Towards a Decentralized Architecture for Optimization

Marco Biazzini, Mauro Brunato and Alberto Montresor
University of Trento
Dipartimento di Ingegneria e Scienza dell'Informazione
via Sommarive 14, 38100 Trento, Italy
{biazzini,brunato,montresor}@disi.unitn.it

Abstract

We introduce a generic framework for the distributed execution of combinatorial optimization tasks. Instead of relying on custom hardware (like dedicated parallel machines or clusters), our approach exploits, in a peer-to-peer fashion, the computing and storage power of existing, off-the-shelf desktops and servers. Contributions of this paper are a description of the generic framework, together with a first instantiation based on particle swarm optimization *(PSO). Simulation results are shown, proving the efficacy of our distributed PSO algorithm in optimizing a large number of benchmark functions.*

1 Introduction

Distributed optimization has a long research history [14]. Most of the previous work assumes the availability of either a dedicated parallel computing facility, or, in the worst case, 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 goal of our work is to investigate an alternative approach to distributed function optimization. The idea is to adopt recent results in the domain of large-scale decentralized systems and peer-to-peer (P2P) systems, where a large collection of loosely-coupled machines cooperate to achieve a common goal. 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.

The applicative scenario we have in mind is 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 scenario is not unlike a Grid system [12]; 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 [4].

An interesting question is whether it is possible to come up with an alternative approach, where a distributed algorithm spreads the load of *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).

Two opposite techniques could be followed to design a distributed optimization algorithm:

- **Without coordination: exploiting stochasticity** — Global optimization algorithms are stochastic by nature; in particular, the first evaluation is not driven by prior information, so the earliest stages of the search require some random decision. Different runs of the same algorithm can evolve in a very different way, so that *parallel independent execution* of identical algorithms with different random seeds yields a better expected outcome w.r.t. a single execution.

- **With coordination: exploiting communication** — Some optimization algorithms can be modeled as par-

allel processes sitting in a multi-processor machine supporting shared data structures. Processes can be coordinated in such a way that every single step of each process (i.e., decision on the next point to evaluate) is performed while taking into account information about all processes. In order for such approach to be efficient, the cost of sharing global information should not overcome the advantage of having many function evaluations performed simultaneously.

In between the two extremal cases presented above (no coordination at all or complete information), it is possible to imagine a wide spectrum of algorithms that perform individual searches with some form of loose coordination. The main contribution of this paper is to present a generic distributed framework that enables experiments into such spectrum, and to discuss a first instantiation of such framework based on *particle swarm optimization* (PSO) [10]. We experimentally demonstrate that on particular conditions, our algorithm shows better performance than the original (centralized) one.

Our algorithmic approach is based on the *epidemic* paradigm, a very light-weight approach to distributed computing. Originated in the context of databases [1], gossip protocols have proven to be able to deal with the high levels of unpredictability associated with P2P systems. Apart from the original goal of information dissemination (messages are "broadcast" through random exchanges between nodes), they are now used to solve several different problems: membership management [3], aggregation [5], topology management [2], etc.

The rest of the paper is organized as follows. Section 2 provides the background of the paper. Section 3 introduces the design of our framework and the decentralized PSO algorithm. Section 4 discusses the experimental results, while Section 5 concludes the paper.

2 Background

Particle Swarm Optimization PSO [10] is a nature-inspired method for finding global optima of functions of continuous variables. Search is performed iteratively updating a small number N (usually in the tens) of random "particles" (solutions), whose status information includes the current position vector $\boldsymbol{x}_i$, the current speed vector $\boldsymbol{v}_i$, the optimum point $\boldsymbol{p}_i$ and the *fitness* value $f(\boldsymbol{p}_i)$, which is the "best" solution the particle has achieved so far. Another "best" value that is tracked by the particle swarm optimizer is the global best position $\boldsymbol{g}$, in which the swarm has achieved the best fitness value obtained so far by any particle in the population.

After finding the two best values, every particle updates its velocity and positions as described by the following equations:

$$\boldsymbol{v}_i = \boldsymbol{v}_i + c_1 * rand() * (\boldsymbol{p}_i - \boldsymbol{x}_i) + c_2 * rand() * (\boldsymbol{g} - \boldsymbol{x}_i) \quad (1)$$

$$\boldsymbol{x}_i = \boldsymbol{x}_i + \boldsymbol{v}_i \quad (2)$$

In these equations, $rand()$ is a random number in the range $[0, 1]$, while c_1 and c_2 are learning factors. Usually $c_1 = c_2 = 2$. The pseudo code of the procedure is as follows:

```
foreach particle i do
    Initialize i;
end
while maximum iterations or
      minimum error criteria is not attained do
    foreach particle i do
        Calculate current fitness value f(x_i);
        if f(x_i) is better than f(p_i) then
            p_i ← x_i;
        end
    end
    g ← bestOf(p_i), i = 1 to N;
    foreach particle i do
        Calculate velocity v_i according to equation 1;
        Update position x_i according to equation 2;
    end
end
```

Particle speeds on each dimension are bounded to a maximum velocity $vmax_i$, specified by the user. If the sum of "accelerations" causes the velocity on that dimension to exceed it, then the velocity on that dimension will be limited to $vmax_i$.

PSO on incomplete topologies The above-described version of PSO assumes that all particles agree on the global best point $\boldsymbol{g}$ found so far, and is often referred to as the "classical" or "full-information" version. Effects of incomplete topologies on the performance of PSO have been studied for sociology-inspired small-world graphs [8] as well as other types of random graphs [11]. Such studies were motivated by the observation that incomplete topologies may prevent the system from concentrating too much on early-found local optima, therefore improving solution quality. Dynamic topologies based on particle clustering have also been proposed [9] to induce a differentiated behavior among groups of particles having similar local best positions. While full information has generally been shown to outperform partial topologies [13], our work focuses on a case where incomplete information is a consequence of network topology, and global data maintenance is not practical.

Epidemic Protocols The research on the application of epidemic protocols in distributed systems started with the seminal work of Alan Demers [1], who proposed several

Param. values			avg	min	max	Var
		F2	0.0	0.0	0.0	0.0
n	$1, 10, 100, 1000$	Zakharov	0.52043	0.23106	0.95915	0.02700
k	$1, 4, 8, 16, 32$	Rosenbrock	0.07979	3.27435E-7	3.98660	0.31785
e	$1000 \times n$	Sphere	2.49767E-51	1.56467E-53	2.20189E-49	0.80330
r	k	Schaffer	0.00972	0.00972	0.00972	0.00000
		Griewank	0.09849	0.01232	0.28627	0.00187

Table 1. Adopted values for the configuration parameters and best results of the first set of experiments - Solution quality vs swarm size

models for the decentralized dissemination of database updates. The models are as follows:

Anti-Entropy Each node periodically selects a random peer and performs an *information exchange* with it; the exact meaning of *exchange* depends on the specific application. In the case of database updates, exchanges may be characterized as *push* – the originator of the exchange sends its own updates to the peer; *pull* – the originator asks for updates from the peer; *push-pull* – both the operations are performed.

Gossip Whenever a node receive an update, it selects a small number k of peers and sends the update to them; if the update has already been received, the node may also decide to stop the spreading of the update with probability p. Values k and p presents a trade-off between the probability of reaching all the nodes and the communication overhead given by redundant messages.

One of the key requirement for implementing an epidemic protocols is the capability of selecting a random peer among all nodes. This is easy when the set of participants is fixed, small and known *a priori*; it is a problem by itself in case of dynamic, large-scale distributed systems. To solve this issue, the concept of *peer sampling service* has been devised; this service provides each node with a uniform random sample of the entire population of a P2P networks [3]. Interestingly enough, protocols implementing the peer sampling service are epidemic as well: push-pull exchanges are performed, in which random subsets of nodes are shuffled among the nodes.

Once solved this issue, a large collection of problems may be solved on top of the peer sampling service. Different problems require specific exchange mechanisms; for example, it is possible to compute the average aggregation over a collection of values stored at peer nodes by simply averaging the values exchanged by a pair of nodes [5].

3 Generic Framework and Specific PSO Algorithm

This section describes the architecture of our optimization framework. We first present the assumptions over the underlying system model; we then present the generic architecture and how we instantiate it based on the PSO paradigm.

3.1 System Model

We consider a network consisting of a large collection of *nodes*. The network is highly dynamic; new nodes may join at any time, and existing nodes may leave, either voluntarily or by *crashing*. Since voluntary leaves may be simply managed through "logout" protocols, in the following we consider only node crashes. Byzantine failures, with nodes behaving arbitrarily, are excluded from the present discussion. We assume nodes are connected through an existing routed network, such as the Internet, where every node can potentially communicate with every other node. To actually communicate with another node, however, a node must know its *identifier*, e.g. a pair ⟨IP address, port⟩.

The nodes known to a node are called its *neighbors*, and as a set are called its *view*. Together, the views of all nodes define the topology of the overlay network. Given the large scale and the dynamism of our envisioned system, views are typically limited to small subsets of the entire network. Views can change dynamically, and so the overlay topology.

3.2 The Architecture

The architecture of our generic framework is composed of three 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

Param. values			min
		F2	0.0
n	$2^i,\ i = 0\ to\ 16$	Zakharov	0.0
k	$1, 4, 8, 16, 32$	Rosenbrock	2.8890E-29
e	2^{20}	Sphere	0.0
r	k	Schaffer	0.0097
		Griewank	0.0221

Table 2. Adopted values for the configuration parameters and best results of the second set of experiments

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, opportunely selected based on both local information (provided by this module, based on past history) and remote information (provided by the coordination service). This module depends heavily on the particular optimization mechanism adopted.
- The *coordination service*, as the name implies, coordinates the selection of points to be evaluated in the search space. Examples of possible strategies include, for example, partitioning of the search space in non-overlapping zones under the responsibility of each node; or broadcasting of search information to all nodes of the system.

3.3 Distributed PSO

We demonstrate the applicability of our framework by discussing a first example instantiation of the architecture. Here, the topology service is based on *peer sampling*, which is a well-known technique for creating and maintaining a random topology over the entire network. The function optimization service is based on *particle swarm optimization*, while coordination is based on an epidemic diffusion algorithm. Each of these components are described in the subsequent sections.

3.3.1 Topology Service: Peer Sampling

The topology service is provided by NEWSCAST [7], which has proven to be a valuable building block to implement several P2P protocols [4]. We provide here a brief description of the protocol and its characteristics.

Each NEWSCAST node maintains a view containing c node descriptors, each of them composed of a remote node identifier and a logical time-stamp. NEWSCAST is based on the epidemic paradigm: periodically, each node (i) selects a random peer from its partial view; (ii) updates its local descriptor and (iii) performs a *view exchange* with the selected peer, during which the two nodes send each other their views, merge them, and keep the c freshest descriptors.

This exchange mechanism has four effects: views are continuously shuffled, creating a topology that is close to a random graph with out-degree c; views can be considered a random sample of the entire network (hence the name "peer sampling"); the resulting topology is strongly connected (according to experimental results, choosing $c = 20$ is already sufficient for very stable and robust connectivity); and finally, the overlay topology is self-repairing, since crashed nodes cannot inject new descriptors any more, so their information quickly disappears from the system.

3.3.2 Function Optimization Service: Distributed PSO

At each node p, the PSO function optimization service (briefly, PSO service) maintains and executes a particle swarm of size k. Each particle $i \in \{1, \ldots, k\}$ is characterized by its *current position* $\boldsymbol{p}_i^p$, its *current velocity* $\boldsymbol{v}_i^p$ and the *local optimum* $\boldsymbol{x}_i^p$. Slightly departing from the standard PSO terminology, we say that each swarm of a node p is associated to a *swarm optimum* $\boldsymbol{g}^p$, selected among the particles local optima. Clearly, different nodes may know different swarm optima; we identify the best optimum among all of them with the term *global optimum*, denoted $\boldsymbol{g}$.

The PSO service works by iterating over the particles, updating the current position and velocity as described in Section 2, and selecting, after each evaluation, the best local optimum as the swarm optimum.

3.3.3 Coordination Service: Global Optimum Diffusion

The coordination service is implemented as an anti-entropy epidemic algorithm, whose task is to spread information about the global optimum among nodes. Such algorithm works as follows: periodically, each node p initiates a communication with a random peer q, selected through the peer sampling service described in the previous section. p sends the pair $\langle \boldsymbol{g}^p, f(\boldsymbol{g}^p) \rangle$ to q, i.e. its current swarm optimum and its evaluation. When q receives such a message, it compares the swarm optimum of p with its local optimum; if $f(\boldsymbol{g}^p) < f(\boldsymbol{g}^q)$, then q updates its swarm optimum with the received optimum ($\boldsymbol{g}^q = \boldsymbol{g}^p$); otherwise, it replies to p by sending $\langle \boldsymbol{g}^q, f(\boldsymbol{g}^q) \rangle$.

The rate r at which messages are sent by the anti-entropy algorithm is a parameter of the algorithm which is related to the communication overhead: The more often messages are sent, the larger bandwidth is required.

Param. values			avg	min	max	Var
		F2	1.0120E-8	1.3080E-10	1.5402E-7	0.4798
n	$10, 100, 1000$	Zakharov	0.0285	0.0076	0.0570	9.4755E-5
k	16	Rosenbrock	11.6670	5.4879	170.8049	530.4765
e	1000	Sphere	0.0027	7.4966E-4	5.8552E-3	0.0259
r	$2, 4, 6, \ldots, 64$	Schaffer	0.0105	0.0097	0.0372	0.0165
		Griewank	0.7133	0.4053	0.8868	0.0094

Table 3. Adopted values for the configuration parameters and best results of the third set of experiments

3.3.4 Robustness to Failures

No special provisions are taken to deal with failures. Messages initiated by random peers can be eventually be lost, with the only effect of slowing down the spreading of information, thanks to our distributed PSO algorithm. Nodes may be subject to churn without affect the consistency of the overall computation, thanks to the robustness provided by the chosen topology service (NEWSCAST). Joining nodes start with a random position and velocity; as soon as they receive an epidemic message containing the swarm optimum and its evaluation, their swarm optimum is updated.

4 Experimental Results

Simulation scenarios We tested our distributed framework using a P2P simulator called Peersim [6]. Each experiment simulates n nodes, each of them runs a swarm of k particles, that repeatedly evaluate a function f. Globally, the n swarms perform e total evaluations, evenly distributed among their particles, and each node exchanges the information about the global optimum with a random peer every r local function evaluations (r is the *cycle length* of the epidemic algorithm implemented in the coordination service). Experiments are repeated 50 times, and individual dots for each experiment are shown whenever possible.

Functions We focused our attention on six well known testing functions (De Jong's F2, Zakharov, Rosenbrock, Sphere, Schaffer's F6, Griewank), whose evaluations produced a quite interesting set of results. While the first (being a Rosenbrock specialization) is defined in a 2-dimensional domain space, all the others have been evaluated in a 10-dimensional domain space. These function are widely used in order to benchmark optimization algorithms, therefore we omit their analytical expression here. The benchmark functions have been carefully selected to provide a large spectrum of behaviors with respect to their solvability with PSO. We can say that F2 is 'easy'; Zakharov, Sphere and Rosenbrock present some 'nice' outcomes; whereas Griewank and Schaffer are the most difficult to treat. Although we are aware that this is not an insightful and thorough classification, we believe that this set represents a diversity of behaviors w.r.t. to PSO.

Figures of merit In the following, three figures of merit are considered: the *solution quality*, measured as the distance between the best known global optimum and the solution obtained by our mechanism; the *total number of evaluations*, performed globally by all swarms; and the *total time* required to complete a task. Time is measured as the number of evaluations locally performed at each node; we deliberately avoid to evaluate actual time, as it depends on the particular function evaluated and the computing power of nodes.

Other figures of merit are only briefly mentioned here, as they can easily obtained from the other parameters. *communication overhead* is produced by both the NEWSCAST layer and the coordination service. Depending on the expected rate of churn, NEWSCAST cycles length could be expected in the range $[10s, 60s]$; during a cycle two messages of few hundred bytes are exchanged per node, inducing an overhead of few bytes per second. Similar considerations can be done for the coordination service, but the overhead depends on the specific gossip rate adopted. The system present a high *robustness to churn*, i.e. the capability of dealing with nodes continuously joining and leaving. In fact, the reliability of the computation does not depend on any single point of failure; even if a large portion of the network fails, the computation will end successfully, slowing down proportionally to the number of failed nodes.

4.1 Evaluating Quality

The first set of experiments is aimed at finding out as the solution quality changes with respect to the number of computing particles per node, and with respect to the number of involved nodes. In each experiment, we report the solution quality of the best global optimum found after 1000 evaluations of the function per node. Gossiping rate is equal to k, meaning that a gossip exchange is performed after all the particles within the same swarm had been evaluated once.

Param. values			avg	min	max	Var
		F2	235940.0	186000.0	271000.0	2.48384E8
n	$2^i,\ i = 0\ to\ 10$	Zakharov	511800.0	478000.0	540000.0	1.42082E8
k	$1, 4, 8, 16$	Rosenbrock	1927000.0	1740000.0	2033000.0	9.75250E9
e	2^{20}	Sphere	36740.0	16000.0	66000.0	1.46687E8
r	k	Schaffer	192020.0	151000.0	224000.0	2.29734E8
		Griewank	–	–	–	–

Table 4. Adopted values for the configuration parameters and best results of the fourth set of experiments

The idea here is the following: how the quality of solution change, if we are willing to dedicate a fixed quantity of time (i.e., number of evaluations per node) but a variable number of nodes to the computation? What is the influence of the main configuration parameter (swarm size) on the results?

Figure 1 shows the outcomes. As we can see, there is a profitable relation between the number of nodes and the solution quality. But this fact only holds while particles per node are bounded from 8 to 16.

The second set of experiments, reported in Figure 2, start with a different assumption: we are willing to provide up to 2^{20} evaluations (total) of the function. What are the best number of nodes and the best configuration parameters to obtain the maximum solution quality?

The best results are obtained when 8 to 256 particles are working, no matter how they are partitioned among the nodes. We can actually slightly narrow the empirically estimated range; a look at the raw data tells us that the most reliable particles range to evaluate the "nice functions" is 16 to 64 particles. Besides, the overhead due to gossiping communications is practically negligible. We can see this by paying attention to the fact that differently sized networks reach the same performance as soon as their number of active particles becomes the same.

So what about having different number of nodes working at the same task? The good news is given exactly by the combination of what we have just observed. The performance of PSO is mostly related to the number of working particles and not to their belonging to a particular node. This means we can choose to have different numbers of involved nodes while keeping the solution quality as much as accurate. Thus we have an effective way to distribute the load of a PSO computation through different machines while obtaining the same performance we would have on a single, but much more powerful, machine.

4.2 Evaluation of Cycle Length

An interesting issue is to understand if and how the cycle length can affect the effectiveness of the computation. For this reason, we ran a set of experiments in which cycle length rate varies between 2 and 64 local function evaluations, while all nodes have 16 particles.

Figure 3 shows the results. As it was easily expected, we discovered that performance is not heavily influenced by the gossiping rate by itself, but rather by ratio between the gossiping rate and the number of evaluations performed since the last message exchange. As this ratio tends to one, performance is sensibly better. We can therefore state that the more the swarms are exchanging information, the better the solution quality is expected to be. What follows is good news for us: the distributed nodes interaction through the adopted protocol is effective and well mimics the information sharing mechanism among the particles within the ilk swarm.

We can also see that the size of the network is important, if all other parameters (like the number of particles per swarm) are fixed. But it is also worth noticing that if the problem is inherently hard to solve for the algorithm, the cycle length is obviously less crucial, because no remarkably better value becomes available for the nodes since the former updates.

4.3 Evaluating Time

The final task is to evaluate how the total time required to obtain a given solution quality changes with respect to network and swarm size. In the last set of experiments, we stopped the simulation as soon as the global solution quality reaches a reasonable threshold ($1E-10$).

Figure 4 shows that the required time is inversely proportional to the number of nodes, but proportional to the size of the swarms. This means that what is crucial here is the convergence speed of the best values within the swarms. Of course the exact amount of time required to converge – in an absolute sense – is strictly dependent on the 'niceness' of the evaluated function w.r.t. the adopted algorithm. The good news we can collect is that performing the optimization task in a distributed and decentralized fashion causes no detriment to the computation and does not adversely affect the quality of the results.

To conclude, it is worth recalling that it is not the per-

formance of the PSO algorithm on the various functions that matters. What is interesting in our research is seeing how the performance varies when the number of nodes and all the other P2P-related settings change. Our experiments point out that a distributed P2P-networking design of the system can actually improve the solution quality. Furthermore, they give us some guidelines about which features are crucially related to performance enhancing and which seem not to be related at all.

5 Conclusions

In this paper, we discussed a novel approach to distributed function optimization, based on recent results in the domain of large-scale decentralized P2P systems. Instead of requiring a specialized architecture or a central server, a distributed algorithm spreads the load of a single optimization task among a group of nodes in a totally decentralized way. The advantages of such an approach are the extreme robustness and scalability, plus the capability of exploiting existent idling resources. We described a framework architecture based on the epidemic paradigm, a very lightweight approach to distributed computing that can be easily adopted to other optimization paradigms.

We ran a large number of experiments in a simulated P2P environment, testing different configurations, in order to find out how the solution quality and the amount of time required to have a certain solution quality change with respect to the number of involved nodes.

As resulting from the presentation and the discussion of our outcomes, we showed that: (i) distributed nodes interaction through the adopted protocol is effective and tantamount to the information sharing mechanism of the adopted solver; (ii) the overhead due to epidemic communications is negligible. Networks of different sized will achieve similar performance, if they host the same number of solver processors (in this case, PSO particles); (iii) distributed and decentralized architecture cause no detriment to the optimization task and does not affect the quality of the results; (iv) we devised and tested an effective way to distribute the load of a PSO computation through different machines while obtaining the same performance we would have on a single, but much more powerful, machine.

Our future work will include the implementation of various different solvers to enrich the function evaluation service and then be able to test module diversification among peers (same solver with different parameters and configurations, different solvers, diverse domain space allocation, etc.).

Acknowledgements

Work supported by the project CASCADAS (IST-027807) funded by the FET Program of the European Commission. We are grateful to Márk Jelasity for his comments on an earlier version of this paper.

References

[1] A. Demers et al. Epidemic algorithms for replicated database maintenance. In *Proc. of the 6th Annual ACM Symp. on Principles of Distributed Computing (PODC'87)*, pages 1–12, Vancouver, Canada, August 1987.

[2] M. Jelasity and O. Babaoglu. T-Man: Gossip-based overlay topology management. In *Engineering Self-Organising Systems: Third International Workshop (ESOA 2005)*, volume 3910 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2006.

[3] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Middleware 2004*, volume 3231 of *Lecture Notes in Computer Science*, pages 79–98. Springer-Verlag, 2004.

[4] M. Jelasity, A. Montresor, and O. Babaoglu. A modular paradigm for building self-organizing peer-to-peer applications. In *Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering*, number 2977 in Lecture Notes in Computer Science, pages 265–282. Springer-Verlag, 2004.

[5] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(1):219–252, 2005.

[6] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. Peersim - peer-to-peer simulator. `http://peersim.sourceforge.net`.

[7] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3):8, Aug. 2007.

[8] J. Kennedy. Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In *Proc.* 1^{st} *Congress on Evolutionary Computation (CEC'99)*, pages 1931–1938, 1999.

[9] J. Kennedy. Stereotyping: Improving particle swarm performance with cluster analysis. In *Proc.* 2^{nd} *Congress on Evolutionary Computation (CEC'00)*, pages 1507–1512, 2000.

[10] J. Kennedy and R. C. Eberhart. Particle swarm optimization. *IEEE Int. Conf. Neural Networks*, pages 1942–1948, 1995.

[11] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proc.* 4^{th} *Congress on Evolutionary Computation (CEC'02)*, pages 1671–1676, May 2002.

[12] C. Kesselman and I. Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[13] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210, June 2004.

[14] E. G. Talbi. *Parallel Combinatorial Optimization*. John Wiley and Sons, USA, 2006.

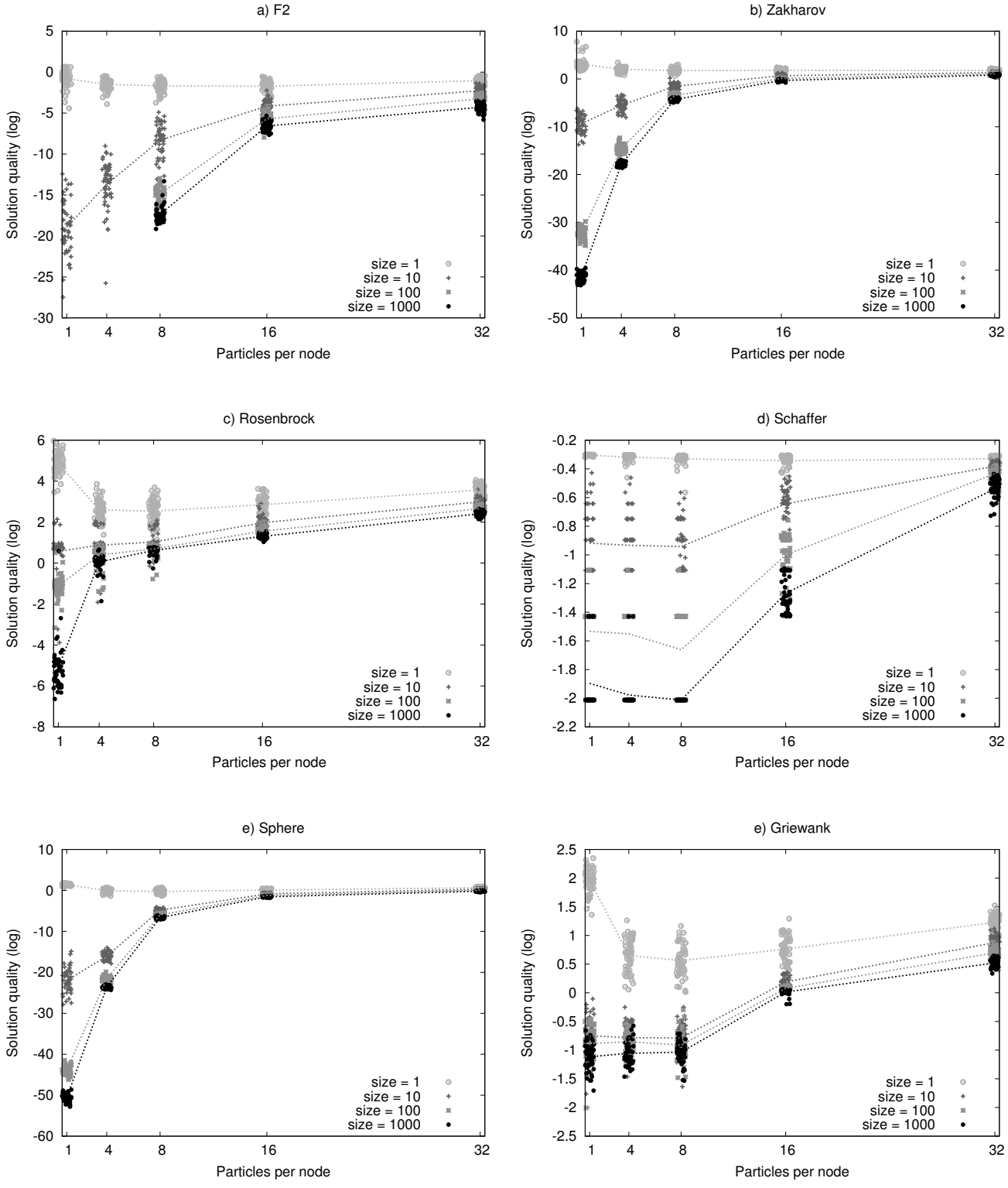

Figure 1. First set – Solution quality vs swarm size

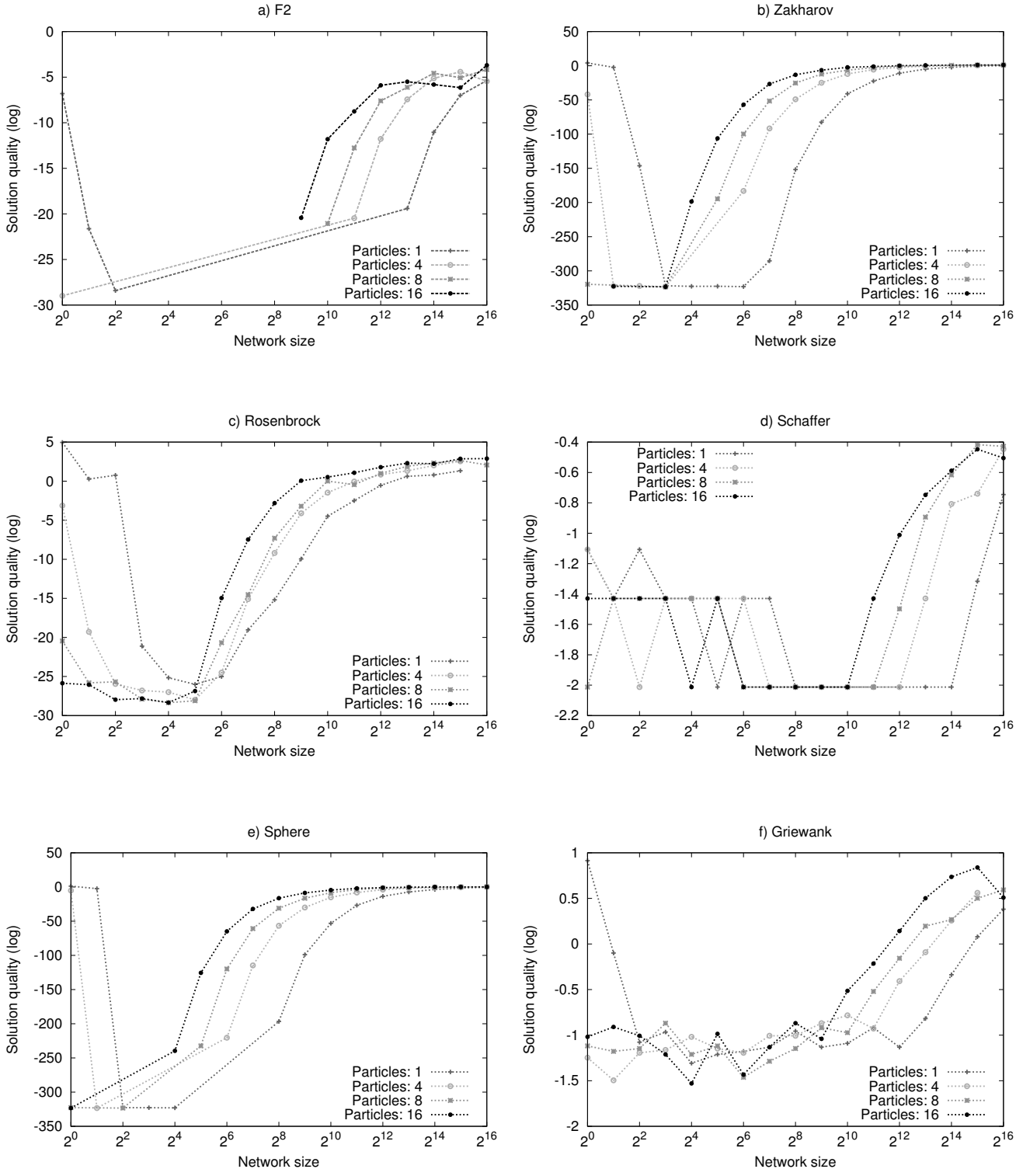

Figure 2. Second set - Solution quality vs network size

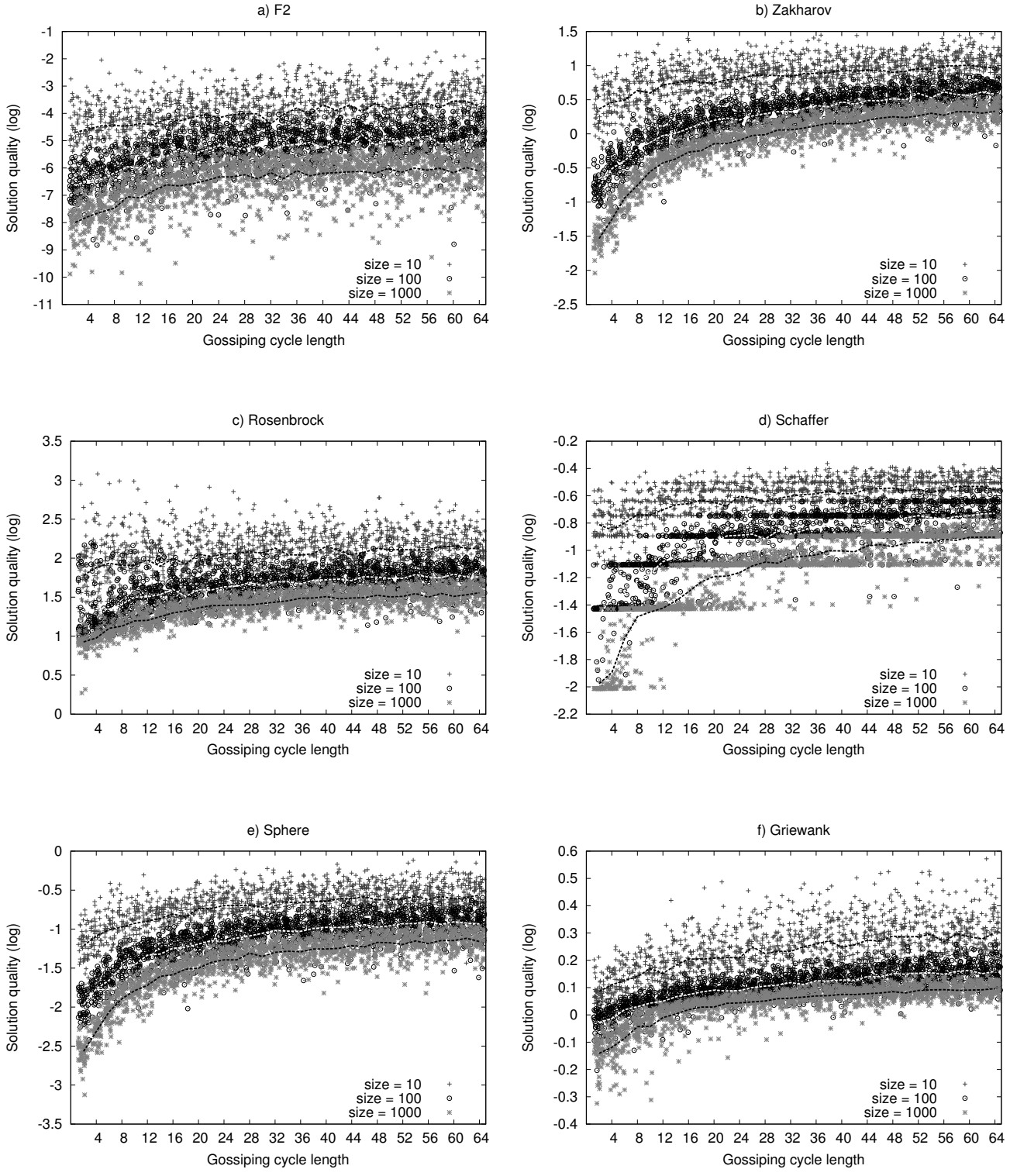

Figure 3. Third set - Solution quality w.r.t. cycle length

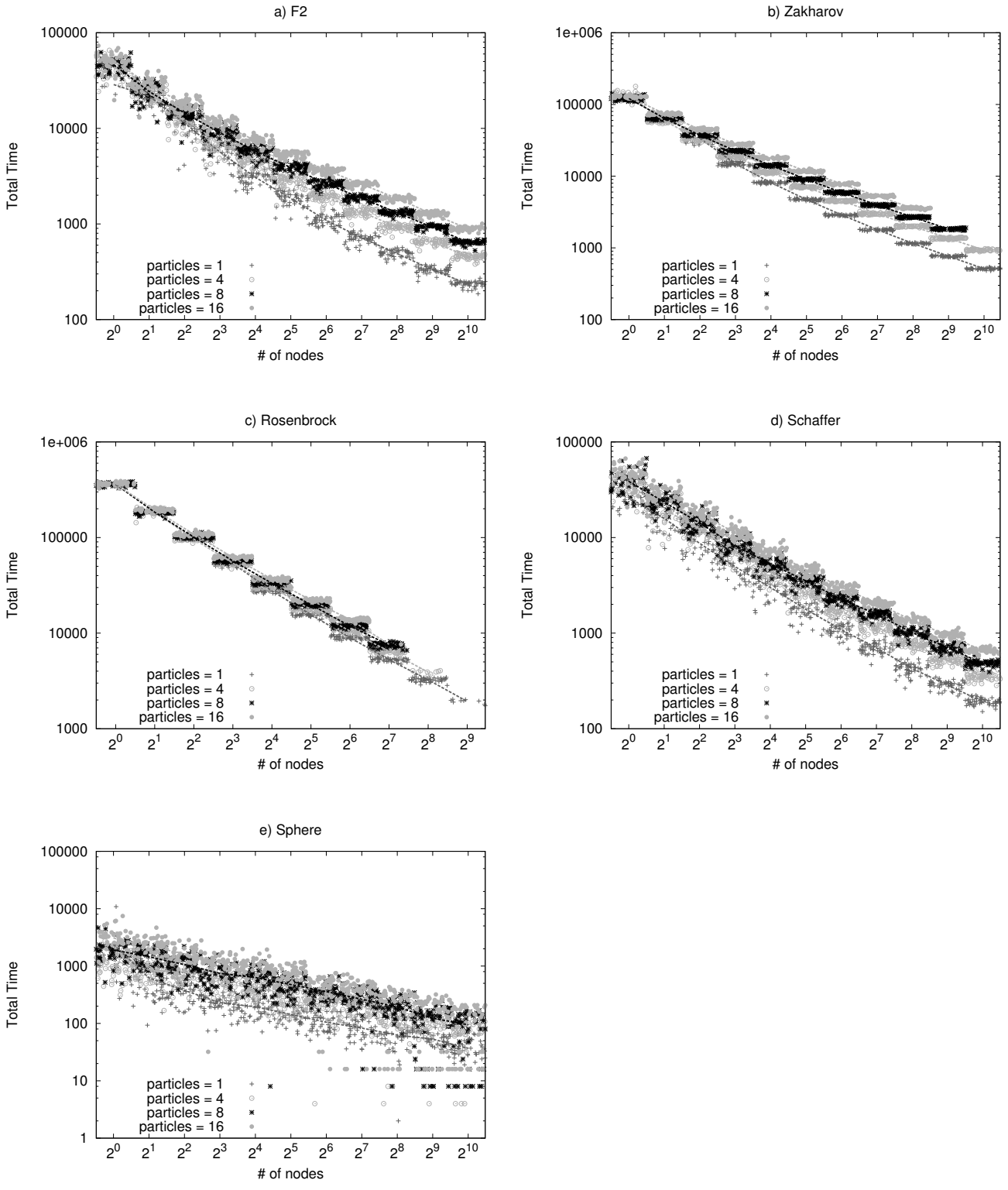

Figure 4. Fourth set - Total time vs network size