Peer-to-Peer Networking and Applications manuscript No.
(will be inserted by the editor)

P2POEM: Function Optimization in P2P Networks

Marco Biazzini · Alberto Montresor

the date of receipt and acceptance should be inserted later

Abstract Scientists working in the area of distributed function optimization have to deal with a huge variety of optimization techniques and algorithms. Most of the existing research in this domain makes use of tightly-coupled systems that either have strict synchronization requirements or completely rely on a central server, which coordinates the work of clients and acts as a state repository. Quite recently, the possibility to perform such optimization tasks in a P2P decentralized network of solvers has been investigated and explored, leading to promising results. In order to improve and ease this newly addressed research area, we designed and developed P2POEM (P2P Optimization Epidemic Middleware), that aims at bridging the gap between the issues related to the design and deployment of large-scale P2P systems and the need to easily deploy and execute optimization tasks in such a distributed environment.

Keywords P2P distributed computing, distributed function optimization, P2P applications

1 Introduction

The job of scientists working on function optimization often involves two different aims [4]: on the one hand, they are required to benchmark new optimization algorithms against a large collection of existing problems; on the other hand, when focusing on a specific optimization problem, they often ought to tackle it by means of a large variety of optimization techniques, to obtain the best possible

M. Biazzini
Université de Nantes, France
LINA
E-mail: marco.biazzini@univ-nantes.fr

A. Montresor
Università degli Studi di Trento, Italy
Dipartimento di Ingegneria e Scienza dell'Informazione
E-mail: alberto.montresor@unitn.it

results. Adding the distributed dimension to such a challenging scenario brings in undoubtable pluses, but only at the price of increasing its complexity while decreasing its tractability.

Distributed optimization has a long research history [36]. Most of the existing work assumes the availability of either a dedicated parallel computing facility, or specialized clusters of networked machines that are coordinated in a centralized fashion (master-slave, coordinator-cohort, etc.). While these approaches simplify management, they normally show severe limitations with respect to scalability and robustness.

The application scenario we have in mind is different: a potentially large organization that owns, or at least controls, several hundreds or even thousands of personal workstations, and wants to exploit their idle periods to perform optimization tasks. In such systems, high level of *churn* may be expected: nodes may join and leave the system at will, for example when users start or stop to work at their workstations.

Such a scenario is not unlike a Grid system [23]; a reasonable approach could thus be to collect a pool of independent optimization tasks to be performed, and assign each of them to one of the available nodes, taking care of balancing the load. This can be done either using a centralized scheduler, or using a decentralized approach [19].

An interesting question is whether it is possible to achieve a comparably good (if not better) performance with an alternative approach, where a distributed, peer-to-peer (P2P) algorithm spreads the load of a *single* optimization task among a group of nodes, in a robust, decentralized and scalable way. We can rephrase the question as follows: can we make a better use of our distributed resources by making them cooperate on a single optimization process? Two possible motivations for such approach come to mind: we want to obtain a more accurate result by a specific deadline (focus on quality), or we are allowed to perform a predefined amount of computation over a function and we want to obtain a quick answer (focus on speed).

Quite recently, this question has been answered positively by several works of ours [6, 7, 10] and by other relevant contributions [21, 33], that solve specific function optimization problems based on P2P approaches. Instead of requiring a specialized infrastructure or a central server, such systems self-organize themselves in a completely decentralized way, avoiding single points of failure and performance bottlenecks. The advantages of such approach are thus extreme robustness and scalability, and the capability of exploiting existing (unused or underused) resources.

But there are a few drawbacks in existing literature. First of all, each of these studies is conducted in isolation, trying to adapt a specific optimization technique to the peculiar characteristics of a P2P environment. It would definitely be helpful to have a distributed framework, hosted on every participant node that can ease the burden of designing, developing and deploying novel optimization algorithms in such environment. This paper proposes P2POEM [9], a framework that addresses both the issue of the interchangeability of the optimization components and the specificity of a P2P environment. Making use of the epidemic paradigm for node-to-node communication, our simple yet versatile architecture can ease the effort of porting and executing optimization tasks in a decentralized network of solvers.

Second, existing P2P optimization algorithms have been evaluated mostly by means of network simulators [6,7,10,21] or through small systems [33]. Experimental evaluations of real deployments are thus lacking. This papers fills this void by presenting results obtained in a real deployment. Popular optimization algorithms are evaluated with respect to problems presenting diverse hardness and in different network scenarios. Results are discussed and compared with results available in the literature. At the same time, the main features of the framework are assessed.

The rest of the paper is organized as follows. The goals of the framework are discussed in Section 2, while a description of the framework architecture is provided in Section 3. Details about the main components of the architecture, i.e. the function optimization service and the communication service, are included in Sections 4 and 5. Experimental results based on our framework show interesting outcomes of a distributed Differential Evolution algorithm, both in a simulated and in a real network deployment. They are presented in Section 6. Then we recall and comment relevant related work in Section 7 and draw our conclusion in Section 8.

2 Problem and Goal

When optimization tasks are distributed among a collection of independent machines, the obvious goal is to obtain a speed-up. But many intriguing possibilities arise, that lead to interesting questions. Can existing algorithms be plugged in a distributed environment? Can multiple instances of the same algorithm cooperate? Can multiple algorithms share their results? How multiple instances are organized, and how do they communicate? Are there classes of optimization problems that are more appropriate for a distributed environment?

Based on the P2P paradigm, P2POEM aims at helping scientists to answer these questions and to bridge the gap between the distributed system and the function optimization fields. P2POEM has two main objectives: (i) to couple a general optimization framework to a P2P middleware that can handle different P2P network topologies and (ii) to offer to function optimization practitioners a framework that requires them to add only a limited amount of wrapping code to plug their legacy or novel optimization code in such an "unfamiliar" environment.

To better illustrate the goals, we need to define some basic terminology. We target distributed systems composed by a (potentially large) number of *solvers*, each of them being a running instance of an *optimization algorithm*. Each solver iteratively evaluates one or more points of an *objective function*, with the goal of finding a better objective value (minimum or maximum). Solvers may share information with each other about the progress of the optimization process. Subsequent iterations of the local instance of the algorithm may depend not only on the objective values that have been found locally, but also on the information coming from other solvers.

When it comes to making optimization algorithms collaborate and share information, several aspects equally matter.

– Due to the huge variety of existing and possible optimization algorithms, a framework has to be generic enough to suit most (if not any) of them. It must be designed in such a way that no substantial refactoring of existent code is required to plug an algorithm in the system, meanwhile making the

execution of the algorithm not unusual to the user, with respect to a centralized environment.
- Another requirement is *interoperability*. Different optimization algorithms, having very little in common, should be able to cooperate to solve the same problem with little or no user intervention. On the other hand, the same optimization algorithm should be able to operate on a large number of different problems with limited adjustments.
- No striving should be required to handle and set up the P2P network of solvers. A basic parametrization of the amount of communications to be enabled should be all that the user needs to know about the way the solvers spread information throughout the network. Moreover, the framework should provide a clear and standard way of handling the acknowledgment of relevant information and using it properly.
- A precise and versatile mechanism has thus to be provided, that can handle in a consistent way the process of information sharing between different solvers. This has to be done according to the user's decisions and transparently handling the differences among the optimization processes, which run in different network nodes.

Several issues are hidden in this brief list of requirements. While no solution can be considered unanimously the best possible answer to all of them, we believe that a significant improvement can derive by using our framework. P2POEM consists of an epidemic protocol, with a set of interfaces and services, written in Java. It comes in two flavors:

- as a framework ready to be plugged in PeerSim [31], a widely used P2P network simulator specialized in the simulation of gossip-based protocols, and
- as a framework ready to be plugged in Cloudware [30], a real-world implementation of the main PeerSim concepts for real deployments in large-scale networks.

An important advantage of this framework organization is that the user can execute optimization tasks both in a simulated and in a real P2P environment without changing a single line of code in the actual implementation of the algorithms and/or of the functions already deployed.

Furthermore, by exploiting the features implemented in the two aforementioned P2P services, P2POEM makes it possible for the user to pay attention to some novel and interesting design issues of a P2P distributed optimization task:

- *Synchronization*. Depending on the algorithms and on the way their computations are distributed, there may be performance decays, due to synchronization. On the other hand, not all the algorithms have strict synchronization requirements. For those which have, it is likewise fundamental to know *when* the synchronization is strictly needed and when it is not. Thus, it is important to achieve a good understanding of the relation between performance drifts and ratio of communication events, in order to find, for each kind of algorithm, the maximal amount of synchronization occurrences that does not penalize the performance.
- *Information sharing*. How much information has to be shared among the distributed instances of an algorithm? Of course, this depends on which algorithm is running, but also on how many of its instances are active in the network at

the same time. A careful consideration of *what* has to be shared and what can be kept as local is crucial for each kind of algorithm.
– *Convergence and communication rate. How often* shared information must be updated? The spreading of the information relies on both the frequency of message exchanges and the interconnection topology. The behavior of the same algorithms in different P2P topologies may present significant differences. A tradeoff between a good information exchange and a rapid advancement of each individual search process has to be found. Moreover, in a totally decentralized system is due to pay attention to the effects of the delays in propagating the updates.

Exploiting the capabilities of the existing applications it relies on, P2POEM provides seamless interaction with the P2P environment and a clean and easy-to-get set of interfaces to quickly plug algorithms to be executed and problems to be solved.

3 Design and Architecture

We provide here an high-level description of the main architectural components of P2POEM. Taking into account the manifold issues concerning the creation and maintenance of a P2P network on the one side, the consistency and performance requirements of an optimization task on the other side, we devised a three layer architecture of independent modules:

– *The topology service* is responsible for creating and maintaining an adequate overlay topology to be used by the other layers to communicate information about the search space. As examples, consider a random topology used by a gossip protocol to diffuse information about global optima; a mesh topology connecting nodes responsible for different partitions of the search space; but also a star-shaped topology used in a master-slave approach.
– *The function optimization service* evaluates the target function over a set of points in the search space. These points are opportunely selected taking into account both local information (provided by this module, based on past history) and remote information (provided by the communication service). This module heavily depends on the particular optimization mechanism adopted.
– *The communication service*, as the name implies, enables nodes to communicate, with the goal of exchanging information about the current best solution and the space that has been explored, coordinating their actions in future runs of the function optimization service.

The three modules cooperate to perform the optimization task and spread useful information among peer nodes. Figure 1 illustrates the three layers architecture and the interaction between the services. The numbers in the figure illustrate the steps carried out by the system, whenever the optimization service decides to communicate results to another peer layer, that is running in a different node.

At the state of the art, P2POEM provides integrated implementations of several optimization heuristics (Particle Swarm Optimization [22], Reactive Affine Shaker [11], various Differential Evolution [35] operators) and hyper-heuristics (a distributed tabu algorithm and several racing policies that we have proposed in [6]), along with abstract classes that greatly ease the burden of plugging legacy code in.

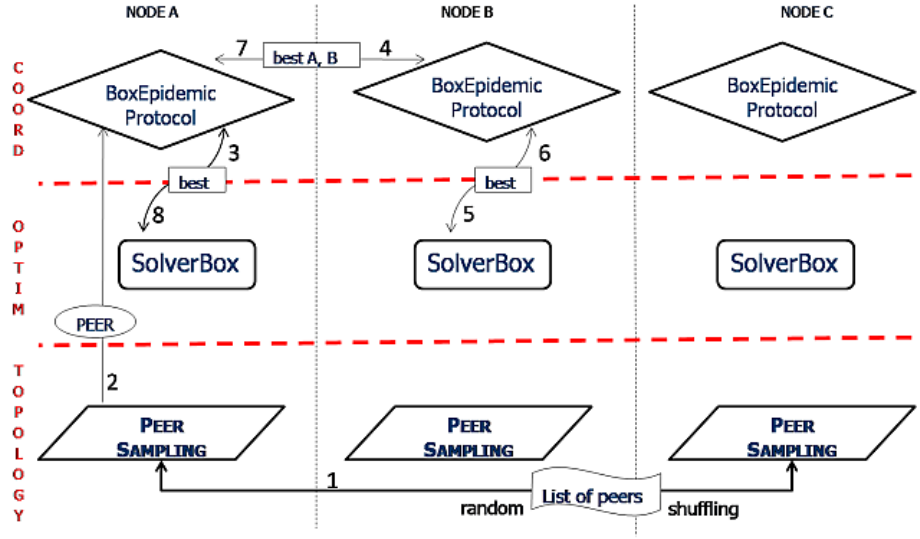


Fig. 1 High level architectural view and communication steps

The first module (topology service) is implemented by PeerSim and Cloudware. They provide (either in simulation or in a real deployment) basic functionalities like a selection of robust topologies, bootstrap mechanisms to let node join and leave, and few basic transport layers. Concerning the overlay network topology, the current implementation of p2poem relies on the newscast protocol to select peers from a random topology [20]. Based on our tests and acquired experience, we felt no need so far to implement other structured topologies, though obviously this may be an easy additional feature to be added in the future. The task of implementing a fully integrated distributed optimization framework is associated with the other two modules, that are described in the following sections.

4 Function Optimization Service

The function optimization service contains two core components: a collection of *entity interfaces*, that must be implemented by the actors of an optimization task, and the SolverBox, whose goal is to orchestrate the execution of these actors. In the rest of this section, we present these components in detail, starting from the main characters acting in the optimization play.

4.1 The entity interfaces

Recalling the description of a generic optimization process provided in Section 2, the two main characters in the game are the function to be optimized and the algorithm that searches the optimal function value. Of course, the huge variety of currently available optimization algorithms and their remarkable structural complexity force us to consider subtler distinctions in our design. We therefore come to a set of four interfaces: Solver, Algo, Meta and Function.

– Interface Solver is the core component of the framework, the one that implements the algorithmic procedures to evaluate the objective function, keeps track of the results and moves toward the objective. The internal complexity of a solver may vary greatly, depending on the implementer's choices. It may be worthwhile having a 'very basic' solver that repeatedly applies a given Algo (in which the real complexity resides) and that simply keeps track of the best solution found so far. Or the implementer might rather like to concentrate everything in the solver and even make no use of any Algo at all. This interface leaves the greatest freedom to the implementer, which is allowed to shape the solver in the most suitable fashion, as long as its interface's methods are properly implemented.
– Interface Algo represents a basic algorithm whose design requires less effort and complexity than a Solver. Moreover, it is meant to be used by a Solver to perform a part of its task. This interface allows the implementation of reusable procedures that can be exchanged with one another (paying attention all of them are compatible with the Solver), e.g. building blocks or heuristics which a Solver can choose among or which a Meta can select and assign to the Solver at each iteration.
– The Meta interface can be useful to plug in the framework meta- or hyper-heuristics [12], a racing algorithm [28], a portfolio selector [17], etc. A Meta object is a component whose goal is to choose among different Algo instances and assign the chosen one to the Solver, for it to be used in the next iteration. No function evaluation is usually required to perform such a task. The implementer is free to choose how to handle the pool of available Algo instances, as well as any functional detail.
– Interface Function corresponds to the function to be evaluated in the framework; this interface declares the few necessary methods for this component to interact with the solver and to describe the operational domain of the problem.

The set of methods of each interface has been conceived as the smaller and simpler set of operations that can guarantee the appropriate interaction among the entities and with the other nodes in the network. According to our research experience, we may say that to adapt an existing, previously implemented algorithm to the proper P2POEM interface, normally requires to write the implementation of few getter and setter methods and the modification of the signatures of the main functions. Thus, no refactoring of the main body of the algorithm is required for it to be plugged in the P2P network of solvers.

4.2 The SolverBox

The SolverBox takes care of the execution of the local solver and provides it the information shared by other solvers in the network. It may be seen as a proactive 'wrapper' of the other entities, that coordinates their work and transparently handles the provisioning of data coming from remote nodes, as well as internal updates resulting from local Solver or Meta iterations.

The actual implementations of the SolverBox on PeerSim and Cloudware differ on various aspects, mainly because in the former the SolverBox is an epidemic protocol [14] among the others, whereas in the latter it is an independent object, running a thread of its own. Notwithstanding this major dissimilarity, the

actions performed are the same and the general contract with the other entities is absolutely identical.

The SolverBox keeps its own records of the best results found so far by any node in the network. These records are updated every time a new best result is found by the local Solver or received from a remote node. This way information can be shared among every node and 'good news' can be spread, while leaving to the specific entities the decision to take advantage of them (when, how often, etc.) or not. Local and gossiped updates may be made available to the internal entities by means of the public methods defined in the entity interfaces.

The SolverBox iterates the same sequence of instructions until a user defined termination criterion is met (usually a given number of function evaluations to be performed). We may briefly summarize the series of actions performed in a SolverBox iteration in the following way:

1. Process any new message from other nodes in the network, updating both the SolverBox and the local entities as needed;
2. If a Meta object has been defined, make it choose the next Algo to be used and assign it to the Solver;
3. Perform the next Solver step, thus evaluating the Function and updating the internal Solver state;
4. If a Meta object has been defined, update it with the latest Solver outcomes;
5. If a new best result has been found in the latest Solver step, update the SolverBox records.

At the end of each iteration, a communication event occurs, which we explain in detail in the next section. The way the SolverBox interacts with the Solver and the Meta instances, as well as any management mechanism are completely transparent to the user, who just needs to know about and take care of the optimization component(s). The only additional effort we require from the user is to provide the values of three basic parameters:

– `maxTs` The maximum number of function evaluations to be performed by the solver run in this box (`long`).
– `push-pull` Whether the epidemic communication policy of the SolverBox is push-pull or not, as explained in Section 5 (`boolean`).
– `solverPool` The number of solvers in the configuration among which the actual solver to be instantiated is chosen (`int`).

The first parameter is quite straightforward. About the second, we just remark that this setting only concerns the SolverBox, not the peer sampling nor any other epidemic protocol belonging to PeerSim or Cloudware that may serve the framework. The third parameter deserves some detail. Any Solver or Meta instance can be configured in the way the user prefers (exploiting the configuration parsing capabilities of PeerSim and Cloudware is of course an easy possibility), but their names must be included in the SolverBox configuration file, so that the SolverBox can know what has to be instantiated and run. To make it quicker to deploy and run experiments in which different Solvers — possibly matched against different Metas — run in different nodes and cooperate, we provide a way to automatically choose uniformly at random a Solver from a specified pool. The `solverPool` parameter gives the cardinality of this pool. In the current implementation of the SolverBox, this selection is made at each node uniformly at random among the Solvers listed in

a configuration file. Further improvements of this mechanism are easy to provide as needed. □

Along with the SolverBox and the entity interfaces we described, P2POEM features several abstract classes that provide standard implementations for most of the methods required to implement the interfaces. These may be considered as useful facilities to plug legacy or novel code in the general architecture. Based on these abstractions, a basic StepperSolver is also provided, that wraps all the operations needed to iteratively use an Algo and records the outcomes. By extending this solver, it is possible to plug and use a previously written algorithm in a very short time.

5 Communication Service

As above mentioned, the other main feature of P2POEM is to provide a communication policy between solvers in the network, thus implementing the *communication service* layer of our architecture. The actual implementation is strictly dependent on the application P2POEM relies on, being it PEERSIM or CLOUDWARE. However, in both cases the epidemic communication paradigm [14] is adopted and gossip messages are sent according to one of the following *exchange policies*:

- *push*: a node spreads information about its state and processes received messages without ever replying to the sender;
- *push-pull*: a node spreads information about its state and may reply, if needed, to received messages.

Obviously, depending on which of the exchange policies has been selected, the user might observe a direct impact not only on the communication overhead, but also on the performance of the optimization task. The quality of the final solution may or may not vary, making one choice clearly preferable to the other, or suggesting a diversification related to the behavior of the specific solver that is running. The amount of information sent in a reply may also be different from that of a pushed message.

Another factor that may make a difference is the *gossiping rate*, i.e. a measure of how often an epidemic message is sent, measured in relative terms with respect to how often a function is evaluated by a given solver. Notwithstanding the actual algorithm running on the single nodes, we may always propagate the current best solution to all nodes. Whenever a node performs a gossip communications with peers, it sends the best solution it knows of. When it receives a message, it updates its own current best solution.

We assume the period of gossip to be not less than one function evaluation, which presupposes that the function is non-trivial and takes a sufficiently long time (in the order of a second or more) to compute. It can be shown that the time to propagate a new current best solution to every node this way takes $O(\log N)$ periods in expectation where N is the network size [32].

In any case, the gossiping rate can be chosen by the user, enabling a finer control of the general information sharing policy. Despite the various technical details related to the maintenance of the communication strategy, the usage of the communication service requires no deep understanding from the user. There is no further necessity to fiddle with parameters or network configurations, once the user has decided the two basic settings:

	data name	data type
mandatory data	Best Value	double
	Best Position	double[]
	Best Algo	String
	Best Timestamp	long
optional data	Solver Name	String
	Solver Data	Object[]
	Meta Name	String
	Meta Data	Object[]

Table 1 P2POEM's epidemic message content.

– whether the type of epidemic communication is push or push-pull and
– which is the probability that the node A sends a message to the node B at the end of each A's SolverBox iteration.

The way a node behaves whenever a communication event occurs deserves further details. Both Solver and Meta are associated with a *gossip probability* parameter, respectively p_{Solver} and p_{Meta}. At the end of each SolverBox iteration, random numbers r_1 and r_2 in $[0, 1[$ are generated, and a message is created containing (i) Solver information if $r_1 < p_{\mathsf{Solver}}$, and (ii) Meta information if the Meta component is present and $r_2 < p_{\mathsf{Meta}}$. Clearly no message is created if neither Solver nor Meta information has to be sent.

Thus the load on the network directly depends on the optimization algorithm, both with respect to the communication pace — number of messages per function evaluation — and the bandwidth use. The SolverBox does not require any additional information to be sent, beyond what is configured for the Solver and the Meta.

As summarized in Table 1, every message contains the best values known by the sender, along with the Algo that has found them (if any). Then it can also contain some entity-specific data provided by the Solver and/or the Meta. A receiver sends back a reply to the sender node if and only if the best value gossiped by the sender is worse than the current best in the receiver and the chosen epidemic policy is push-pull.

In this way the user is able to control the information spreading rate among the various distributed components, choosing the best option with respect to their characteristics, or depending on other relevant aspects (e.g. the overall number of nodes in the network, or the number of nodes hosting the same Solver/Meta as opposed to how many nodes host a different one, etc.).

Whenever a message is received, it is appended to an 'inbox' queue. All the messages in this queue are processed at the beginning of the next SolverBox iteration, until the queue is emptied. Synchronization of reading and writing events is considered and no message loss occurs. While processing each of the messages in the queue, the SolverBox updates its current best values as needed. Then it makes the entity-specific data in the message available to its own Solver/Meta if and only if the data have been provided by an instance of the very same class of Solver/Meta. This way a peer-entity exclusive communication among instances of the same class, but running in different nodes, is also provided.

This "dual-level" communication mechanism is important for different reasons. First, it provides a general data sharing format that all nodes can exploit to spread information among all optimizers, being them instances of the same Solver or not.

Thus, it actually allows different algorithms on different nodes to share relevant basic information and exploit it, according to their own peculiar strategy. Second, it enables a richer communications only among those peer-entities — instances of the same classes running on different nodes — that can actually understand it. The very same Solver or Meta instances can share ad-hoc information, thus obtaining relevant improvements. If by chance entity-specific data come to a different kind of Solver or Meta, no harm will occur to the optimizer, that will simply exploit the general information, not even being aware of the existence of other "unintelligible" data.

It is worth noticing that the user is not required to know about — nor to fiddle with — message composition at sender nodes or message parsing at receiver nodes. All it is required to the user is to implement the proper methods to handle specific Solver or Meta data within the implemented algorithms, as we detailed in Section 4.1.

We can model the expected communication cost of P2POEM as follows.

Let P_s be the probability given by the value of p_{Solver} and P_m the probability given by the value of p_{Meta}. Equation 1 defines the probability P_M that a message is sent by a peer after any function evaluation.

$$P_M = P_s + P_m - (P_s * P_m) \tag{1}$$

As it is clear by looking at Table 1, the size of each message depends both on the optimization algorithm and on the function being optimized. Higher dimensional functions may require more data to identify a point of evaluation (mandatory data). Then the algorithm designer can take any decision about the amount of information to be shared among different Solver or Meta instances (optional data). Let L be the length of the mandatory data, L_s the length of the optional data provided by the Solver and L_m the length of the optional data provided by the Meta (assumed to be equal to 0 if no Meta is defined). We may compute the expected bandwidth use B_p of a single peer after any function evaluation as showed in Equation 2 :

$$B_p = (L * P_M) + (P_s * L_s) + (P_m * L_m) \tag{2}$$

This accounts for messages that are pushed by a peer to another. If a *push-pull* policy is enabled, there will be some replies to these messages too. The current implementation of the SolverBox automatically replies to a message only if the local current best value is better than the one sent within the message. The reply always includes mandatory data only, meaning that it does not involve in any way neither the Solver nor the Meta (because the update to the current best value, received with the reply, is transparently handled by the SolverBox itself). Of course this is a choice that can be modified in several ways, for instance by letting the Solver decide if, how and when to reply to a direct push. So far we preferred not to add such a complication for the framework users (being the Solver what concerns them the most). This anyway implies that, at the present, it is not possible to draw a model of the communication overhead generated by these replies.

Few more notes about the sending/receiving process. PeerSim is a single-threaded application that emulates simple transfer protocols. This means that the optimization job and the communication events are actually serialized and can be fully controlled and handled by the user. Quite a different situation we have in Cloudware. This application is inherently multi-threaded and of course

Notation	Linear combination
DE/rand/1/*	$T = E + Fact \cdot (A - B)$
DE/rand/2/*	$T = E + Fact \cdot (A + B - C - D)$
DE/best/1/*	$T = Best + Fact \cdot (A - B)$
DE/best/2/*	$T = Best + Fact \cdot (A + B - C - D)$

Table 2 DE operators

the P2POEM implementation on CLOUDWARE fully exploits the possibility to concurrently perform optimization tasks and communications by using different threads. However, this makes things slightly less predictable than in a simulated environment, because threads are managed differently by different operating systems and their actual scheduling can vary significantly, depending on many factors (hardware architecture, current machine workload, job scheduling policy, ...), none of which may be predicted or controlled by P2POEM users.

Nonetheless, our tests have shown that, under fair workload conditions on the various network nodes, the behavior of a P2POEM application on CLOUDWARE is remarkably stable. Once a message is generated/received by a transport protocol, for it to be delivered by/to the SolverBox only few iterations may be required.

6 Experimental Results

Differential Evolution (DE) is a well-known and broadly studied method for the optimization of multidimensional functions. We briefly sketch here the basic ideas behind the algorithm and refer the reader to [35] for a detailed description.

Basically, DE is a scheme for generating trial parameter vectors, commonly denoted as *individuals*. Individuals are vectors of coordinates in the objective function domain, thus denoting a point in which the function may be evaluated. A group of individuals is called a *population*.

DE generates new individuals by linearly combining two or more members of the current population, through the execution of a specific *operator*. The set of available operators defines different DE variants; an example is given in Table 2, which summarizes the most common instances. In this table T is the new individual generated by the combination of other individuals. A, B, C, D, E are individuals chosen at random within the current population, while *Best* is the individual representing the current optimal solution. $Fact \in [0 \dots 2]$ is a user-defined real constant factor that controls the amplification of the differential variation.

Operators are usually classified according to the notation proposed in [35], that is $DE/x/y/z$, where:

- x specifies the vector to be mutated; it can be either *rand*, meaning a random selection from the whole population, or *best*, meaning that best known solution *Best* is used.
- y is the number of difference vectors used (1 or 2).
- z denotes the way the probability of performing the linear combinations is drawn; it can assume either the value *bin*, that corresponds to independent binomial experiments for each dimension; or the value *exp*, that corresponds to a conditional probability for each dimension w.r.t. the precedent one.

Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), the DE algorithm generates trial individuals by repeatedly applying one or more operators to a given population and substitutes population members if their quality is greater. No global probability distribution is required to generate trial individuals while moving toward the optimum (being it a maximum or a minimum), thus the algorithm is completely local and self-organizing. It is easy to see that, provided a way to properly distribute the overall population, DE is a suitable candidate for distributed optimization tasks.

It is well known that distributed versions of evolutionary algorithms can outperform sequential ones (compared under the same computational effort, measured in number of function evaluations). This motivates the growing amount of research about the ways the local populations can be made interact with each other in order to improve the final result.

We have already presented in a previous work of ours some results about performing distributed optimization by means of DE in P2P simulated networks [8]. Several nodes cooperate in a P2P fashion to optimize a function, each of them running an instance of DE on a local population. At the end of every DE iteration (selection of individuals, application of an operator, evaluation of the function on the resulting individual, solution quality assessment) each node sends a message to another node, randomly chosen by a peer sampling service. The message contains the best individual found so far. This way the the solvers share useful information and may use the updates coming from other peers to improve their own solution.

Such an algorithmic design is a good "test case" for P2POEM. In order to assess the effectiveness and the reliability of our framework, we performed experiments with legacy code, replicating the original settings both in simulated and real deployment. We recall that the main goal here is not the absolute performance of the DE algorithm itself, but to verify that :

- porting legacy code in our framework requires no modification of the core optimization algorithm;
- thanks to P2POEM, the same optimization code can be used both in a simulated environment and in a real deployment;
- the results obtained in the real deployment are similar to the outcomes of the simulations, meaning that our multi-thread gossip middleware causes no harm to the computation;
- the performance of distributed DE algorithms in a real deployment is comparable to that of the same algorithms already tested in a simulated environment.

In the following, we give a more technical description of our experimental settings and present some results.

6.1 Experimental settings

The results we present are obtained by using the P2POEM implementation both on PEERSIM and CLOUDWARE. We rely on their peer sampling module [20] to provide nodes with random sample taken from the entire system population.

While in a simulated environment the actual distributed deployment and the bootstrap phase are of no concern, some decisions have to be taken when moving to a real network. We run our experiments on the Amazon EC2 cloud computing

	Function $f(x)$	D	$f(x^*)$	K
Rosenbrock	$\sum_{i=1}^{9} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$	$[-100, 100]^{10}$	0	1
Griewank	$\sum_{i=1}^{10} x_i^2/4000 - \prod_{i=1}^{10} \cos\left(x_i/\sqrt{i}\right) + 1$	$[-600, 600]^{10}$	0	$\approx 10^{19}$
Rastrigin	$100 + \sum_{i=1}^{10} \left(x_i^2 - 10\cos(2\pi x_i)\right)$	$[-5.12, 5.12]^{10}$	0	$\approx 10^6$
Cassini1	description available from ESA at `http://www.esa.int/gsp/ACT/inf/op/globopt/evvejs.htm`			
Cassini2	description available from ESA at `http://www.esa.int/gsp/ACT/inf/op/globopt/edvdvdedjds.htm`			

Table 3 Test functions. D: search space; $f(x^*)$: global minimum value; K : number of local minima.

environment[1], deploying 16 machines (Standard Small Instances). While we vary the total number of solvers in the network, we always run an equal number of processes on each machine. We thus have a fair load among the (identical) machines, also considering the overall number of threads concurrently running within the same experiment.

Concerning the bootstrap phase, we adopt a simple strategy to reduce the start up delay of the computation among the solvers, that we believe viable in both a closed environment like Amazon EC2 and an open environment like a P2P network. All the peers (being them processes running on different machines or not) are given the network address of a *bootstrap-host* and periodically ask it for an initial set of neighbors. The optimization task only starts on each peer after this set has been received. This way, all nodes receive the first set of neighbors and start the computation within very few seconds. Once a node has received the initial neighbor set, its local peer sampling service maintains a local random sample of the network in the usual way and the bootstrap-host is no longer contacted.

Considering the *optimization service*, we adopt the source code already used in our previous work [8] as a legacy and easily plug it in the framework:

- the DE scheme implementing the Solver interface
- the various DE operators implementing the Algo interface
- the objective functions implementing the Function interface

In each node the SolverBox follows the steps described in Section 4.2, making DE optimizing the objective function by using the given operator. Though fully implemented, no Meta entity is used, so we have a network of DE solvers using the same operator from the beginning till the end of each experiment.

Information are spread by an epidemic protocol (*communication service*) that works as described in section 5, following a *push* policy.

We selected well-known test functions, as shown in Table 3. Rosenbrock is a non-trivial ten-dimensional unimodal function. The rest of the functions are multimodal. Griewank is a spheroidal ten-dimensional function with superimposed high frequency sinusoidal "bumps". Rastrigin is a highly multimodal ten-dimensional function whose local minima are regularly distributed. It is a difficult test case for most optimization methods.

Cassini1 and Cassini2 are realistic applications related to the Cassini spacecraft trajectory design problem of the European Space Agency (ESA). The two problems have 6 and 22 real variables, respectively, and an unknown number of

[1] `http://aws.amazon.com/`

local optima. These problems have been studied extensively and are known to contain an enormous number of local optima and to be strongly deceptive for local optimizers [1].

Notation	Linear combination
DE/best/2/exp	$T = Best + Fact \cdot (A - B)$
DE/rand/2/exp	$T = E + Fact \cdot (A + B - C - D)$
DE/randToBest/2/exp	$T = E + Fact \cdot (Best - A + C - D)$

Table 4 The set $\mathcal{O}$ of operators used in our experiments.

The set $\mathcal{O}$ of DE operators we use is shown in Table 4. We adopt the standard notation presented in Table 2, keeping the proper semantics. We recall that the operators with *Best* use the global best solution in the network (as learned through gossip).

We assume that each Solver maintains a local population of 8 individuals. In all our experiments we vary the following parameters:

- **network size** (N) the number of nodes (solvers) in the network;
- **individual gossip rate** (G) the probability to send, after every function evaluation, the sub-optimal individual in the local population that has been updated more recently.

A total number of 2^{20} function evaluations are performed during each experiment, equally divided among the peers.

To test the effectiveness of the implementation in case of faulty network conditions, we considered the following scenarios:

- No message loss.
- SolverBox messages are lost with a probability of 0.25.

We run 5 independent experiments for all parameter combinations, using

$$N \in \{2^7, 2^8, 2^9, 2^{10}\} \text{ and } G \in \{0, 0.5, 1\}$$

combined with all the operators in Table 4, on all test functions, in both the described scenarios. For each experiment, we trace the best solution known by all nodes at each new iteration.

As we explained in Section 5, the communication cost of an optimization task can be modeled by taking into account the dimensionality of the specific Function being optimized and the parameters of the used Solver. In our experiments, $p_{Solver} = 1$ in all settings. Thus a message is sent after every function evaluation by each peer. With a probability that depends on the value of G, though, the message may contain a different amount of data. Table 5 summarizes the expected bandwidth use of a peer at any function evaluation (whose duration has been extimated based on the overall observed walltime) during our experiments (discarding the eventual imposed loss rate). The mandatory data length depends mainly on the dimensionality of the Function, whereas the length of the optional data depends on the current implementation of DE, which has not been optimized to minimize the communication costs, and clearly on the value of G.

Function	Time taken for 1 evaluation	Mandatory data	Optional data
Test functions	123 ms	112	$470 + (G * 280)$
Cassini1	300 ms	80	$374 + (G * 184)$
Cassini2	230 ms	208	$758 + (G * 568)$

Table 5 The expected length of a message sent by a peer at any Solver iteration during our experiments is the sum of Mandatory data and Optional data and depends on the actual value of G. By considering the average time taken for a single iteration, it is possible to estimate the bandwidth use during the computation.

6.2 Notes on the framework usability

Adapting the legacy code to the P2POEM framework has been a task characterized by a very limited intervention on the legacy code. A basic refactoring of method signatures (to match the new interfaces) has been enough for the DE algorithms to run on the PEERSIM simulator. The overall change is in the order of few tens of lines of code and, what is more important, the resulting source code can run on both the simulator and the CLOUDWARE platform.

This shows that P2POEM can be a useful facility for optimization practitioners who are not familiar with P2P distributed systems. Simple getters and setters for relevant variables, plus the change of methods signatures (to meet the framework interfaces) are likely to be the only required changes to some legacy (Java) code for it to be successfully plugged in the framework. In case of native code written in a language other than Java, the Java Native Interface (JNI) may be used and even simpler implementations of the framework interfaces would serve as glue code (this actually happens in the case of the two ESA functions we evaluate, that are implemented using C language, linked to the JVM environment via JNI, and to the framework by less than 60 lines of Java code).

What is perhaps more interesting is that, thanks to P2POEM abstractions facilities, the serialization of the data and the handling of the messages among solvers is completely independent from the optimization code itself. This fact allows the user to focus on the operative decisions about which data is to be shared and when, leaving to the framework the whole chore of distributing it efficiently.

As a further note, a thorough analysis of the logs has shown that, in a real deployment, the actual workload of the machines affects the pace of the communication between the peers. A non-uniform allocation of CPU for different peers may result in a bursty communication behavior, visible at the receivers, whose inbox queue presents a number of messages that ranges between 0 and 6 at the beginning of each iteration, with an average of 2.4. We think that the extent of our tests does not allow a general consideration about these figures and a general evaluation about their impact. It is worth noticing, as it becomes clear from the results we present in the following, that this fact has been quite harmless within our experiments. Moreover we think that this situation mimic well that of a volunteer computing platforms (not dissimilarly from [15]), which our framework is clearly targeted to.

6.3 Commented results

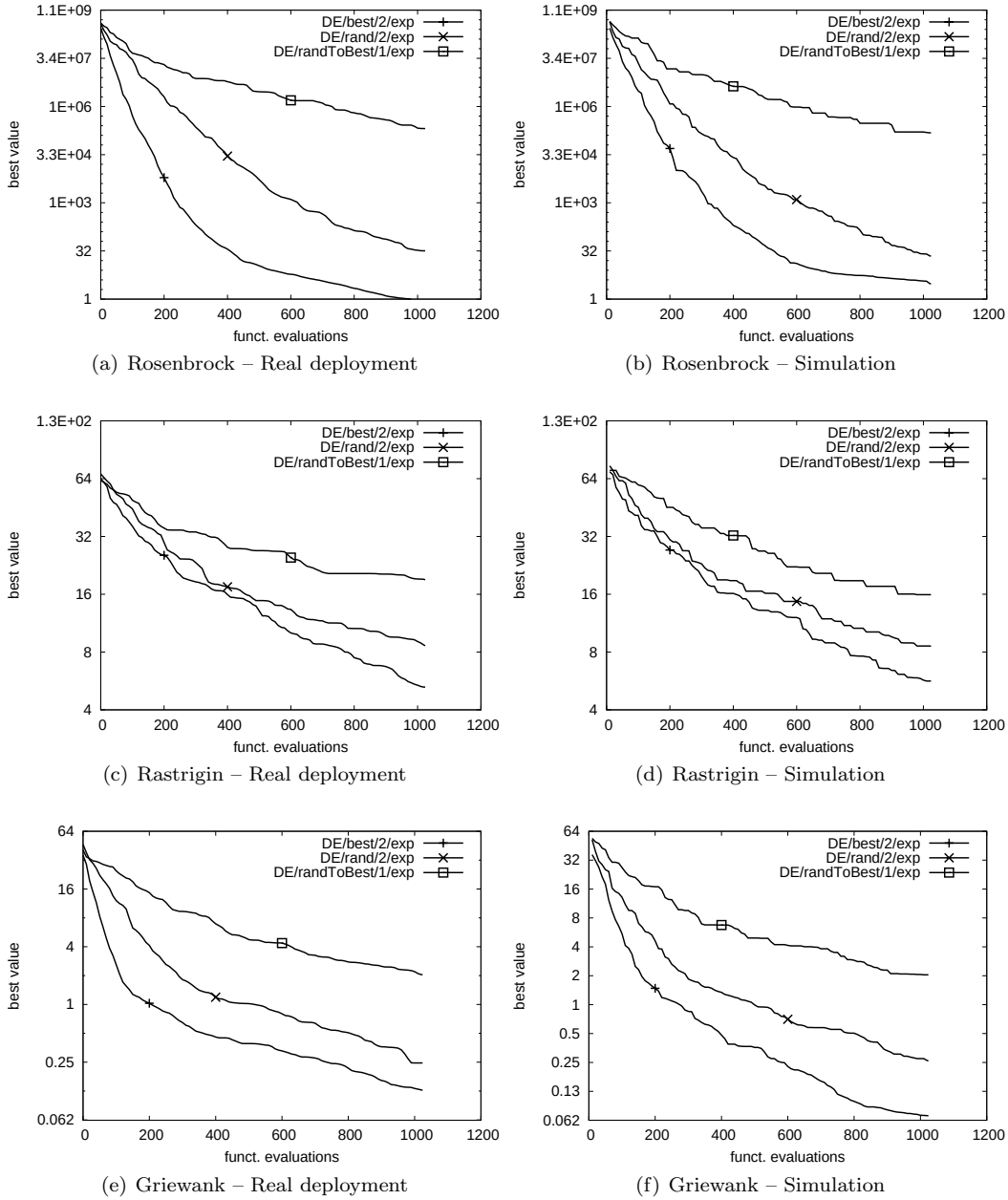
Some general remarks about the plots we present. On the vertical axis the best solution averaged over 5 experiments is plotted for each DE iteration (the lower the value, the better the quality). On the horizontal axis there is the number of performed function evaluations. This is a common way of counting the elapsed time while analyzing the dynamics of an optimization task. Wall clock time is heavily dependent on the function to optimize and on the algorithm to run, so it is less informative for our purposes. The plots we present in this section are truly representing the whole set of experiments, whose complete set of more than 300 pictures we cannot obviously include.

Figure 2 shows a comparison between simulations and real deployment. We performed a first set of tests with the same basic setting in the two environments (1024 peers, $G = 0$, no message loss), to verify the behavior of the algorithms. P2POEM has proven to be a reliable tool to plug in and distribute optimization algorithms without altering their performance. The results obtained in a simulated environment are clearly comparable to those obtained on a simulator under the same settings. By examining the outcomes of the datasets in detail, we actually see that the results obtained in the real deployment tend to be slightly better than the simulative ones, considering the global behavior of the operators targeting the local optima during the computation. If this is only a stochastic effect or it is related to the different pace of the peers communication is currently under investigation.

In fact, concerning the real deployment case, we see that no harm comes from the multi-threaded nature of the application and the outcomes are macroscopically undistinguishable from the simulation ones. On the one hand, this is not unexpected, because gossip protocols are known to be able to cope well with communication delays that are far worse than what we possibly have in these experiments. Nonetheless, a closer look at the raw output data confirms that even when a large number of threads are concurrently running on the same machines, messages are sent, received and processed within very few protocol iterations, thus keeping the overall information flow smooth enough not to worsen the overall performance of the optimization task.

Even if the performance of the optimization task by itself is not the major concern in the present experimentation, to provide a set of tool that may improve distributed optimization problems is definitely the long term goal which we work for. Thus it is also relevant to analyze the outcomes with respect to the behavior of the distributed DE algorithm we adopt. The following results have been obtained using P2POEM on CLOUDWARE.

As it is visible in Figure 3, our outcomes confirm that the gossip communication mechanism is able to cope with a faulty environment. Even in case of a fairly high amount of message losses (plots on the right), the optimization achieves good results and the effect of the parameter G is clearly visible. The operators evidently improve their performance, as soon as more useful information becomes available through gossiping. These results are comparable to those presented in our mentioned work [8], which have been obtained in a simulated environment. By examining the datasets of the two works, we see that the same operators have similar behavior — taking the corresponding network size — for the same functions, in that they find local optima whose values are in the same order of magnitude after a similar amount of function evaluations. It is anyway to be noted



(a) Rosenbrock – Real deployment

(b) Rosenbrock – Simulation

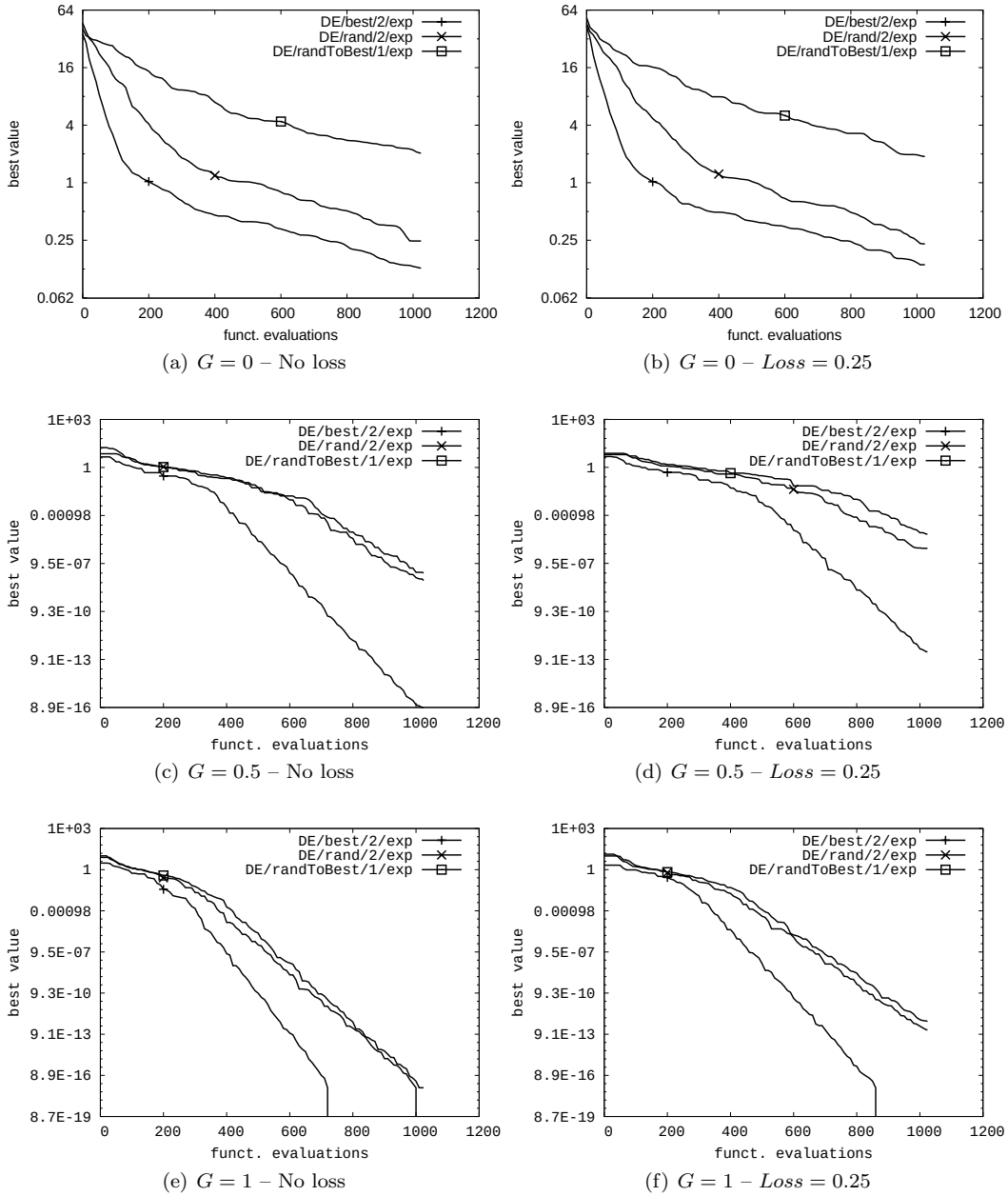
(c) Rastrigin – Real deployment

(d) Rastrigin – Simulation

(e) Griewank – Real deployment

(f) Griewank – Simulation

Fig. 2 Comparison between simulations and real deployment.



(a) $G = 0$ – No loss

(b) $G = 0$ – $Loss = 0.25$

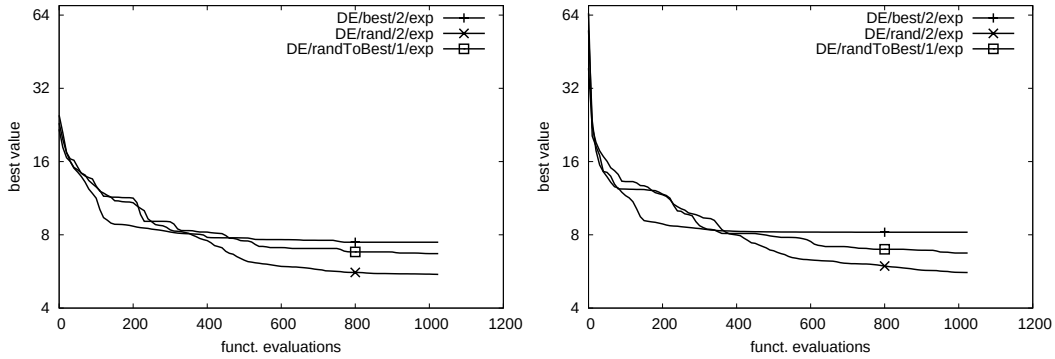
(c) $G = 0.5$ – No loss

(d) $G = 0.5$ – $Loss = 0.25$

(e) $G = 1$ – No loss

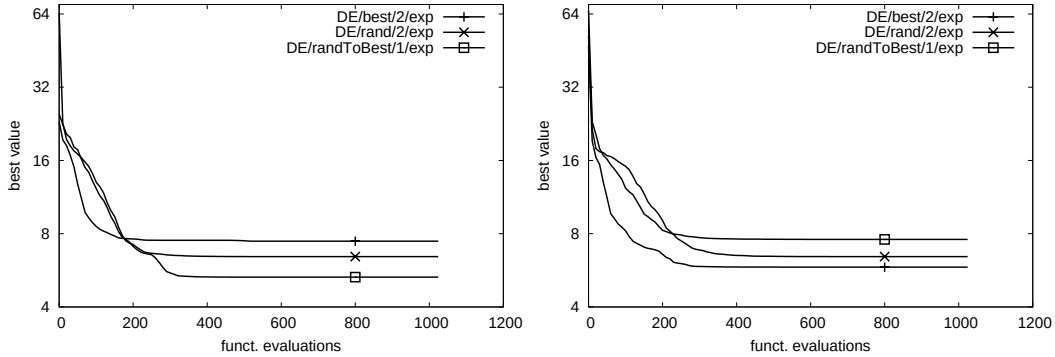
(f) $G = 1$ – $Loss = 0.25$

Fig. 3 Griewank – How the gossip rate G improves performance in different networks (1024 solvers).



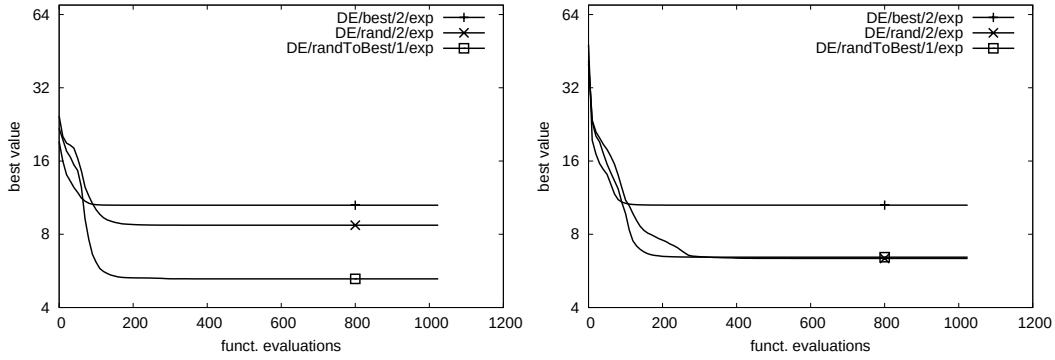
(a) $G = 0$ – No loss

(b) $G = 0$ – $Loss = 0.25$



(c) $G = 0.5$ – No loss

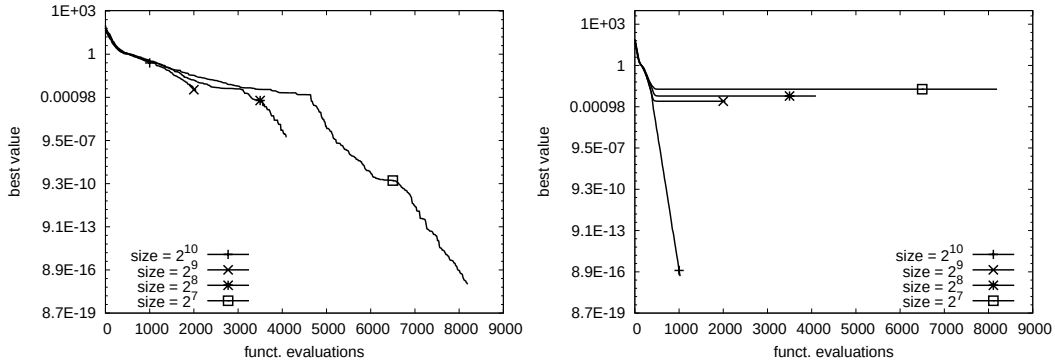
(d) $G = 0.5$ – $Loss = 0.25$



(e) $G = 1$ – No loss

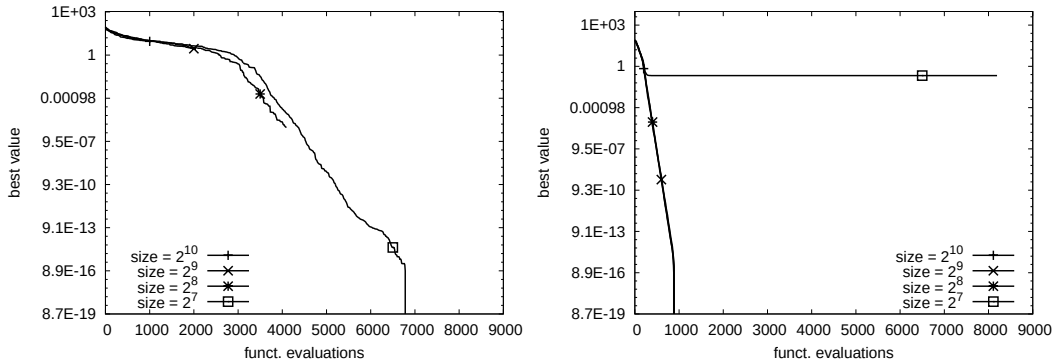
(f) $G = 1$ – $Loss = 0.25$

Fig. 4 Cassini1 – How the gossip rate G improves performance in different networks (1024 solvers).



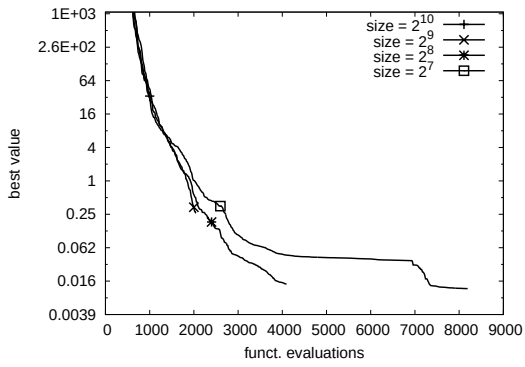
(a) Griewank – $G = 0$

(b) Griewank – $G = 1$

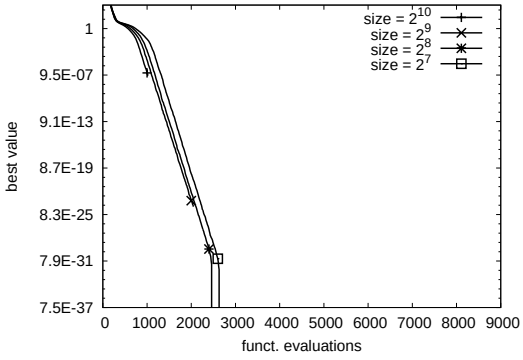


(c) Rastrigin – $G = 0$

(d) Rastrigin – $G = 1$



(e) Rosenbrock – $G = 0$



(f) Rosenbrock – $G = 1$

Fig. 5 DE/Rand/2/Exp – Results for different network sizes.

that the faulty network scenarios are different (here message losses, there a uniform controlled churn) and affect the computation in a quite different way. In particular, message losses imply no "restarting side-effect" for the solver — as it happens in case of churn —, thus being a pure obstacle to the optimization task. This detrimental effect may merely slow down the effectiveness of the operator (see DE/Best/2/exp in subfigures (e) and (f)) or worsen its performance (as it is the case for the other operators in the same subfigures), depending on how much the operator itself rely on the information that is distributed. As we see in subfigure (f), the adverse network condition does not prevent an operator to reach the minimum value of this function (that is where the corresponding line goes vertically down), whereas with no message loss the same outcome is reached by two operators and slightly earlier (see subfigure (e)).

An overview of the results concerning the Cassini functions is presented in Figure 4. Here the effect of the information sharing mechanism is still visible, as a comparison between the first group — subfigures (a) and (b) — and the second group — subfigures (c) and (d) — immediately shows, but it has no effect on the final outcome. It does rather affect the convergence speed to the main attraction basin of the minimum. This indicates that the true minimum is quite likely to be close and improving the quality of the solution gets harder and harder as the computation goes on. While the results we plot here nicely show the dynamics of the optimization task, they actually present no remarkable achievement.

But we anyway verified the effectiveness of the distributed optimization design, which our framework is able to contribute to, by running a further dedicated set of experiments. By extending the overall number of function evaluation to 2^{22} and the network size to 2048 peers, we have been able to find a new global optimum for the Cassini1 function, with value: 4.930708, at the point:

-789.7561990498968, 158.30408685063892, 449.38588187459754,
4.71084087421275, 1024.750694389757, 4552.8956725497155

that is equivalent to the current best known value for this function, as published by ESA (see Table 3). Notably, the same experimental setting have given a value of 8.616000 for Cassini2, which is within the seven best values ever found for this function. We think these are remarkable results, because neither the optimization algorithm nor the distributed optimization design have been "tuned" against these specific problems.

In Figure 5 we finally present a comparison among results obtained for different network sizes, while using the same DE operator (DE/Rand/2/Exp) on various functions. A detailed analysis of this pictures enlightens several interesting aspects of the distributed computation. We summarize here the main noteworthy points.

The first thing that must be noticed is the fact that the overall number of function evaluation is equal to 2^{20} in all the experiments. Thus, depending on the total number of nodes in the network (N), we allocate a different number of evaluation per node ($2^{20}/N$). This means that the experiments involving less nodes take more time than those involving a greater number of nodes. We are therefore able to see if and when the distribution of the optimization task over a large number of machine is really effective, besides obviously being time saving.

By comparing the subfigures (a) and (b) we clearly see how spreading the computation pays off only if the information diffusion is maximized. While if $G = 0$

the quality of the solution heavily depends on the number of evaluations that are performed on each node, once the gossiping mechanism is used to spread useful information we obtain a patent improvement for the largest network. But the relation among quality of results, information sharing and network size is not trivial. In particular, we see here that a too aggressive rate of information diffusion leads to an early convergence to sub-optimal results, worsening the performance for the smaller networks. This phenomenon is well known in the field and indicates that the parameters of the optimization must be tuned accordingly to the specific "coupling" operator-function. In this specific case, a slower pace in spreading the information ($G = 0.5$ or less) gives much better results (not shown here) when we deploy 2^8 and 2^9 nodes.

Subfigures (c) and (d) show in fact how the distributed optimization task may greatly benefit from the deployment of large numbers of solvers. The minimum of this function is reached in almost all the cases and with a huge speedup when $G = 1$, with respect to the case in which $G = 0$ and only the number of evaluations per node matters. Again, 2^7 solvers cannot cope with a fast rate of information dissemination.

The most interesting outcomes, from the point of view of the function optimization task itself, are perhaps those shown in subfigures (e) and (f). For this function, not only the improvement is relevant for all the networks, but it is also incomparably greater than any results that could be achieved without enabling the population diffusion mechanism (notice the big difference of scale on the vertical axis between the two subfigures). From the point of view of our distributed P2P design, by using larger networks of solvers we obtain for this coupling a great improvement both in the quality of the solution and in the time taken for its computation.

These results, obtained in a real deployment, confirm and extend similar outcomes known in the literature and obtained in simulations, thus assessing the viability of the P2P epidemic approach to function optimization problems. Together with the new remarkable results concerning the real world Cassini functions, they also show the versatility and the usefulness of our framework.

7 Related work

The framework we propose operates as a "trait d'union" between P2P overlay systems and function optimization algorithms. We present in Section 7.2 some recent proposals of optimization frameworks with respect to which P2POEM begs to differ. But first we recall some interesting example of P2P optimization algorithms that have recently come to attention.

7.1 P2P optimization algorithms

P2P heuristic optimization is quite a newborn branch of distributed optimization. As it happens in all the happy beginnings, researchers are extensively exploring the various issues this new field entails. We give here a brief overview of the recent publications, mostly related to P2P implementations of population-based algorithms.

In [29], the authors presented some preliminary evaluations of a parallel hybrid MO-EA (multi-objective evolutionary algorithm) deployed in a P2P environment. Several results show that is possible to successfully parallelize such an evolutionary algorithm in a P2P fashion, exploiting the resources available in a network of 120 heterogeneous PCs.

In [18], an extensive experimentation was performed, to test the fault tolerance of the island model on GA, when executing them on a distributed system. The results show that this model can be trusted when running experiments on a non-reliable parallel or distributed infrastructure, the quality of the outcomes being at most 2% worse (on average) than when a reliable infrastructure is employed, without adding any special techniques required for dealing with faults.

Recently, the same authors [34] proposed new models to substitute failing particles of the same kind of multi-objective PSO algorithm, in such a way that the capability of the affected swarm to explore the search space can be enhanced. The initialization of the new particles is performed using a combination of binary search to fill the gaps in the space between the two known Pareto-front extremes and edge extension, to improve the exploration beyond the known Pareto-front.

Van Steen et al. [37] presented a fully decentralized evolutionary algorithm in which the population size is kept stable by locally adapting the surviving rate of the individuals according to global population estimations, performed by means of gossiping protocols. The parent and survivor selection can be done completely autonomously and asynchronously, without central control, yet avoiding the risk of population explosion or implosion.

In [27] the authors used the same algorithm to tackle hard problems whose computational time increase exponentially as they scale. Moreover, they compared their results with a sequential GA algorithm, outperforming this competitor thanks to the gossiping adaptive mechanism used to control the population.

Laredo et al. [24] proposed the Gossiping-based Evolvable Agent model (actually already devised in [26]), where every individual of an evolutionary algorithm's population self-schedules its own action as an agent (evolvable agent) and dynamically self-organizes its neighborhood via newscast (gossip-based). Tests run in a really distributed deployment with multi-threaded configurations confirm that P2P evolutionary algorithms are competitive with respect to not-distributed ones.

Finally, in [25] the authors presented an extensive evaluation of a generic P2P evolutionary algorithm with respect of different (simulated) network size and churning conditions, while tackling a known hard test function. Results show that the gossiping enabled small-world structure of the network makes the algorithm very robust even when very high churn rates are applied.

As a whole, existing literature about P2P heuristic optimization confirms that not only this approach to distributed optimization is viable, but moreover that can suit hard problems, provided that an adequate number of resources are available. The availability of many cheap and interconnected machines is an easy goal to obtain in many contexts nowadays and this new kind of algorithms show the capability to fruitfully exploit such a computational environment.

7.2 Function Optimization Frameworks

As said, the main goals motivating P2POEM's design concern the easy deployment of novel or legacy optimization algorithms in a P2P decentralized network of solvers. The framework we propose addresses the issues of the interchangeability of the optimization components and focuses on the specificity of a P2P environment. Then we also aim at providing a tool that requires little or no intervention on the algorithm's code when moving from a simulated to a real distributed network. Several other optimization frameworks exist, whose goals and targets are slightly different. We briefly present the most prominent in the following. We thus clarify what makes P2POEM original, by describing their main characteristics.

Very few frameworks are natively suited for distributed environments. One of them is ParadisEO [13]. It is an object-oriented framework dedicated to the reusable design of parallel and distributed meta-heuristics. It embeds the implementations of widely used evolutionary algorithms and local searches algorithms, along with the most common parallel and distributed models and hybridization mechanisms. Their implementation is portable on distributed-memory machines as well as on shared-memory multiprocessors. The provided software architecture allows a good level of code re-usability, achieved by means of a quite complex hierarchy of classes and templates the user has to master. Although not P2P-oriented, this framework covers a large range of optimization techniques, enabling their active cooperation in a distributed fashion. A similar approach, but limited to evolutionary algorithms and to their master/slave coordinated execution has guided the design of DistributedBeagle [16].

mallba [2] is a software tool for the resolution of combinatorial optimization problems using generic algorithmic skeletons implemented in C++. Skeleton classes implement the generic core functionalities of various optimization methods and devise three different implementations for any of them: sequential, parallel for Local Area Networks and parallel for Wide Area Networks. The user must fill in the required classes with a specific problem-dependent implementation, whereas the optimization and parallelization technique is completely provided by the library. Communications handling is done via a light middleware layer that simplifies their tuning and provides easy access to message passing, broadcasting and coordination facilities. While the design principles of this framework are quite similar to P2POEM's, the complexity of the overall architecture makes the adoption of legacy code a not trivial chore.

A remarkable framework that addresses both the issues of assisting the user in the algorithm design and helping the distributed execution is dream [3]. This P2P system is based on the island model, implemented through epidemic protocols. dream is targeted toward Wide Area Networks (WAN), where communications rely on Internet standard protocols. Moreover, various kind of user interfaces are provided to meet the needs of users with different skills, interests or commitment. The range of issues and the target that this framework aims to address are larger and slightly different from ours. dream is devoted to evolutionary algorithms, while we propose a framework which can suit different kind of optimization algorithm too. Of course this means that our external interfaces are much less specialized, because they *must not* demand the user's algorithms to comply with very specific constraints or design patterns. Then our tool is primarily meant to be used by those scientists who want to easily and quickly deploy and test their algorithms

in a P2P distributed fashion. DREAM has been devised as a platform that can run virtually any agent-based distributed application and offers several different user interfaces, to suit the most different kinds of users.

An interesting approach to address the issue of solvers P2P distribution has been recently proposed in [5]. G2DGA is a framework implemented as a hybrid P2P overlay with two types of objects, (i) islands that run a GA process, and (ii) a supervisor that perform monitoring and adaptation. The supervisor creates the island objects and defines a migration policy that is sent to each of them, specifying the migration interval, rate, and a list of neighboring islands. The islands run the GA and handle migration, which is asynchronous. The migrants are sent directly between the peers (islands), while the supervisor collects feedback data from the islands and is responsible for adaptation. G2DGA is based on G2P2P, which is a P2P distributed object framework based on .NET remoting and XML encoded message passing communications.

8 Conclusion

We presented a novel framework that aims at easing the burden of performing function optimization tasks in a decentralized P2P network of solvers. Distributed optimization has already a long and rich history, but little has been done to make it exploit the (potentially) large computing facilities a reliable P2P network can provide. P2POEM bridges the gap between the P2P world and the needs of optimization practitioners, who are often not able to find a reasonably simple way to run their algorithms in such a distributed environment.

We described the architecture of our framework, based on the separation between a *topology service* that maintains the network topology, an *optimization service* that performs the function optimization task and a *communication service* that enables the information sharing mechanism among the optimizers that run in different nodes.

We presented the main components of the framework: the SolverBox, which is the main local coordinator running in each node; the *entity interfaces*, that make it possible to plug novel and legacy optimization algorithms in the P2P environment, providing a full interaction with the networking services and the communication protocol. We explained in detail how the communication protocol works and how a sophisticated peer-entity communication is provided among the nodes, making them cooperate according to the user decisions.

Finally we provided experimental results about how we used P2POEM to perform some experiments using Differential Evolution, a well-known optimization technique. Thanks to the versatility of our framework, we could adopt legacy code and use it to perform simple optimization tasks in a real deployment with a very low refactoring overhead, (adaptation of functions signatures and implementation of very basic getter and setter methods for variables, as required to implement P2POEM's interfaces). Results showed the viability of the gossip optimization approach and the reliability of our framework as a middleware tool.

The outcomes of our experiments show that our framework is able to support the very same optimization code both in a simulated environment and in a real deployment, while handling in a transparent way all the communication issues, leaving to the user the only chore of the optimization task. The results obtained

in a real network of more than a thousand nodes are comparable to those existing in literature, that have been obtained in a simulated environment. Moreover a state-of-the-art matching result has been achieved while evaluating a "real world" function, related to a well known space crafting problem widely studied by the European Space Agency.

Further tests are required to evaluate P2POEM's dependability in a real deployment. Different optimization algorithms have to be evaluated and different problems have to be challenged too. We are working to see whether the positive impact of the information sharing mechanism we devised can be extended to other population-based optimization techniques. This might lead to the profitable cooperation of different kind of optimization techniques on a common problem. Ongoing research is devoted to the refinement of our distributed DE algorithm, that may achieved a much better performance by sharing different information with a more sophisticated mechanism. Various possibilities are easily accessible with little intervention on the algorithms, thanks to P2POEM versatility.

Distributed function optimization on decentralized P2P networks is just at the beginning of a promising story. We think our tool will contribute to prove it is a challenge worth taking.

References

1. B. Addis, A. Cassioli, M. Locatelli, and F. Schoen. Global optimization for the design of space trajectories, 2008. Optimization Online eprint archive `http://www.optimization-online.org/DB_HTML/2008/11/2150.html`.
2. E. Alba, G. Luque, J. Garcia Nieto, G. Ordonez, and G. Leguizamon. MALLBA: a software library to design efficient optimisation algorithms. *IJICA*, 1(1):74–85, 2007.
3. M. G. Arenas, P. Collet, A. E. Eiben, M. Jelasity, J. J. Merelo, B. Paechter, M. Preuß, and M. Schoenauer. A framework for distributed evolutionary algorithms. In *Parallel Problem Solving from Nature - PPSN VII*, volume 2439 of *LNCS*, pages 665–675. Springer, 2002.
4. R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*. Springer Publishing Company, Incorporated, 2008.
5. J. Berntsson. G2DGA: an adaptive framework for internet-based distributed genetic algorithms. In *Proc. of GECCO '05*, pages 346–349, New York, NY, USA, 2005. ACM.
6. M. Biazzini, B. Bánhelyi, A. Montresor, and M. Jelasity. Distributed hyper-heuristics for real parameter optimization. In *Proc. of GECCO'09*, pages 1339–1346, Montreal, Québec, Canada, July 2009.
7. M. Biazzini, B. Bánhelyi, A. Montresor, and M. Jelasity. Peer-to-peer optimization in large unreliable networks with branch-and-bound and particle swarms. In *Applications of Evolutionary Computing*, pages 87–92. Springer, 2009.
8. M. Biazzini and A. Montresor. Gossiping differential evolution: a decentralized heuristic for function optimization in p2p networks. In *Proceedings of the 16th International Conference on Parallel and Distributed Systems (ICPADS'10)*, Dec. 2010.
9. M. Biazzini and A. Montresor. P2POEM, 2011. `http://p2poem.sf.net`.
10. M. Biazzini, A. Montresor, and M. Brunato. Towards a decentralized architecture for optimization. In *Proc. of IPDPS'08*, Miami, FL, Apr. 2008.
11. M. Brunato, R. Battiti, and S. Pasupuleti. A memory-based rash optimizer. In *Proc. of AAAI-06 Workshop on Heuristic Search, Memory Based Heuristics and their applications*, Boston, MA, 2006.
12. E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of Metaheuristics*, pages 457–474. Springer, 2003.
13. S. Cahon, N. Melab, and E.-G. Talbi. Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.
14. A. Demers et al. Epidemic algorithms for replicated database maintenance. In *Proc. of the 6th ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 1–12. ACM Press, Aug. 1987.

15. T. Desell, M. Magdon-Ismail, B. Szymanski, C. Varela, H. Newberg, and D. Anderson. Validating evolutionary algorithms on volunteer computing grids. In F. Eliassen and R. Kapitza, editors, *Distributed Applications and Interoperable Systems*, volume 6115 of *Lecture Notes in Computer Science*, pages 29–41. Springer Berlin / Heidelberg, 2010.
16. C. Gagne, M. Parizeau, and M. Dubreuil. Distributed Beagle: An environment for parallel and distributed evolutionary computations. In *Proc. of the 17th Int. Symposium on High Performance Computing Systems and Applications*, Sherbrooke, Québec, Canada, Apr. 2003. NRC Research Press.
17. C. P. Gomes and B. Selman. Algorithm portfolios. *Artif. Intell.*, 126(1-2):43–62, 2001.
18. J. I. Hidalgo, J. Lanchares, F. Fernández de Vega, and D. Lombra na. Is the island model fault tolerant? In *GECCO '07*, pages 2737–2744, New York, NY, USA, 2007. ACM.
19. M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, August 2005.
20. M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3):8, 2007.
21. J. Jimnez Laredo, D. Lombraa Gonzlez, F. Fernndez de Vega, M. Garca Arenas, and J. Merelo Guervs. A peer-to-peer approach to genetic programming. In S. Silva, J. Foster, M. Nicolau, P. Machado, and M. Giacobini, editors, *Genetic Programming*, volume 6621 of *Lecture Notes in Computer Science*, pages 108–117. Springer Berlin / Heidelberg, 2011.
22. J. Kennedy and R. C. Eberhart. Particle swarm optimization. *IEEE Int. Conf. Neural Networks*, pages 1942–1948, 1995.
23. C. Kesselman and I. Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
24. J. Laredo, P. Castillo, A. Mora, and J. Merelo. Exploring population structures for locally concurrent and massively parallel evolutionary algorithms. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 2605–2612, June 2008.
25. J. Laredo, P. Castillo, A. Mora, J. Merelo, and C. Fernandes. Resilience to churn of a peer-to-peer evolutionary algorithm. IJHPSA, 2008. Volume 1, Number 4.
26. J. L. J. Laredo, A. E. Eiben, M. Schoenauer, P. A. Castillo, A. M. Mora, F. F. de Vega, and J. J. M. Guervós. Self-adaptive gossip policies for distributed population-based algorithms. *CoRR*, abs/cs/0703117, 2007.
27. J. L. J. Laredo, E. A. Eiben, M. van Steen, P. A. Castillo, A. M. Mora, and J. J. Merelo. P2P evolutionary algorithms: A suitable approach for tackling large instances in hard optimization problems. In *Proc. of Euro-Par*, volume 5168 of *LNCS*, pages 622–631. Springer-Verlag, 2008.
28. O. Maron and A. W. Moore. The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11:193–225, 1997.
29. N. Melab, M. Mezmaz, and E.-G. Talbi. Parallel hybrid multi-objective island model in peer-to-peer environment. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 6*, page 190.2, Washington, DC, USA, 2005. IEEE Computer Society.
30. A. Montresor. Cloudware, 2011. `http://cloudware.sf.net`.
31. A. Montresor and M. Jelasity. Peersim: A scalable p2p simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pages 99–100, Seattle, WA, Sept. 2009.
32. B. Pittel. On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, February 1987.
33. I. Scriven, A. Lewis, D. Ireland, , and J. Lu. Distributed multiple objective particle swarm optimisation using peer to peer networks. In *IEEE Congress on Evolutionary Computation (CEC)*, 2008.
34. I. Scriven, A. Lewis, and S. Mostaghim. Dynamic search initialisation strategies for multi-objective optimisation in peer-to-peer networks. *IEEE Congress on Evolutionary Computation, CEC '09*, pages 1515 – 1522, 2009.
35. R. Storn and K. Price. Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, Dec. 1997.
36. E. G. Talbi. *Parallel Combinatorial Optimization*. John Wiley, 2006.
37. W. R. M. U. K. Wickramasinghe, M. van Steen, and A. E. Eiben. Peer-to-peer evolutionary algorithms with adaptive autonomous selection. In *Proc. of GECCO'07*, pages 1460–1467. ACM Press New York, NY, USA, 2007.