Peer-to-peer Optimization in Large Unreliable Networks with Branch-and-Bound and Particle Swarms*

Balázs Bánhelyi[1], Marco Biazzini[2], Alberto Montresor[2], and Márk Jelasity[3]

[1] University of Szeged, Hungary, banhelyi@inf.u-szeged.hu
[2] University of Trento, Italy, biazzini,montreso@dit.unitn.it
[3] University of Szeged and HAS, Hungary, jelasity@inf.u-szeged.hu

Abstract. Recent developments in the area of peer-to-peer (P2P) computing have enabled a new generation of fully-distributed global optimization algorithms via providing self-organizing control and load balancing mechanisms in very large scale, unreliable networks. Such decentralized networks (lacking a GRID-style resource management and scheduling infrastructure) are an increasingly important platform to exploit. So far, little is known about the scaling and reliability of optimization algorithms in P2P environments. In this paper we present empirical results comparing two algorithms for real-valued search spaces in large-scale and unreliable networks. Some interesting, and perhaps counter-intuitive findings are presented: for example, failures in the network can in fact significantly *improve* performance under some conditions. The two algorithms that are compared are a known distributed particle swarm optimization (PSO) algorithm and a novel P2P branch-and-bound (B&B) algorithm based on interval arithmetic. Although our B&B algorithm is not a black-box heuristic, the PSO algorithm is competitive in certain cases, in particular, in larger networks. Comparing two rather different paradigms for solving the same problem gives a better characterization of the limits and possibilities of optimization in P2P networks.

1 Introduction

During the past decade various large scale networks have emerged as computing platforms such as the Internet, the web, in-house clusters of cheap computers, and, more recently, networks of mobile devices. The exploitation of these networks for computing and for other purposes such as file sharing and content distribution has followed a different path. Whereas computing is normally performed using GRID technologies, other applications, due to legal and efficiency reasons, favored fully decentralized self-organizing approaches, that became known as peer-to-peer (P2P) computing.

It is an emerging area of research to transport P2P algorithms back into the world of scientific computing, in particular, distributed global optimization. P2P algorithms can replace some of the centralized mechanisms of GRIDs that include monitoring and

* http://dx.doi.org/10.1007/978-3-642-01129-0_10 (Short version in *Applications of Evolutionary Computing*, Springer LNCS 5484). This work was supported by the European Space Agency through Ariadna Project "Gossip-based strategies in global optimization" (21257/07/NL/CB). M. Jelasity was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences. B. Bánhelyi was supported by the Ferenc Deák Scholarship No. DFÖ 19/2007, Aktion Österreich-Ungarn 70öu1, and OTKA T 048377.

control functions. For example, network nodes can distribute information via "gossiping" with each other and they can collectively compute aggregates of distributed data (average, variance, count, etc) to be used to guide the search process [1]. This in turn increases robustness and communication efficiency, allows for a more fine-grained control over the parallel optimization process, and makes it possible to utilize large-scale resources without a full GRID control layer and without reliable central servers.

The interacting effects of problem difficulty, network size, and failure patterns on optimization performance and scaling behavior are still poorly understood in P2P global optimization. In this paper we present empirical results comparing two P2P algorithms for real-valued search spaces in large-scale and unreliable networks: a distributed particle swarm optimization (PSO) algorithm [2] and a novel P2P branch-and-bound (B&B) algorithm based on interval arithmetic. Although our B&B algorithm is not a black-box heuristic, the PSO algorithm is competitive in certain cases, in particular, in larger networks. Some interesting, and perhaps counter-intuitive findings are presented as well: for example, failures in the network can in fact significantly *improve* the performance of P2P PSO under some conditions.

Related work. Related work can be classified as parallel optimization, P2P networking, and, very recently, the intersection of these two fields. We focus on this last category, mentioning that, for example, [3] is an excellent collection of parallelization techniques for various algorithms, and, for example, [4] is a useful reference for P2P computing in general.

In P2P heuristic optimization, proposed algorithms include [5, 6, 2]. They all build on gossip-based techniques [1, 7] to spread and process information, as well as to implement algorithmic components such as selection and population size control. Our focus is somewhat different from [5, 6] where it was assumed that population size is a fixed parameter that needs to be maintained in a distributed way. Instead, we assume that the network size is given, and should be exploited as fully as possible to optimize speed. In this context we are interested in understanding the effect of network size on performance, typical patterns of behavior, and related scaling issues.

In the case of B&B, we are not aware of any fully P2P implementations. The closest approach is [8], where some components, such as broadcasting the upper bound, are indeed fully distributed, however, some key centralized aspects of control remain, such as the branching step and the distribution of work. In unreliable environments we focus on, this poses a critical problem for robustness.

Contributions. Our contributions include (i) a completely decentralized self-organizing B&B algorithm, presented in Section 2, where no manager or master nodes are needed and (ii) a scalability analysis, presented in Section 3, with simulations of P2P networks of various sizes involving node churn, and with a comparison to a P2P PSO implementation, that is designed to operate under the same conditions.

2 The Algorithms

Our target networking environment consists of independent nodes that are connected via an error-free message passing service: each node can pass a message to any target node, provided the address of the target node is known. We assume that node failures are possible. Nodes can leave and new nodes can join the network at any time as well.

In the following we describe two algorithms that operate in such networking environments. Both assume that all nodes in the network run an identical algorithm; thus no special role exists such as a master or slave. This design choice increases both robustness to failure and scalability.

None of the algorithms contain special methods to deal with leaving or joining nodes. New nodes simply start participating after a default initialization procedure, and failing nodes are tolerated automatically via the inherent (or explicit) redundancy of the algorithm design, as we explain later.

The above self-organizing features are made possible via a randomized communication substrate both algorithms are based on. This is remarkable especially because the two algorithms are rather different, yet they are based on similar basic P2P algorithms and services. In this section we first briefly overview this communication substrate, and then we discuss the two algorithms. Only our novel B&B algorithm is discussed in full detail, as the other ideas are taken from previous publications.

2.1 Peer Sampling and its Applications

We assume that all nodes are able to send a message to a random node from the network at any time. This very simple communication primitive is called the *peer sampling service* that has a wide range of applications [9]. In this paper we will use this service as an abstraction, without referring to its implementation; lightweight, robust, fully distributed implementations exist based on the analogy of *gossip* [9]. We note that one of the earliest approaches to use a similar service, called *newscast*, was the DREAM framework [10].

The algorithms below will rely on two particular applications of the peer sampling service. The first is gossip-based broadcasting, and the second is diffusion-inspired load balancing. In gossip-based broadcasting, nodes periodically communicate pieces of information they consider "interesting" to random other nodes. This way, information spreads exponentially fast. Several techniques exist to increase the efficiency and performance of the method [11].

In diffusion-based load balancing, nodes periodically test random other nodes to see whether those have more load or less load, and then perform a balancing step accordingly. This process models the diffusion of the load over a random network.

Although we do not discuss the implementation of these functions, it has to be noted that their communication cost is moderate: gossiping involves periodically sending small messages to random peers. The period of communication can be configured. In our case this period will be in the order of a few function evaluations. For difficult realistic problems this results in almost negligible communication costs.

2.2 P2P PSO

Based on gossip-based broadcasting, a distributed implementation of a PSO algorithm was proposed [2]. Here we will use a special case of this algorithm, where particles are mapped to nodes: one particle per node. The current best solution, a key guiding information in PSO, is spread using gossip-based broadcast. In a nutshell, this means we have a standard PSO algorithm where the number of particles equals the network size and where the neighborhood structure is a dynamically changing random network. For more details, the reader is kindly referred to [2].

Algorithm 1 P2P B&B

```
1: loop                                                              ▷ main loop
2:     I ← priorityQ.getFirst()          ▷ most promising interval; if queue empty, blocks
3:     (I_1, I_2) ← branch(I)                          ▷ cut the interval in two along longest side
4:     min_1 ← upperBound(I_1)                  ▷ minimum of 8 random samples from interval
5:     min_2 ← upperBound(I_2)
6:     min ← min(min,min_1,min_2)                              ▷ current best value known locally
7:     b_1 ← lowerBound(I_1)                  ▷ calculates bound using interval arithmetic
8:     b_2 ← lowerBound(I_2)
9:     priorityQ.add(I_1, b_1)                          ▷ queue is ordered based on lower bound
10:    priorityQ.add(I_2, b_2)
11:    priorityQ.prune(min)          ▷ remove entries with a higher lower bound than min
12:    p ← getRandomPeer()                                        ▷ calls the peer sampling service
13:    sendMin(p, min)                                            ▷ gossips current minimum
14:    if p has empty queue or local second best interval is better than p's best then
15:        sendInterval(p, priorityQ.removeSecond())          ▷ gossip-based load balancing step
16:    end if
17: end loop
18: procedure ONRECEIVEINTERVAL(I(⊆ D), b)
19:    priorityQ.add(I, b)          ▷ D ⊆ ℝ^d is the search space, b is lower bound of I
20: end procedure
21: procedure ONRECEIVEMIN(min_p)
22:    min ← min(min_p,min)
23: end procedure
```

2.3 P2P B&B

Various parallel implementations of the B&B paradigm are well-known [3]. Our approach is closest to the work presented in [12] where the bounding technique is based on interval-arithmetic [13]. The important differences stem from the fact that our approach is targeted at the P2P network model described above, and it is based on gossip instead of shared memory.

The basic idea is that, instead of storing it in shared memory, the lowest known upper bound of the global minimum is broadcast using gossip. In addition, the intervals to be processed are distributed over the network using gossip-based load balancing.

The algorithm that is run at all nodes is shown in Algorithm 1. Each node maintains a priority queue and a current best minimum value. The priority queue contains intervals ordered according to their lower bound, where the most promising interval has the lowest lower bound.

The lower bound for an interval is calculated using interval arithmetic, which guarantees that the calculated bound is indeed a lower bound. This way, in the lack of failures in the network, the algorithm is guaranteed to eventually find the global minimum. However, we continuously have a current best value as well, so the algorithm can be terminated at any time. Any function with a precise mathematical definition supports interval arithmetic, although in some cases at a relatively large cost. Detailed discussion of the details of interval arithmetic is out of the scope of this paper, please refer to [13].

We start the algorithm by sending the search domain D with lower bound $b = \infty$ to a random node. In faulty environments we can send this initial interval to more than one node. Termination is not discussed here: since it is an any-time algorithm, as mentioned

	Function $f(x)$	D	$f(x^*)$	K
Sphere2	$x_1^2 + x_2^2$	$[-5.12, 5.12]^2$	0	1
Sphere10	$\sum_{i=1}^{10} x_i^2$	$[-5.12, 5.12]^{10}$	0	1
Griewank10	$\sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]^{10}$	0	$\approx 10^{19}$
Schaffer10	$0.5 + (\sin^2(\sqrt{\sum_{i=1}^{10} x_i^2}) - 0.5)/$ $(1 + (\sum_{i=1}^{10} x_1^2)/1000)^2$	$[-100, 100]^{10}$	0	≈ 63 spheres
Levy4	$\sin^2(3\pi x_1) +$ $\sum_{i=1}^{3}(x_i - 1)^2(1 + \sin^2(3\pi x_{i+1})) +$ $(x_4 - 1)(1 + \sin^2(2\pi x_4))$	$[-10, 10]^4$	-21.502356	71000

Table 1. Test functions. D: search space; $f(x^*)$: global minimum value; K : number of local minima.

above, any suitable termination condition (time-based, quality-based, etc) is applicable, just like in the case of metaheuristics.

Note that there are countless points where the algorithm can be optimized and fine-tuned, such as the branching step, the upper bound approximation, the load balancing step, and so on. There are various techniques to optimize the bounding step as well, such as using derivatives, etc. We intentionally keep the most basic version, which in this form has very few parameters. One of them is the number of samples in the upper bound approximation that we fix at 8. The other is the number of nodes the initial problem is sent to at startup time. This will be 1 for networks without failures, and 10 for networks with churn (nodes joining and leaving).

3 Experimental Results

The algorithms described above were compared empirically using the P2P network simulator PeerSim [14].We first describe the experimental setup and methodology and subsequently we present and discuss results.

3.1 Experimental Setup

We selected well-known test functions as shown in Table 1. We included Sphere2 and Sphere10 as easy unimodal functions. Griewank10 is similar to Sphere10 with high frequency sinusoidal "bumps" superimposed on it. Schaffer10 is a sphere-symmetric function where the global minimum is surrounded by deceptive spheres. These two functions were designed to mislead local optimizers. Finally, Levy4 is not unlike Griewank10, but more asymmetric, and involves higher amplitude noise as well. Levy4 is in fact specifically designed to be difficult for interval arithmetic-based approaches.

We considered the following parameters, and examined their interconnection during the experiments:

- **network size** (N): the number of nodes in the network
- **running time** (t): the duration while the network is running. Note that it is is *not* the sum of the running time of the nodes. The unit of time is one function evaluation.

– **function evaluations** (E): the number of *overall* function evaluations performed in the network
– **quality** (ϵ): the difference of the fitness of the best solution found in the entire network and the optimal fitness

For example, if $t = 10$ and $N = 10$ then we know that $E = 100$ evaluations are performed.

Recall, that the simulated network consists of independent nodes that are connected only via an error-free message passing service. Messages are delayed by a uniform random delay drawn from $[0, t_{eval}/2]$ where t_{eval} is the time for one function evaluation. In fact, t_{eval} is considerable in realistic problems, so our model of message delay is rather pessimistic.

To simulate churn, in some of the experiments nodes are replaced with a certain probability (churn rate) with uninitialized nodes in any given fixed-length time. For simplicity we applied one fixed churn rate: 1% of nodes are replaced during a time interval taken by 20 function evaluations. The actual wall-clock time of one function evaluation has a large effect on how realistic this setting is. In real P2P networks the observed churn rate is around 0.01% per second, corresponding to 2-3 hour uptimes on average [15]. In our setting we allow for 2000 function evaluations during average uptime, which maps to 5 seconds per function evaluation. If a function is faster to evaluate, our churn rate setting becomes more pessimistic.

To simplify discussion, we assume that the startup of the protocol is synchronous, that is, all nodes in the network are informed at a certain point in time that the optimization process should begin. The fine details of the startup process is out of the scope of this paper, but even in the worst case, in the lack of synchrony and a priori knowledge at the nodes, gossip-based solutions can be applied that are orders of magnitude faster than the timescale of the optimization task.

3.2 Results and Discussion

Since neither of the algorithms is fine tuned, and since our focus is the exploitation of very large networks, here we are interested in understanding the overall scaling behavior of the algorithms. We focus on two key properties in this context: (i) scaling with the constraint of a fixed amount of available function evaluations, and (ii) with the constraint of having to reach a certain solution quality.

Our first set of experiments involves running the two algorithms with and without churn until 2^{20} function evaluations are consumed.[4] There are two questions one can ask: what is the solution quality that is reached, and what is the running time of the algorithms?

Solution quality is illustrated in Figure 1. The first clear effect is that for the larger networks, where network size is close to the available function evaluations, performance degrades quickly in all cases. This is not surprising, as in that case there are only very few evaluations available at all nodes, so search degrades to random search *if* the algorithm is greedy and wants to utilize all available resources as quickly as possible.

[4] We note here that for B&B, one cycle of Algorithm 1 was considered to take 20 evaluations, that is, in addition to the $2 \cdot 8 = 16$ normal evaluations, the interval-evaluation was considered to be equivalent to 4 evaluations (based on empirical tests on our test functions).

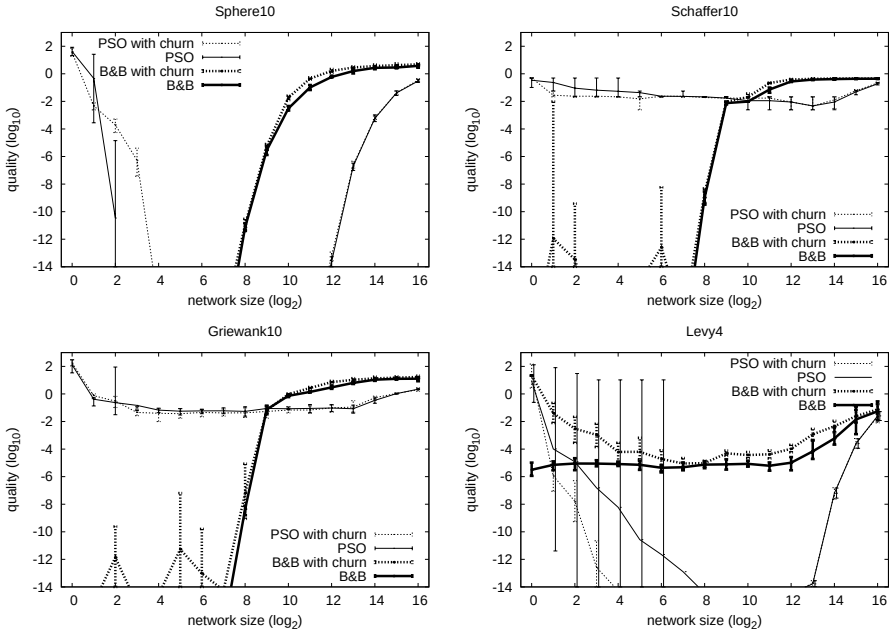

Fig. 1. Solution quality as a function of network size and churn. Geometric mean is plotted (average digits of precision) with error bars indicating the 10% and 90% percentiles of 10 experiments (100 experiments for the more unstable Levy4). Note that lower values for quality are better (indicate more precise digits).

There is an interesting case though: Sphere2. It is not shown because it is so easy for both algorithms that it is solved to optimality by P2P B&B in all cases, and by PSO in almost all cases except for the largest networks. The interesting effect we can discover is that the B&B approach "refuses" to utilize the entire network, because it cannot generate enough promising intervals (pruning is "too" efficient) and therefore it can deliver optimal solutions irrespective of network size, but at the cost of longer running times (see also Figure 2, as explained later). Depending on the context, this effect can be very advantageous but harmful as well.

Another observation is that, while B&B is very efficient on the smaller networks, PSO consistently outperforms B&B on the large networks. Whereas B&B can never benefit from larger network sizes (since it only increases the chance of processing some intervals unnecessarily), PSO has an optimal network size that represents enough possibility for exploration, but that also allocates enough function evaluations for each node to perform exploitation as well.

Function Levy4, that is hard for B&B, turns out to be easier for PSO, where PSO significantly outperforms B&B (note, that in the case of Griewank10 and Schaffer10, the situation is the opposite, and the sphere functions are easy for both algorithms). On Levy4, PSO does actually get stuck in bad local optima occasionally, but it can break out sufficiently often to provide a good average performance, whereas B&B gets bogged down not being able to do enough pruning due to the characteristics of the function.

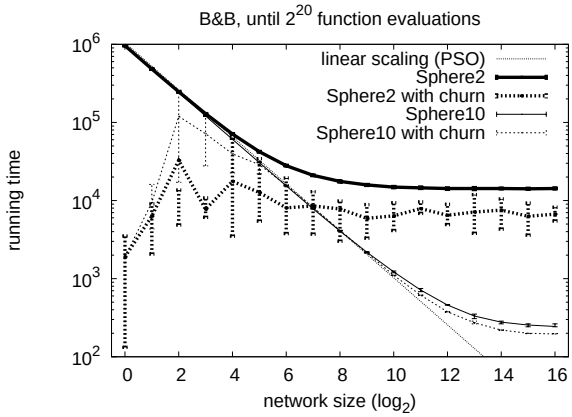

Fig. 2. Running time of P2P B&B to reach 2^{20} evaluations. Average is shown with error bars indicating the 10% and 90% percentiles of 10 experiments

A further support for this explanation is the curious effect of churn. On Levy4, churn increases the ability of PSO to break out of local optima via, in effect, restarting the nodes every now and then, while of course the global best solution never gets erased; it keeps circulating in the network via gossip. Indeed, in the experiments with churn, the performance of PSO is both better on average and more stable (has a lower variance). Again, just like larger networks, for B&B churn is always guaranteed to be harmful or neutral at best; Figure 1 also supports this observation.

As of running time, P2P PSO always fully utilizes the network by definition, assuming a synchronous startup of the protocol, so its running time is $2^{20}/N$. This of course cannot be interpreted as linear scaling, since the actual quality of the output will be different in different network sizes.

In the case of P2P B&B, the situation is more complex, as illustrated by Figure 2. Only the sphere functions are shown: the other functions behave almost identically to Sphere10. For larger networks the curve leaves linear scaling since there startup effects start to become significant: B&B needs $O(\log N)$ cycles of its main loop (due to gossip-based load balancing) until sub-tasks reach all nodes. In addition, for problems that are especially easy, such as Sphere2, there are simply not enough sub-tasks to distribute because pruning is too efficient. This way, the algorithm never utilizes more than a given number of nodes, independently of how many are available. For more difficult problems, this effect kicks in at much larger network sizes.

In the case of churn, for small network sizes there is a significant probability that all nodes get replaced at the same time (that is, before old nodes could communicate with new ones). In such cases all sub-tasks get lost and the optimization process stops: hence the short running times, that do never reach 2^{20} evaluations.

Figure 3 shows results illustrating our second question on scaling: the time needed to reach a certain quality. The most remarkable fact that we observe is that on problems that are easy for B&B, it is extremely fast (and also extremely efficient, as we have seen) on smaller networks, but this effect does not scale up to larger networks. In fact, the

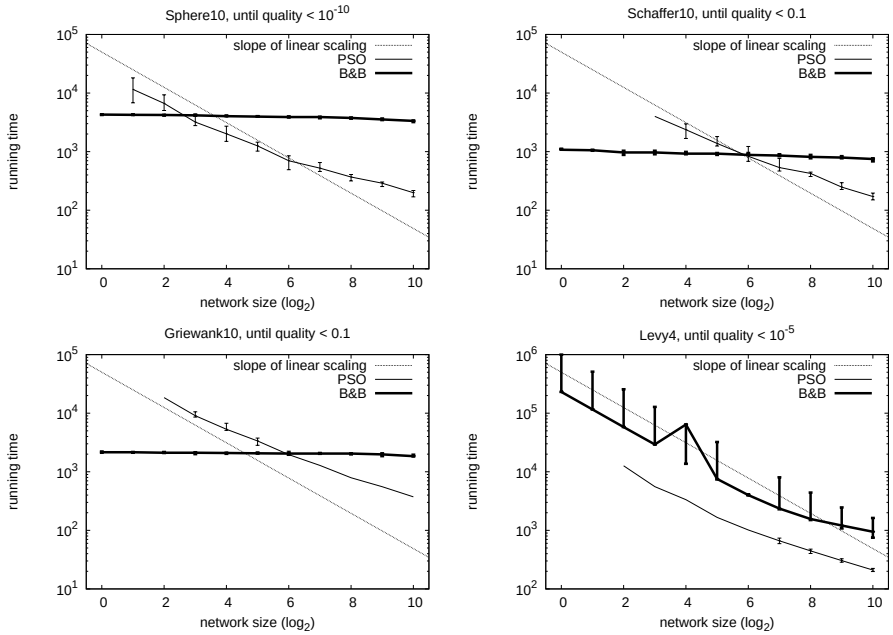

Fig. 3. Running time to reach a certain quality (see plot titles). The median is plotted with error bars indicating the 10% and 90% percentiles of 10 experiments (100 experiments for the more unstable Levy4). Missing error bars or line points indicate that not enough runs reached the quality threshold within 2^{20} evaluations (when the runs were terminated).

additional nodes (as we increase network size) seem only to add unnecessary overhead. On Levy4, however, we observe scaling behavior similar to that of PSO.

4 Conclusions and Future Work

We have seen evidence that the set of difficult problems is different for the two algorithms tested, and overall they both show a rich set of behavioral patterns with respect to various aspects of scaling.

An interesting observation is that parameters of the environment, such as size and failure patterns should best be interpreted as (meta-)algorithm parameters controlling exploration and exploitation. In the case of B&B this meta-level operates on the intervals: if we increase network size, selection pressure decreases: more intervals get evaluated and "branched". However, since B&B is extremely conservative with the removal of intervals, decreasing selection pressure often results in increasing overhead, if the problem at hand is easy.

For P2P PSO, increasing the network size is equivalent to increasing the population size. Interestingly, a non-zero churn rate introduces a restarting operator for PSO, that can in fact increase performance on at least some types of problems.

Unlike in small-scale controlled environments, in P2P networks system parameters (like network size and churn rate) are non-trivial to observe and exploit. An exciting

research direction is to monitor these parameters (as well as the performance of the algorithm) in a fully-distributed way, and to design distributed algorithms that can adapt automatically.

References

1. Kermarrec, A.M., van Steen, M., eds.: ACM SIGOPS Operating Systems Review 41. (October 2007) Special issue on Gossip-Based Networking.
2. Biazzini, M., Montresor, A., Brunato, M.: Towards a decentralized architecture for optimization. In: Proc. of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS'08), Miami, FL, USA (April 2008)
3. Talbi, E.G., ed.: Parallel Combinatorial Optimization. Wiley (2006)
4. Steinmetz, R., Wehrle, K., eds.: Peer-to-Peer Systems and Applications. Volume 3485 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2005)
5. Wickramasinghe, W.R.M.U.K., van Steen, M., Eiben, A.E.: Peer-to-peer evolutionary algorithms with adaptive autonomous selection. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM Press New York, NY, USA (2007) 1460–1467
6. Laredo, J.L.J., Eiben, E.A., van Steen, M., Castillo, P.A., Mora, A.M., Merelo, J.J.: P2P evolutionary algorithms: A suitable approach for tackling large instances in hard optimization problems. In: Proceedings of Euro-Par. (2008) to appear.
7. Jelasity, M., Montresor, A., Babaoglu, O.: A modular paradigm for building self-organizing peer-to-peer applications. In Di Marzo Serugendo, G., Karageorgos, A., Rana, O.F., Zambonelli, F., eds.: Engineering Self-Organising Systems. Volume 2977 of Lecture Notes in Artificial Intelligence., Springer (2004) 265–282 invited paper.
8. Bendjoudi, A., Melab, N., Talbi, E.G.: A parallel P2P branch-and-bound algorithm for computational grids. In: Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2007), Rio de Janeiro, Brazil (2007) 749–754
9. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.M., van Steen, M.: Gossip-based peer sampling. ACM Transactions on Computer Systems **25**(3) (August 2007) 8
10. Arenas, M.G., Collet, P., Eiben, A.E., Jelasity, M., Merelo, J.J., Paechter, B., Preuß, M., Schoenauer, M.: A framework for distributed evolutionary algorithms. In Merelo Guervós, J.J., Adamidis, P., Beyer, H.G., Fernández-Villacañas, J.L., Schwefel, H.P., eds.: Parallel Problem Solving from Nature - PPSN VII. Volume 2439 of Lecture Notes in Computer Science., Springer-Verlag (2002) 665–675
11. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87), Vancouver, British Columbia, Canada, ACM Press (August 1987) 1–12
12. Casado, L.G., Martínez, J.A., García, I., Hendrix, E.M.T.: Branch-and-bound interval global optimization on shared memory multiprocessors. Optimization Methods and Software (2008) Accepted for publication.
13. Ratschek, H., Rokne, J.: Interval methods. In Horst, R., Pardalos, P.M., eds.: Handbook of Global Optimization. Kluwer (1995)
14. PeerSim: http://peersim.sourceforge.net/
15. Castro, M., Costa, M., Rowstron, A.: Performance and dependability of structured peer-to-peer overlays. In: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04), IEEE Computer Society (2004)