Capítulo 1

Introduction

Evolutionary Algorithms are a set of population-based stochastic search techniques able to solve optimisation problems in reasonable time. However, for very demanding applications and large problem instances the computational efforts of EAs can be high. Fortunately, EAs are rather easy to execute in a parallel fashion offering a straightforward way to improve scale-up properties [?]; the nature of EAs is inherently suited to be parallelised at population level. Nevertheless, a remaining challenge in parallel EAs is the central management of the evolutionary cycle (parent selection, reproduction, survivor selection) imposing limitations on scalability.

Within this context, we propose in this thesis the Evolvable Agent model, a new decentralised evolutionary algorithm that takes full advantage of a dynamic Peer-to-Peer environments.

As in any other distributed EA, a P2P EA faces two aspects, the algorithmic and the computational performance. The first is related to the structural adaptations that the algorithm suffers when deployed on several loosely coupled processors while the latter corresponds to the computational speedup that can be expected. In fact, distributed EAs are studied as a way of preserving genetic diversity while improving the runtime of the algorithm [?]. To this end, a good understanding on the physical platform, P2P systems in this case, can be leveraged in its design.

Peer-to-Peer (P2P) systems provide a powerful parallel infras- tructure able to constitute a single virtual computer composed of a potentially large number of interconnected resources without central control [Steinmetz and Wehrle(2005)]. Such a computing paradigm defines a rich set of topologies for the interconnection of nodes at application level, so-called overlay networks. The main idea behind a P2P EA is to designate each individual in the population as a peer and adopt a population structure defined by a P2P overlay network [Wickramasinghe et al(2007)]. Then any given individual has a limited number of neighbors and mating is restricted to the P2P neighborhood.

Interactions in such a spatially structured EA can be visualized as a graph where vertices represent individuals and edges the relationships between them [?]. In this sense, a traditional unstructured population (a.k.a. panmictic population) is represented as a complete graph, whereas other approaches define a richer set of population structures such as regular lattices [Giacobini et al(2005b)], toroid [Giacobini et al(2004)] or small-world [Giacobini et al(2005a)], [Preuss and Lasarczyk(2004)], [Giacobini et al(2006)].

Based on these studies, the Evolvable Agent model defines a decentralised population structure by means of the gossiping protocol newscast that behaves asymptotically as a small-world graph. The influence of such kind of structures in the environmental selection pressure of Evolutionary Algorithms is close to that in panmictic populations used by default in canonical approaches.

However, there are still many challenging issues in the parallelization of EAs in P2P systems. Questions such as *decentralization* (such a computation paradigm is devoid of any central server), *scalability* (since P2P systems are large-scale networks) or *fault tolerance* (given that resources are added and eliminated dynamically, often as a consequence of a decision from an user that volunteers CPUs under his control) become of the maximum interest and have to be addressed.

Therefore, the aim of this thesis will be study such issues in order to conclude the viability of the Peer-to-Peer Evolutionary Computation paradigm.

1.1. Motivation

Setting an adequate population size is a key to obtain good performances in EAs, that is, to preserve a good quality in the solutions without spending extra computational efforts. That way, a small problem instance requires a smaller population size than a larger instance of a more difficult problem. In order to obtain good performances, studies of scalability on the population size, such e.g. Thierens in [?] for GAs or Pelikan et al. in [?] for BOAs, show that the population size should roughly scale with an order $O(l^{\alpha})$, where l is the chromosome length and α is a constant that depends on the algorithm and the problem complexity. In the same way, Fernandes and Rosa show in [?] that the number of generations g also scales following an order $O(l^{\beta})$. Such factors influence the scalability of an EA as highlighted in Algorithm 1.

Therefore, the overall scalability order of an EA might be represented as $O(l^{\alpha+\beta+\zeta})$ by simply assuming a problem in which the evaluation function scales with a polinomial order $O(l^{\zeta})$ which is rather reasonable considering

Algorithm 1 Time consuming keys in the evolutionary loop

```
for $i = 0$ to $g$ generations do
    for $j = 1$ to $n$ individuals do
        $evaluate_{n_{ij}}(l)$
    end for
end for
```

realistic problems.

1.2. Structure of the thesis

Chapter **??**
Chapter **??**
Chapter **??**
Chapter **??**
Chapter **??**

All the above mentioned issues are discussed in detail within the different sections of this thesis. Section **??** reviews the state of the art literature related to P2P EAs, from the first attempts to the current lines of research. Section **??** provides some insights into the role that the population structure plays on the environmental selection pressure of an EA, so that, in Section **??**, the overall architecture of the *EvAg* model can be better understood as a spatially structured and decentralized EA. From that point, the rest of the chapter focuses on the properties of the *EvAg* model under the different issues of P2P EAs. In Section **??**, a *scalability* analysis of the model is presented, in Section **??** the *fault tolerance* is assessed under different *churn* scenarios and in Section **??**, the proposed P2P EA is applied to DOP. Finally, Section **??** extends the content of the chapter by exposing some conclusions and future challenges.

1.3. Evolutionary Computation

1.4. Parallel Architectures

Due to the great advance in the computer architecture research area, parallel computing platforms cover nowadays a wide range of systems going from special purpose architectures to interconnected off-the-shelf components. Given that P2P systems add virtualization to the physical architecture (as it will be detailed in the following chapters), there is no reason for preventing a

P2P model to be implemented in any kind of parallel architecture. Therefore, we review in this section the most common types of computing architectures.

In this section, we will refer to Parallel Architectures as the organisation of multi-processors architectures and memory access with the aim of load-balancing a computer program in an efficient way. Nevertheless, parallelism is present in computer architectures at very different levels. Even in sequential architectures, there is a certain degree of parallelisation at instruction level by the use of techniques such as instruction fetching, pipelining or super-scalar processing. This way, pure sequential computers remain more as a theoretical model based on the Alan Turing universal machine [?] than as a real computer architecture. In fact, Burks, Goldstine and von Neumann discuss in [?] one of the first proposal for implementing a computer which takes into account the use of 40 tubes in parallel to codify a word, quoting the authors: *in a parallel machine all corresponding pairs of digits are added simultaneously, whereas in a serial one these pairs are added serially in time.* That is, parallelism has been considered from the very beginning of computer architectures.

To a better understanding, Section 1.4.1 presents some of the most relevant ways of classification for parallel architectures and Section 1.4.2 describes different ways for interconnecting processors.

1.4.1. Parallel Architectures Taxonomy

A widely accepted and probably the most extended taxonomy for computer architectures is the one due to Flynn in [?]. It classifies computer architectures depending on the notions of instruction and data stream. Using both variables, computer architectures are classified in four main groups: Single instruction, single data-stream (SISD), single instruction, multiple data-stream (SIMD), multiple instruction, single data-stream (MISD) and multiple instruction, multiple data-stream (MIMD).

SISD represents sequential architectures processing a single instruction and a single data-stream at once. As commented at the introduction of this section, such architectures exploit a certain low level parallelism such as super-scalar processing. Despite nowadays multi-processors systems are imposing within the Desktop computer market, personal computers, mobile device processors, and generally, von Neumann based architectures are the most representative examples of SISD.

SIMD stands for those architectures able to process a single instruction over several data-streams. That is, several processors load the same instruction and apply it to different data. Such architecture is designed to take advantage of spacial parallelism in data as in the case of problems using matrixes, vectors or other types of regular structures. Masspas MP-2 and Cray T3D are some examples of computers using a SIMD architecture. However, it is also a common practice to insert SIMD instruction sets in SISD machines as the instruction set MMX in the Intel Pentium, SSE3 in Pentium 4 or the technology 3DNow! of AMD [?].

MISD refers to architectures that performs different operations on the same data. In practice, it is a quite uncommon architecture and just few computers implement it with an special purpose such as task replication in fault-tolerant computers or the systolic computing architecture proposed by Gupta et al. in [?] for image registration in real time.

MIMD is one of the most extended architectures to achieve parallelism in which processors execute different instructions on different data. Processors perform the operations asynchronously requiring some mechanisms for intercommunication in order to jointly perform a given task. Despite Flynn's taxonomy does not explicitly mention, either shared memory or distributed memory categories belong to MIMD architectures.

Despite being widely accepted, Flynn's taxonomy leave out of the classification some computer architectures. Hence, some additional classifications have been proposed as the one of Kumar et al. in [?] that distinguishes between distributed and shared memory architectures taking into account the architecture of the memory address space.

Shared-Memory architectures. It refers to those architectures including specific control for the concurrent access of processors to a common memory address space. They can be divided into two subgroups:

- **Shared-Memory multiprocessors**

  In this case, the memory is physically common to all processors and the access is synchronised at a hardware level.

- **Virtual shared-memory multiprocessors**

  Every processor has its own private memory, although it is virtually accessible from the rest. Synchronisation is managed at a higher level typically by the operative system.

Distributed Memory architectures. This architecture is conformed by a group of independent computers which are interconnected by means of a network. Every computer has its own processor, memory address space and an input/output system. In contrast with shared-memory systems, communication between processors is due to the input/output system using message-passings interfaces.

1.4.2. Interconnection topologies

As an answer for the different requirements of efficiency and robustness in the realm of parallel architectures, the interconnection between processors can adopt multiple topologies either in the physcal wiring or the logical structure. This way, a decision on the implementation of a given topology has to consider specific objectives of the architecture. For instance, a complete graph topology (i.e. Fig. 1.4 (b)) offers a very efficient way for the communication inter-processor, however, for a number of processors n, the number of links scales with an order $\frac{n(n-1)}{2}$ making of it an unfeasible solution in large-scale architectures.

This section reviews some of the most commonly used topologies in parallel architectures showing their main features.

Star topology As depicted in Figure 1.1, in this topology all processors communicate via a central one acting as a hub or master. This is a typical configuration in local area networks and scalability is mainly restricted by the hub performance.

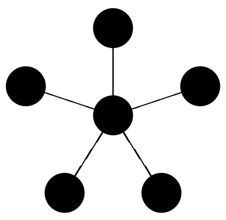
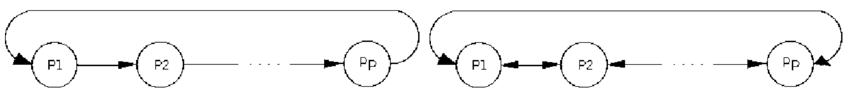


Figura 1.1: Star topology.

Ring topology In architectures using a ring topology, the processors are interconnected in a point-to-point loop. The sense of communication can be unidirectional as in Figure 1.2(a) (either in a clockwise or a counterclockwise sense) or bidirectional as in Figure 1.2(b). Under a heavy load, it performs better than a star since it does not require of a central server, however, any malfunctioning component can block the entire network run.

A well-known instance of such interconnection topology is the token ring architecture [?]. In addition, it constitutes a good example about how there can be differences between the logical and physical organisation within the same architecture. In this case, token ring is logically organised as a ring whereas physically wired as a star.



(a) Unidirectional Ring. (b) Bidirectional Ring.

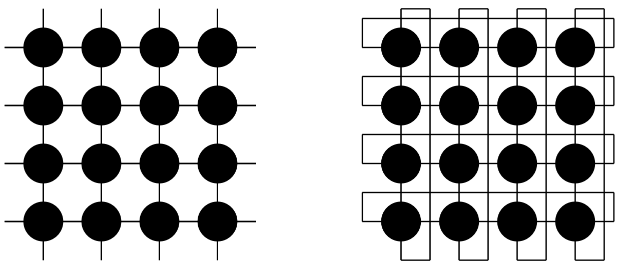
Figura 1.2: Ring topologies

Grid topology Processors are organised as a grid of n rows by m columns as depicted in Figure 1.3. Whenever the extremes are connected in a loop, the topology is known as toroidal grid.

Despite grids usually consist in a two dimensional mesh, dimensions can be extended to create hypercubes or being reduced to a single dimension in the case of ring topologies.

Computer clusters are commonly interconnected following one of the variants of grid topologies, e.g. the symmetric architecture proposed by Akers and Krishnamurthy in [?].

Graph topology Graphs are the generalisation for any kind of topology. That is, any other interconnection type is a sub-type of a graph. Nevertheless, interconnection topologies referred as graph use to be applied



(a) Grid topology. (b) Toroidal grid.

Figura 1.3: Different grid topologies.

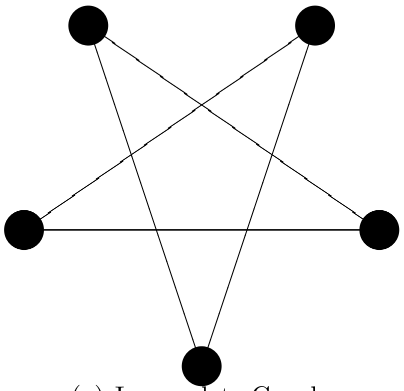
for describing complex or irregular topologies and regular topologies that can not be classified otherwise.

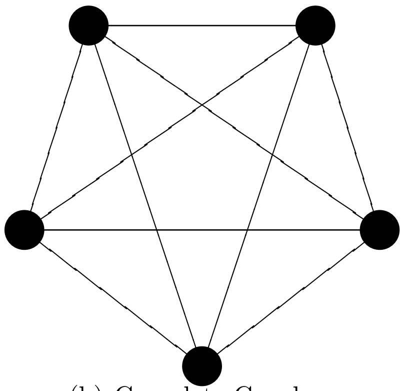
Figure 1.4 shows three common types of graph interconnections:

a) Incomplete graphs stand for those architectures in which processors are connected to more than a single processor using a point-to-point link. Unlike in a ring, redundant links allow some resilience to links/nodes failures without the entire collapse of the system.

b) Despite having a good robustness to failures, complete graphs are just used in architectures with few processors. The number of links scaling with a quadratic order turns such a topology into an unfeasible approach for interconnecting a large number of processors.

c) Complex graphs are a special case of incomplete graphs that promote fault-tolerance while preserving a good scalability. Réka and Barabási present an exhaustive analysis in [**?**] where they prove the robust and scalable behaviour of several types of complex networks. Taking into account such properties, complex graphs interconnections are commonly used in large and unreliable architectures as they are P2P systems.


(a) Incomplete Graph.


(b) Complete Graph.

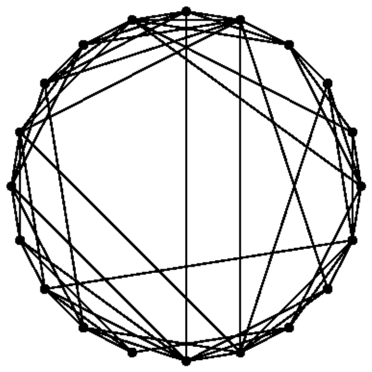

(c) Complex Graph.

Figura 1.4: Some examples of graph topologies.

1.5. Conclusions