tomassini: Selection intensity in cellular evolutionary algo-

rithms for regular lattices, IEEE Trans. Evol. Comput. **9**, 489–505 (2005)
46.28 C. Witt: Runtime analysis of the $(\mu+1)$EA on simple pseudo-Boolean functions, Evol. Comput. **14**(1), 65–86 (2006)
46.29 D. Sudholt: The impact of parametrization in memetic evolutionary algorithms, Theor. Comput. Sci. **410**(26), 2511–2528 (2009)
46.30 M. Giacobini, E. Alba, A. Tettamanzi, M. Tomassini: Modeling selection intensity for toroidal cellular evolutionary algorithms, Proc. Genet. Evol. Comput. Conf. (GECCO '04) (Springer, Berlin, Heidelberg 2004) pp. 1138–1149
46.31 J. Rowe, B. Mitavskiy, C. Cannings: Propagation time in stochastic communication networks, 2nd IEEE Int. Conf. Digit. Ecosyst. Technol. (2008) pp. 426–431
46.32 J. Scharnow, K. Tinnefeld, I. Wegener: The analysis of evolutionary algorithms on sorting and shortest paths problems, J. Math. Model, Algorithms **3**(4), 349–366 (2004)
46.33 B. Doerr, E. Happ, C. Klein: Crossover can provably be useful in evolutionary computation, Theor. Comput. Sci. **425**, 17–33 (2012)
46.34 B. Doerr, E. Happ, C. Klein: A tight analysis of the $(1+1)$-EA for the single source shortest path problem, Proc. IEEE Congr. Evol. Comput. (CEC '07) (IEEE, Bellingham 2007) pp. 1890–1895
46.35 C. Horoba, D. Sudholt: Ant colony optimization for stochastic shortest path problems, Proc. Genet. Evol. Comput. Conf. (GECCO 2010) (ACM, New York 2010) pp. 1465–1472
46.36 D. Sudholt, C. Thyssen: Running time analysis of ant colony optimization for shortest path problems, J. Discret. Algorithms **10**, 165–180 (2012)
46.37 E. Alba, J.M. Troya: A survey of parallel distributed genetic algorithms, Complexity **4**, 31–52 (1999)
46.38 E. Alba, M. Tomassini: Parallelism and evolutionary algorithms, IEEE Trans. Evol. Comput. **6**, 443–462 (2002)
46.39 E. Alba, N. Nedjah, L. de Macedo Mourelle: *Parallel Evolutionary Computations* (Springer, Berlin, Heidelberg 2006)
46.40 E. Alba: *Parallel Metaheuristics: A New Class of Algorithms* (Wiley-Interscience, New York 2005)
46.41 T.G. Crainic, N. Hail: Parallel metaheuristics applications. In: *Parallel Metaheuristics: A New Class of Algorithms*, (Wiley-Interscience, New York 2005)
46.42 S. Droste, T. Jansen, I. Wegener: On the analysis of the $(1+1)$ evolutionary algorithm, Theor. Comput. Sci. **276**, 51–81 (2002)
46.43 T. Friedrich, P.S. Oliveto, D. Sudholt, C. Witt: Analysis of diversity-preserving mechanisms for global exploration, Evol. Comput. **17**(4), 455–476 (2009)
46.44 C. Witt: Worst-case and average-case approximations by simple randomized search heuristics, Proc. 22nd Symp. Theor. Asp. Comput. Sci. (STACS '05) (Springer, Berlin, Heidelberg 2005) pp. 44–56
46.45 T. Jansen, K.A. De Jong, I. Wegener: On the choice of the offspring population size in evolutionary algorithms, Evol. Comput. **13**, 413–440 (2005)
46.46 C. Igel, M. Toussaint: A no-free-lunch theorem for non-uniform distributions of target functions, J. Math. Model, Algorithms **3**(4), 313–322 (2004)
46.47 J. Lässig, D. Sudholt: Experimental supplements to the theoretical analysis of migration in the island model, 11th Int. Conf. Parallel Probl. Solving Nat. (PPSN 2010) (Springer, Berlin, Heidelberg 2010) pp. 224–233
46.48 F. Neumann: Expected runtimes of evolutionary algorithms for the Eulerian cycle problem, Comput. Oper. Res. **35**(9), 2750–2759 (2008)
46.49 B. Doerr, N. Hebbinghaus, F. Neumann: Speeding up evolutionary algorithms through asymmetric mutation operators, Evol. Comput. **15**, 401–410 (2007)
46.50 B. Doerr, D. Johannsen: Adjacency list matchings – An ideal genotype for cycle covers, Proc. Genet. Evol. Comput. Conf. (GECCO '07) (ACM, New York 2007) pp. 1203–1210
46.51 B. Doerr, C. Klein, T. Storch: Faster evolutionary algorithms by superior graph representation, 1st IEEE Symp. Found. Comput. Intell. (FOCI '07) (2007) pp. 245–250
46.52 R.A. Watson, T. Jansen: A building-block royal road where crossover is provably essential, Proc. Genet. Evol. Comput. Conf. (GECCO '07) (ACM, New York 2007) pp. 1452–1459
46.53 T. Jansen, I. Wegener: On the analysis of evolutionary algorithms – A proof that crossover really can help, Algorithmica **34**(1), 47–66 (2002)
46.54 T. Jansen, I. Wegener: Real royal road functions – Where crossover provably is essential, Discret. Appl. Math. **149**, 111–125 (2005)
46.55 T. Storch, I. Wegener: Real royal road functions for constant population size, Theor. Comput. Sci. **320**, 123–134 (2004)
46.56 S. Fischer, I. Wegener: The one-dimensional ising model: Mutation versus recombination, Theor. Comput. Sci. **344**(2/3), 208–225 (2005)
46.57 D. Sudholt: Crossover is provably essential for the ising model on trees, Proc. Genet. Evol. Comput. Conf. (GECCO '05) (ACM, New York 2005) pp. 1161–1167
46.58 P.S. Oliveto, J. He, X. Yao: Analysis of the $(1+1)$-EA for finding approximate solutions to vertex cover problems, IEEE Trans. Evol. Comput. **13**(5), 1006–1029 (2009)
46.59 T. Jansen, P.S. Oliveto, C. Zarges: On the analysis of the immune-inspired B-cell algorithm for the vertex cover problem, Proc. 10th Int. Conf. Artif. Immune Syst. (ICARIS 2011) (Springer, Berlin, Heidelberg 2011) pp. 117–131
46.60 I. Wegener: Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions. In: *Evolutionary Optimization*, ed. by R. Sarker,

X. Yao, M. Mohammadian (Kluwer, Dordrecht 2002) pp. 349–369
46.61 F. Neumann, I. Wegener: Randomized local search, evolutionary algorithms, and the minimum spanning tree problem, Theor. Comput. Sci. **378**(1), 32–40 (2007)
46.62 D. Sudholt, C. Zarges: Analysis of an iterated local search algorithm for vertex coloring, 21st Int. Symp. Algorithms Comput. (ISAAC 2010) (Springer, Berlin, Heidelberg 2010) pp. 340–352
46.63 D. Sudholt: General lower bounds for the running time of evolutionary algorithms, 11th Int. Conf. Parallel Probl. Solving Nat. (PPSN 2010) (Springer, Berlin, Heidelberg 2010) pp. 124–133
46.64 P.K. Lehre: Fitness-levels for non-elitist populations, Proc. 13th Annu. Genet. Evol. Comput. Conf. (GECCO '11) (ACM, New York 2011) pp. 2075–2082
46.65 B. Doerr, D. Johannsen, C. Winzen: Drift analysis and linear functions revisited, IEEE Congr. Evol. Comput. (CEC '10) (2010) pp. 1967–1974
46.66 E. Cantú Paz: A survey of parallel genetic algorithms, Tech. Rep., Illinois Genetic Algorithms Laboratory (University of Illinois at Urbana Champaign, Urbana 1997)
46.67 M. Tomassini: *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time* (Springer, Berlin, Heidelberg 2005)