11

P2P B&B and GA for the Flow-Shop Scheduling Problem

A. Bendjoudi[1], S. Guerdah[2], M. Mansoura[2], N. Melab[3], and E-G. Talbi[3]

[1] Université A/Mira de Béjaia, Département d'Informatique
CEntre de Recherche sur l'Information Scientifique et Technique (CERIST),
Laboratoire d'Ingénierie et Théories des Systèmes Informatiques, Algiers, Algeria
ahcene.bendjoudi@gmail.com
[2] Université Mouloud Mammeri de Tizi Ouzou,
Département Informatique, Tizi Ouzou, Algeria
{samir.guerdah,madjid.mansoura}@gmail.com
[3] Université des Sciences et Technologies de Lille, France
Laboratoire d'Informatique Fondamentale de Lille LIFL
UMR CNRS 8022, Cité scientifique
INRIA Futurs - DOLPHIN
59655 - Villeneuve d'Ascq cedex - France
{talbi,melab}@lifl.fr

Summary. Solving exactly Combinatorial Optimization Problems (COPs) using a Branch-and-Bound algorithm (B&B) requires a huge amount of computational resources. The efficiency of such algorithm can be improved by its hybridization with meta-heuristics such as Genetic Algorithms (GA) which proved their effectiveness, since they generate acceptable solutions in a reasonable time. Moreover, distributing at large scale the computation, using for instance Peer-to-Peer (P2P) Computing, provides an efficient way to reach high computing performance. In this chapter, we propose ParallelBB and ParallelGA, which are P2P-based parallelization of the B&B and GA algorithms for the computational Grid. The two algorithms have been implemented using the ProActive distributed object Grid middleware. The algorithms have been applied to a mono-criterion permutation flow-shop scheduling problem and promisingly experimented on the Grid5000 computational Grid.

Keywords: P2P Computing, Branch and Bound, Genetic Algorithms, Grid Middleware, Flow-Shop Scheduling.

11.1 Introduction

In practice, many problems can be modelled as combinatorial optimization problems. These problems are often large and classed NP-hard [20] such as scheduling and quadratic assignment problems. To solve these problems, various methods were proposed in the literature. Meta-heuristics proved their effectiveness, since they generate acceptable solutions, in a reasonable time. Searching an exact solution for this kind of problem remains unpractical when the problem size grows,

F. Xhafa, A. Abraham (Eds.): Meta. for Sched. in Distri. Comp. Envi., SCI 146, pp. 301–321, 2008.
springerlink.com

because the execution time increases in an exponential way. To mitigate this constraint, hybridization between exact and heuristic methods and their parallelization are two effective ways in terms of improving the computing performances, in particular the use of large scale parallelism based on *Grid Computing* [19] or *Peer-to-Peer Computing* [34, 36].

Grid and P2P Computing are emerging technologies allowing to share various resources at a large scale. Grid Computing uses an infrastructure for globally sharing compute-intensive resources such as supercomputers or computational clusters. P2P Computing, using for instance *XtremWeb* [18] or *ProActive* [1], is based on the exploitation of non used CPU cycles or completely idles. Nowadays, these two technologies provide effective tools to achieve high performance in solving large-scale problems. Particularly, solving exactly complex combinatorial optimization problems is a good challenge for GRID/P2P Computing.

The Branch-and-Bound algorithm (*B&B*) is the most known method for exact resolution of combinatorial optimization problems (*COP*). B&B explores the search space by implicitly enumerating subtrees. The whole exploration of this space is impossible considering the exponential increase in the number of solutions when the size of the problem increases. The use of good lower and upper bounds reduces the number of subtrees to enumerate. Meta-heuristics provide sub-optimal solutions in a reasonable time (they allow us to reach an acceptable solution in a short time). Genetic Algorithms (GAs) belongs to Evolutionary Algorithms (EAs) which make use of a randomly generated population of solutions. The initial population is enhanced through a natural evolution process. At each generation of the process, the whole population or a part of the population is replaced by newly generated individuals. Several parallel versions of B&B [11, 13, 35, 39, 40, 41] and GA [3, 22, 30] are studied in the literature. The B&B algorithm is suitable to be parallelized given that the subtrees can be explored independently. The only shared information in the algorithm is the value of the best known solution (upper bound). Likewise, the parallelism is necessary to not only reduce the resolution time of GAs, but also to improve the quality of the provided solutions. The hybridization of these two categories permits to improve the performances of the total execution time.

Recently, some approaches [5, 6, 8, 12, 33] aiming at exploiting P2P/GRID computing and at deploying scientific applications requiring a great computing power, have been developed. [5, 6] are based on the *Master/Worker* paradigm [23, 38]. The main drawback of this approach is bottlenecks created on the master process because the inter-worker communications transit throw the master. Our work presents two parallel B&B and GA Algorithms and their hybridization based on master/worker paradigm with direct communications between workers. Therefore, bottlenecks are eliminated. We develop the peer-to-peer version using ProActive middleware which enables direct communications between the various peers (*workers*) of the network without flowing throw an intermediary (*master*). We applied the two algorithms and their combination version to mono-criterion permutation flow-shop problem *PFSP*. PFSP consists to find a schedule of a set of jobs on a set of machines that minimizes the completion time (makespan).

Jobs are scheduled in the same order on all machines and each machine can not be simultaneously assigned to two jobs.

The remainder of this chapter is structured as follows : Section 11.2 highlights the major points for the parallelization of a Branch-and-Bound algorithm and a brief description of parallel genetic algorithms. The concept of P2P Computing, ProActive middleware and the various tools that it offers to develop a distributed application on a peer-to-peer system are presented in Section 11.3. In Section 11.4, we present our parallelization of the Branch and Bound algorithm *ParallelBB* and genetic algorithm *ParallelGA* intended to be deployed on a large scale computing. In Section 11.5, its peer-to-peer implementation on top of ProActive middlware (namely *PHyGABaB*). Preliminary large scale deployment and performance evaluation on a P2P computing network formed and managed by ProActive showed in Section 11.6. We conclude this chapter in Section 11.7.

11.2 Parallel Combinatorial Optimization

Combinatorial optimization problems are often NP-hard, complex and CPU time-consuming. Exact methods and meta-heuristics are two major traditionally used approaches [7]. Exact techniques may be useful for solving small problem instances, but in realistic cases they are inefficient as they are extremely time-consuming. Conversely, meta-heuristics provide near-optimal solutions and allow to meet the resolution delays often imposed in the industrial field. The parallelization of these two categories is an efficient way to solve larger instances of problems in a reasonable time. In the following, we present parallelization of theses two categories as described in the literature.

11.2.1 Parallel Branch-and-Bound Algorithms

Branch-and-Bound algorithms are the most known techniques for an exact resolution of COPs. They make an implicit enumeration of the whole search space, because of the impossibility of a complete enumeration of all solutions of the search space due to the exponential growing of the potential solutions. B&B algorithms are characterized by four operations: *branching, bounding, selection and elimination*. In the first operation, the solution space of a given problem is partitioned into a number of smaller subsets on which the same optimization problem is defined. The bounding rule is used to compute the lower bound of the optimal solution of the considered problem. When a new solution (*upper bound*) is identified, it is compared to the actual lower bound in order to decide whether it is necessary to decompose the subproblem or not. The elimination rule uses these bounds to determine when further decomposition of a subproblem is unnecessary, so it identifies nodes which do not lead to the optimal solution and eliminates them. The subproblems are explored according to the selection rule. We can find the following exploration methods: *depth first search, breath first search, best bound,... etc*. A serial implementation of the algorithm consists of a sequential execution of these four operations.

The subtrees generated when executing a B&B algorithm can be explored independently, this makes the parallelization of these algorithms easier. The only global information in the algorithm is the value of the upper bound. Its parallelization may be attached to the architecture of the calculator machine, synchronization, granularity of generated tasks, communication between different processes and the number of computing processors. In the literature, several works on parallelization of B&B had been conducted [11, 13, 35, 39, 40, 41]. Geondron and Crainic [11] classified the parallelization strategies into three classes according to the degree of parallelization: *parallelism of type 1* introduces parallelism when performing the operations (generally the bounding operation) on generated subproblems (e.g. executing the bounding operation in parallel for each subproblem). This type of parallelism depends on the problem to be solved. In *parallelism of type 2* the search tree is built in parallel (e.g. processes work on several subproblems simultaneously). The *parallelism of type 3* also implies the building of several trees in parallel. The information generated when building one tree can be used for the construction of another. Thus the tree is explored concurrently.

The processes which participate in the computation of the parallel algorithm select their tasks from a *work pool*. A work pool is a memory where the processes select and store their work units (generated and not yet explored subproblems). Two types of work pool can be distinguished: *single work pool* and *multiple work pool*. Generally the first type is implemented on shared memory systems [13] and the second type uses several allocation memories. The first type is more adequate for the applications based on the *master/worker* [5] paradigm. Indeed, *master* process distributes part of computing (*tasks*) on a set of *workers* processes. When workers finishes their execution, the main process collects the obtained results. This paradigm is very used in scientific applications dedicated to be deployed on massively parallel systems (cluster, Grid computing). However, this paradigm presents a major drawback, it creates bottlenecks on the *master* process [4, 5].

11.2.2 Parallel Genetic Algorithms

Evolutionary Algorithms (EAs) [7] are stochastic search techniques and population-based algorithms. Genetic Algorithms (GAs), proposed by Holland [25] are the most known algorithms in this field [17]. They are powerful search techniques that are used successfully to solve problems in different disciplines. They are based on principles of natural selection and recombination. They attempt to find the optimal solution to the problem at hand by manipulating a population of candidate solutions. The population is evaluated and the best solutions are selected to reproduce and mate to form the next generation. Over a number of generations, good traits dominate the population, resulting in the improvement of the quality of the solutions.

Starting with a randomly generated population, a new generation is produced with three genetic operators: selection, reproduction and mutation. With the Selection operator we decide which individuals to survive. In the reproduc-

tion operator, two individuals are selected to produce a child which inherits its two parents. The mutation operator alters the genetic code of an individual to promote diversity (see [17] for more details on GAs).

In the literature, several works have been dedicated to parallel GAs (see [14, 21, 24, 32]) and classified them into two categories: parallelization of computation (fine grained) and parallelization of population (coarse-grained). In the first model, the operations commonly applied to each of the individuals are performed in parallel. The coarse-grained type is the most popular and used category. In this type, an initial population is divided into sub-populations which will evolve separately (in parallel) and exchange individuals following a migration protocol. They are usually implemented on distributed memory computers (MIMD) and based on the island model. The most recent example is the work of Mezmaz *et al* [33]. The authors proposed a P2P hybrid Genetic-Mimetic Algorithm based on the island model aiming at exploiting P2P/GRID-Computing.

11.3 P2P Computing and the ProActive Middleware

Distributed systems and applications are called Peer-to-Peer (P2P) if they employ distributed resources to perform a function in a decentralized manner. Here, resources includes (computer power, data storage and network bandwidth), the function concerns (distributed computing, data sharing, communication and collaboration) and decentralization (algorithms, data or both of them). That is the definition given in [34]. In distributed computing area, the idea is to exploit sparse computing resources (idle CPU cycles) and high performance can be obtained by using a large number of standard machines. *XtremWeb* [18], *SETI@Home* [8] and *ProActive* [1] are some examples of Peer-to-Peer middlewares dedicated to distributed computing. In this chapter we are interested in ProActive. It is a Java library which proposes an API, a graphical interface and parallel, distributed and concurrent programming tools [1]. A distributed application built with ProActive is composed of active objects *AO* [15]. An active object is a remote object having its own thread and receives calls on its public methods. Each *AO* has its own activity and the capability to decide in which order it will serve the method calls. The AOs are created on a support called Virtual Nodes VN. Association between a JVM and a VN is made by an XML deployment descriptor. ProActive is a SPMD (*Single Program Multiple Data*) middleware where a great number of interconnected nodes execute the same application operating on several distributed data. As the majority of P2P middlewares, designed to a distributed computing, the ProActive motivation is to use idle CPU cycles. A P2P network formed by ProActive, is a set of dynamic JVMs or VN which operates as a network of computing nodes. The concept of resource in ProActive is reduced to JVMs. Each JVM which wants to take part in calculation, launches a P2P Service which is a "*daemon*" executing on each node [16].

With ProActive, communications are done by remote method calls between AOs. It includes three types of communications: (1) *Synchronous calls:* the

method call is blocking, the execution is suspended until the arrival of the called method result; (2) *Asynchronous calls:* calls are not blocking and the execution of the program can continue without waiting for the result. An appointment ensures that the request arrives well at the called before continuing the activity. A future object is created waiting for any result. A future object represents the result of one of method call of this object which did not arrive yet; (3) *Single direction calls:* calls are not blocking (the appointment is always present), no result is awaited and no future is created.

The groups of communication[9] are another power tool provided in ProActive for distributed programming. A group of communication is the local representant of a set of objects distributed on interconnected machines. When a method is called upon a group, the execution environment sends an invocation request of the method on the group members, awaits one or more answers of the members according to the defined policy, and returns back the result to the caller. For more details on ProActive middleware see [9]).

11.4 Parallel B&B and GA for P2P Environment

11.4.1 ParallelBB

The Parallel B&B algorithm *"ParallelBB"* we developed is a high level parallelization algorithm, and belongs to *type 2* of the Gendron and Crainic classification. ParallelBB is developed with the *(Master/Worker)* paradigm with direct communication worker/worker and worker/master, to avoid the bottlenecks created on the master process. The master divides the initial problem into a set of subproblems *(tasks)*. Indeed, it builds reduced, independent and fine grained subproblems which can be treated in parallel by mono-processors. A single work pool is available on the master process which distributes the tasks among the workers. After this blocks waiting for the results of each one of them. In the following, we present principal operations of parallelBB.

Branching

The branching operation is performed serially by the master process. The master prepares an initial tree (Fig. 11.1) with a depth equal to K. Let n be the number tasks explored by the workers, N the initial size of the problem: $n \leq \prod_{0 \leq i \leq k} (N - i)$.

$T_1, T_2 \ldots T_n$, in the figure represent subtrees, each one contains a partial solution having a size equal to the current level in the tree. K is a parameter of ParallelBB which depends on two important factors: initially, it depends on the size of the considered problem, for example, the number of jobs in the case of a permutation Flow Shop Problem. The depth of the tree increases with the increasing of the number of jobs. Therefore, K must be sufficiently great to generate a large number of subtrees which will be treated in parallel by the workers. Thus, we allow to generate subproblems of a reasonable granularity to be performed by each worker. K also depends on (the size of the computing network).

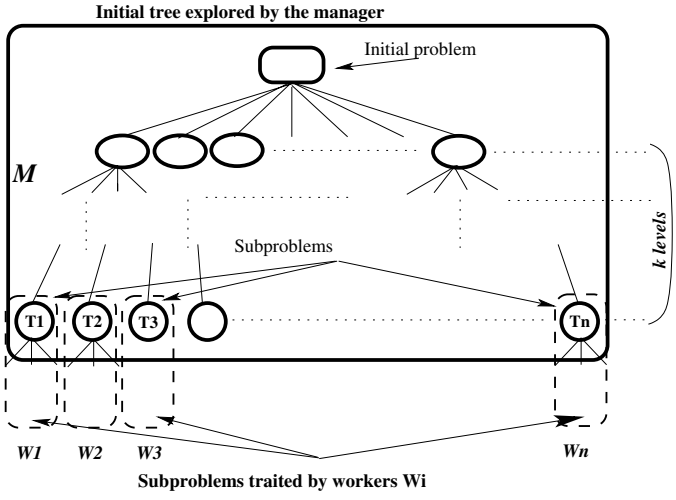

Fig. 11.1. General scheme of ParallelBB

In our case, K depends on the number of workers available in the network. If there is a reduced number of peers in the network, it is more interesting to have a reduced number of parallel tasks. Otherwise, we will loose more time in the communication between the workers and the distribution of the tasks. Among the roles of the master, the attribution of tasks (subtrees) to the workers. If the number of workers is greater than the number of tasks, the master considers only the workers which it needs. On the contrary, the master will make a redistribution of tasks to each new available[1] worker.

Selection and Elimination

The elimination operation is used only to eliminate subtrees having a lower bound greater than or equal to the upper bound. The policy of the tree exploration used by the master and the workers is different. The master explores the initial tree in width to build subtrees which will be explored in parallel. The master explores nodes by priority to the most promising nodes i.e. nodes having a lower bound less than or equal to the upper bound found until now, by all other workers. These subtrees are qtored in a priority-based queue. The workers explore their subtrees in depth and use *Best First Search* policy. They use a stack with opposite priority stacking of the subtrees nodes according to least promising ,i.e. at the top of the stack we find the most promising nodes. Thus, the workers start initially with promising nodes.

[1] A worker is said available if it accomplished its task or it finished the calculation which was assigned to it.

Communication and knowledge sharing

The global knowledge (all processes knowledge) related to the upper bound is increased and updated each time a given worker finds a new upper bound. This operation is performed by broadcasting the upper bound to all workers. The collaborative work between the workers, using the communication of the upper bound, allows us to gain much in computation time. Several branches can be eliminated, more quickly than in a traditional B&B (sequential B&B) before their exploration, quite simply by consulting the solution found so far. Unlike traditional B&B, where this same upper bound is known only when the exploration process reaches into the current node. By using this algorithm, a significant number of branches can be eliminated. These branches can't be cut in a sequential B&B because the upper bound making it possible can be found only in the future. This solution is situated in a search space which will be explored only later.

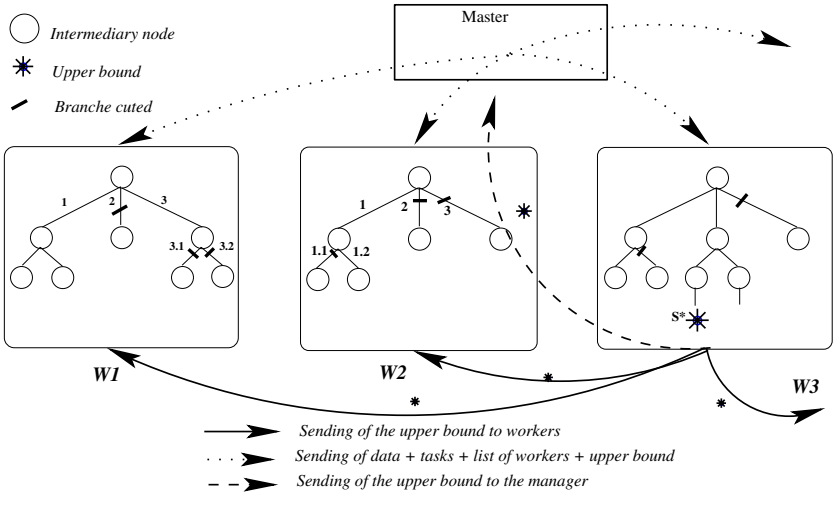

Fig. 11.2. Communications between processes of ParallelBB

In Fig. 11.2, the upper bound (solution S^*) was found by the worker W_3. This solution is in a future search space[2] compared to the search spaces of W_1 and W_2. When W_3 sends the upper bound S^* to W_1 and W_2, it allows then to eliminate the branches: (2 and 3.1) in the subtree of W_1 and (1.1, 2 and 3) in the subtree of W_2.

The master increases also the workers knowledge concerning all other workers executing in the system (dynamic management). The various types of communication can be summarized as follows (see Fig. 11.2) :

[2] In a serial execution, S^* will not be found before exploring all subtrees belonging to the search space of W_1 and W_2.

- *The Master to a Worker:* (1) Sending of the task to perform (data of the problem and the subtree to explore). (2) Sending of the pool of executing workers. This knowledge allows each worker to know its environment concerning other workers in progress for a collaborative work. (3) Initialization of the global upper bound. This knowledge allows each worker to eliminate branches from the beginning in its search space.
- *A Worker to the Master:* (1) Sending of the final solution obtained by the worker (if the worker finishes the exploration of the subtree). (2) Sending of the upper bound which is better than the global current knowledge of the algorithm. this allows the master to improve the knowledge of the future workers with this upper bound.
- *A Worker to a Worker:* Each worker sends the upper bound to all the workers in its communication window (workers in progress) so that these workers will be able to reduce the search space by eliminating a great number of branches. The communication window of a worker is reduced to its neighbors i.e. a worker communicates only with the workers in progress (its neighbors).

11.4.2 ParallelGA

In this section, we present briefly the parallel genetic algorithm we developed *ParallelGA*. As ParallelBB, ParallelGA is a master/worker-based algorithm with direct communications between workers. The master process divides the initial population into subpopulations with a reduced size. The exploration of these subpopulations will be considered as parallel tasks and can be handled by a single processor. All sub-populations will evolve in parallel by the available workers, each worker executes its instance of the algorithm. The master redistributes not handled subpopulation each time a worker terminates its part of calculation. The size of initial generated population must be sufficiently great to increase the search space and then increase chances to reach acceptable solution. The master fixes the size of sub-populations according to the number of available workers and the power of processors.

Like in ParallelBB, communications are very important in the case of a ParallelGA. The workers communicate their best individuals to their neighbors (see Fig. 11.3). This migration of individuals allows us to prevent convergence and to prevent workers to turn in locals minimum. After a fixed number of generations (or after a fixed time interval), the migration of elite individuals occurs.

The different types of communications of ParallelGA can be summarized as follows:

- *Master to Workers:* The master communicates the task to perform (data of the problem, subpopulation and different parameters of the GA) as well as the pool of executing workers.
- *A worker to the master:* the only information that a worker sends to the master is the final result when it terminates its part of computation.
- *a worker to a worker:* The communication between workers consists in the exchange of individuals (see Fig. 11.3).

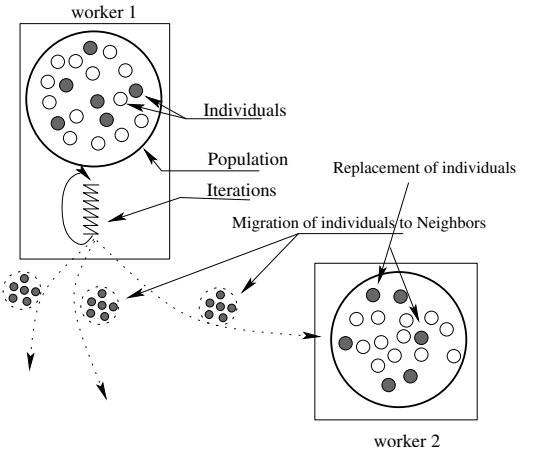

Fig. 11.3. Migration of individuals

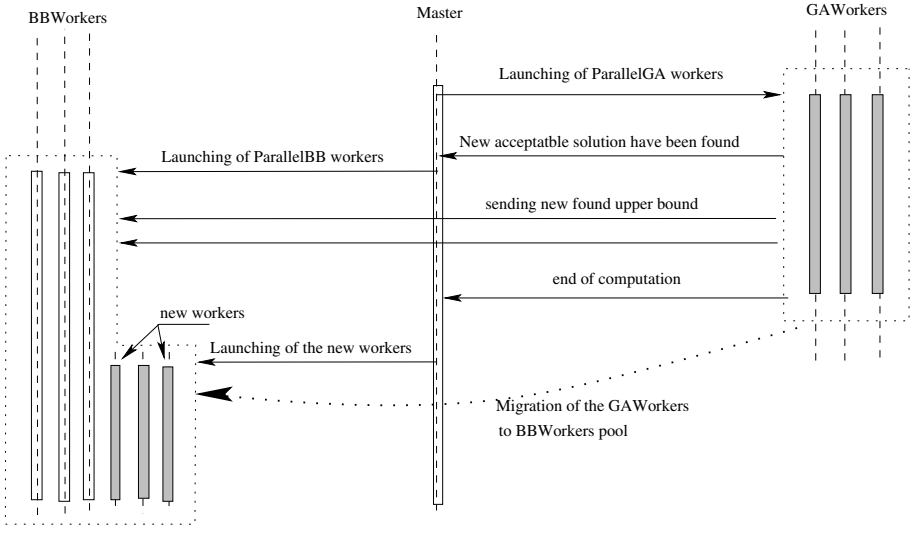

Fig. 11.4. Behavior of the different workers

11.4.3 Hybridization

A quick execution of an exact algorithm like *ParallelBB* needs to start computation with a near-optimal upper bound that we can obtain by *ParallelGA*. As shown in (Fig. 11.4), we launch first *ParallelGA* to obtain an acceptable initial value of the upper bound which is passed to *ParallelBB*. We use two types of workers: workers participating in the exploration of the B&B search space

BBWorkers and those participating in the exploration of the GA population *GAWorkers*. *ParallelBB* starts computation before the termination of *ParallelGA* and run in parallel. *BBWorkers* receive the value of the upper bounds from *GAWorkers* each time new ones have been found. At the end of exploration of all *ParallelGA*'s subpopulations, the concerned *GAWorkers* migrate and join the exploration of *ParallelBB* search tree process, thus they take a *BBWorkers* behavior.

11.5 Peer-to-Peer Implementation on Top of ProActive

The implementation of *ParallelBB* and *ParallelGA* on *ProActive* gave rise to our Active Application[3] *PHyGABaB*. This application is based on two entities (active objects *AOs*): *P2PWorker* and *P2PMaster*. ProActive provides active nodes *ANs* (JVMs), recovered on the whole of the network, which are ready to receive calculation. These ANs are *P2PWorkers* receiving tasks (subtrees or subpopulation to explore). In the case of a static grid managed by ProActive, the P2P services *P2PService* are already launched, i.e., each host shares its JVM and at least one *P2PWorker* which can receive AOs. An *AN* can receive one or more *AOs*. When the *P2PMaster* is created on the local JVM, it consults an XML deployment descriptor where *P2PMaster* will find ANs. At the end of this stage, *P2PMaster* will have a list of *P2PWorkers* ready to receive calculation. When such nodes are ready, they can directly receive computational work coming from the *P2PMaster*, new active objects will be launched otherwise. In this case, the XML descriptor must be modified so that it can deal with the dynamic nature of a P2P network and that receive new peers which arrive into the initial network (see Section. 11.5.3 for the handling of new arrivals).

11.5.1 Distribution of the Computation among Workers

After initializing the workers, *P2PMaster* generates a set of independent tasks (subtrees and/or subpopulations). These tasks are represented by passive objects (see Fig. 11.5). Before the *P2PMaster* sends a task to a *P2PWorker*, it increases the knowledge of each *P2PWorker* concerning its environment. This knowledge concerns the set of workers executing other tasks, the best solution found so far (when the worker participates in the branch and bound tree exploration) or only the initial subpopulation (when the worker participates in the exploration of the GA population).

Each time a task is assigned to a *P2PWorker*, a future object is created and added to the future list (*futureList*)of *P2PMaster*. *P2PMaster* waits all the future objects coming from the *P2PWorkers* appearing in its list. The future *P2PWorker* represents the result of the calculation of its task that the *P2PMaster* assigned to it. This is accomplished by listening its response (the

[3] An Active Application is an application based on active objects. Any application developed using the ProActive middleware must be Active so that it can be deployed on the Computing Network.

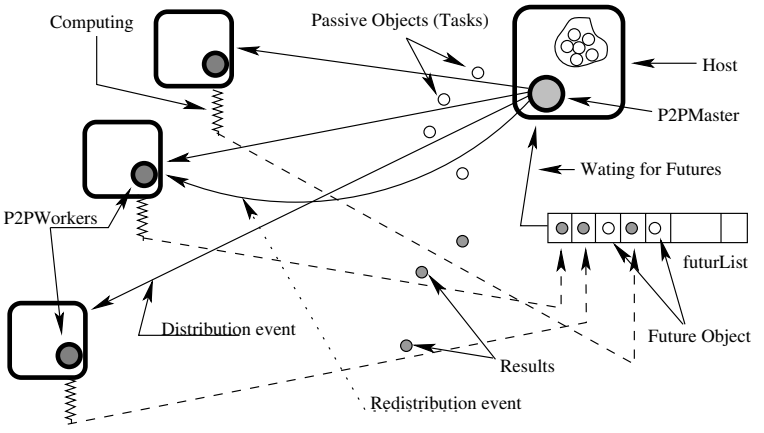

Fig. 11.5. Tasks distribution on workers

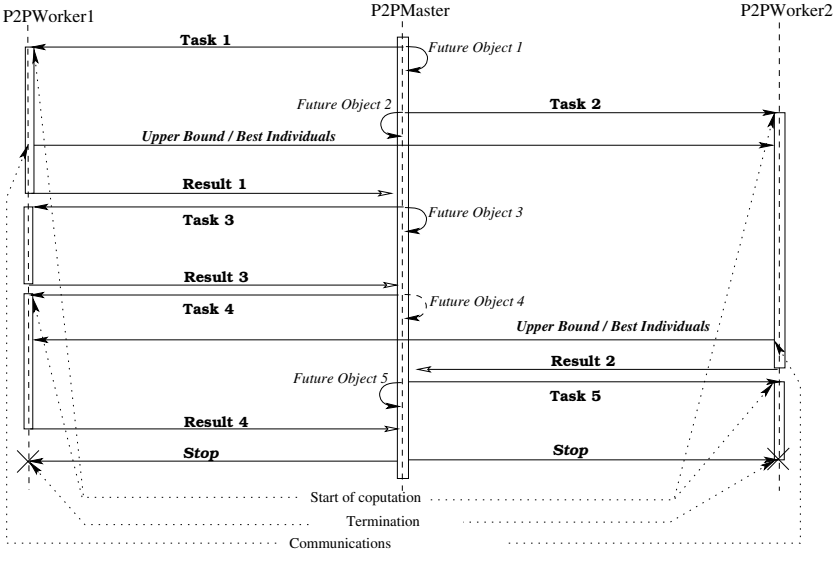

Fig. 11.6. Sequence diagram of tasks distribution

computation result) in the future. The listening is of type *wait for any event* made by the method *waitForAny(futureList)* and is accomplished by waiting for any event coming from *P2PWorkers* appearing in the list $futureList$. The event is started with each termination of a task treatment. After that the *P2PMaster* recovers the result produced by the future *P2PWorker*, it creates and reallocates a new passive object to it.

The chronology of events presented in (Fig. 11.5) is not really true because all operations are in asynchronous mode i.e. an event of type redistribution can arrive before other events of type distribution when one of *P2PWorkers* returns back result before *P2PMaster* finishes distributing of all tasks. We can see well on (Fig. 11.6) the sequence diagram of the chronology of all events of the tasks distribution.

11.5.2 Communications

Communication between different components (*P2PMaster* and *P2PWorkers*) is very important for its good functioning. We have seen previously that the *workers* communicate frequently to ensure the freshness of the upper bound and the migration of individuals. The use of classic communication between *P2PWorkers*, i.e., by sending one message for each *P2PWorker* is not efficient in this type of application where the communication cost is very high. If we use the classical communication we will need to broadcast the same message to all *P2PWorkers* in our computing pool. This procedure requires the traversal of the whole *P2PWorkers* list (thousands or millions), in other words, each *P2PWorker* must have one copy of all *P2PWorkers* taking part in the computation. This solution is not interesting because of huge amount of time required by the traversal of the entire list and the memory space necessary to store requiring when saving the set of *P2PWorkers* while this space is to be minimized.

With ProActive, we opted for the communication groups with single direction, non blocking and asynchronous methods invocation. We created *BBWorkerGroup* and *GAWorkerGroup* which are the two local representants of a set of *P2PWorker* recipients of a message. *BBWorkerGroup* represents all *P2PWorkers* participating in ParallelBB computation and *GAWorkerGroup* represents *P2Pworkers* participating in the exploration of subpopulations in the ParallelGA. When a *P2PWorker* wants to send a message to its colleagues, it passes by these two groups, which implement the same communication method which is even implemented on all the *P2PWorkers*. We developed two communication methods: *shareBestValue* and *shareSubPopulation*allowing the workers to share respectively the best value of the current solution (upper bound) and their selected individuals (elite). A *P2PWorker* calls these two communication methods in order to share their upper bound or subpopulation. Thus *BBWorkerGroup* and *GAWorkerGroup* call this same method on the set of *P2PWorkers* they represent.

11.5.3 New Arrivals (New Peers)

The dynamic availability of peers is one of the P2P networks characteristics where the resources (here JVMs) join and leave frequently the system. Each JVM is at the same time client and server for other JVMs. The ProActive middleware manages the new coming peers in the system by a listener implemented with the method (*nodeCreated*). *P2PMaster* implements this interface which listens for eventually peers connections in the network. A P2P daemon is launched on

each machine participating in the computation. When a new node is connected, an AN is created there and one *P2PWorker* is established to receive the computation units. *P2PMaster* decides the affiliation of this new peer, indeed, it will behave as a *BBWorker* or as *GAWorker*. *P2PMaster* adds this new *P2PWorker* to the *P2PWorkers* set list and to the corresponding group of communication (*BBWorkerGroup* or *GAWorkerGroup*). After that, if this new *P2PWorker* belongs to *BBWorkerGroup* it could be informed of the global upper bound. Other *P2PWorkers* will be able to have an idea on the progress and the solutions obtained by this new *P2PWorker*. Whatever the affiliation of this new peer, the *P2PMaster* sends to it its task and will behave as other *P2PWorkers*. A new peer, arriving at the computational network, adheres to the group of communication. The peers forming the old group of communication update their group by adding this new peer. This operation is managed by *P2PMaster* which sends to all the group of communication members the new configuration of the group, i.e., the adding of this new peer. This operation allows new *BBWorkerGroup* members to avoid the exploration of subtrees unnecessarily, this allows reduce execution time. To obtain the real global upper bound, the new peer selects randomly a peer and then sends its initial upper bound. If the upper bound of this selected peer is inferior to the received one, it proceeds to its correction by broadcasting the real global upper bound.

11.5.4 Fault Tolerance

The peers failure is taken into account by both ProActive (middleware-level) and our application (application-level). With ProActive we create tow types of servers: *Resource server* and *Fault tolerance servers*.

The *resource server* returns a free node that can host the recovered AO, this server can store free nodes by two different ways:

- At deployment time: the user can specify in the deployment descriptor a resource virtual node. Each node mapped on this virtual node will automatically register itself as free node at the specified resource server.
- At execution time: the resource server can use an underlying P2P network (see [1]) to reclaim free nodes when a hosting node is needed.

The *fault tolerance servers* are used for checkpointing operations, the localization of AOs, and the failure detection.

In our application, when a peer disconnects, the *P2PMaster* sends its part of calculation to one or more other available peer(s) and recover(s) only the first returned solution of the same task and ignores other results representing the same task. This process is performed at the end of the computation of all tasks.

11.6 Large Scale Deployment and Performance Evaluation

In the following, we present the different experiments and obtained results of the exact algorithm *ParallelBB* hybridized with the heuristic *ParallelGA* applied to

the permutation flow-shop problem (PFSP) which is a reference problem in the given its importance in many industrial areas.

11.6.1 PFSP Formulation

A Permutation Flow-Shop Problem (PFSP) is a scheduling in which all tasks of all jobs are scheduled on all machines in the same order. The execution of a job J_i on the machine M_k is called operation $O_{i,k}$ and its execution time will be noted $p_{i,k}$, $t_{i,k}$ represents its starting date and $c_{i,k}$ its release time. We designate also by $r_{i,k} = \sum_{l<k} p_{i,l}$ the early instant in which the job J_i can start its operation on M_k and by $q_{i,k} = \sum_{l>k} p_{i,l}$ the latency duration (minimal time selling between the end of J_i on M_k and the end of the total scheduling).

11.6.2 Modeling and Lower Bound Calculation

We applied our algorithm to the PFSP and we considered the total completion time *Makespan* (C_{Max}) cost function. In ParallelGA, an individual (permutation) is considered as a vector of jobs. The root of the tree generated by ParallelBB represents a configuration where no task is assigned to any machine. A node with depth n will have a configuration with n assigned tasks.

The effectiveness of B&B algorithms resides in the use of a good estimation of the optimal solution. M. R. Garey, D. S. Johnson and R. Sethi (Garey and al., 1976) proved that the PFSP problem becomes NP-hard beyond 3 machines. The calculation of the lower bound for a PFS problem is based on two results. The first one is found by Johnson [27] (*rule of Johnson*). A transitive rule $\preceq$ is defined as follows:

$$J_i \preceq J_j \Leftrightarrow min(p_{i,1}; p_{j,2}) \leq min(p_{i,2}; p_{j,1}) \tag{11.1}$$

If $J_i \preceq J_j$, then there exists an optimal scheduling for a FSP (P) in which the job J_i precedes the job J_j[27]. Thus, the PFS problem with two machines $F2||C_{max}$ can be solved in $O(nlogn)$ [31]. The optimal solution is obtained by sorting the jobs having the execution times on the first machine shorter than the second in the ascending order. Then, sort jobs having their execution time on the second machine shorter than on the first one in the descending order. This result was extended by Jackson [28] and Mitten [29] for the resolution of a two machines PFS problem with lags $F2|l_j, permut|C_{max}$ where each job has a *lag* l_j which represents the minimal duration between $t_{j,2}$ and $c_{j,1}$. They demonstrated that the optimal solution of this problem is obtained using the Johnson to the values $p_{i,1} + l_i$ for the jobs on the 1^{st} machine and $l_i + p_{i,2}$ on the 2^{nd} one.

$$J_i \preceq J_j \Leftrightarrow min(p_{i,1} + l_i; l_j + p_{j,2}) \leq min(l_i + p_{i,2}; p_{j,1} + l_j) \tag{11.2}$$

The most known lower bound for the PFS problem with m machines is the bound proposed by Lageweg et al. [26]. They used the optimal solutions values

for all 2 machines subproblems with lags. Given two machines M_k and M_l (with c $k < l$), it is indeed possible to extract such problem posing:

$$p_{j,1} = p_{j,k}; l_j = \sum_{k<\mu<l} p_{j,\mu}; p_{j,2} = p_{j,l} \tag{11.3}$$

In practice, we consider all couples of machines M_k et M_l (with $k < l$) and we extract for each couple a PFS with two machines lags substituting the values $p_{i,1}$ by $p_{i,1}+l_i$ and $p_{i,2}$ by $l_i+p_{i,2}$. We notate $P^*_{Ja}(\jmath, M_k, M_l)$ the Jackson-Mitten optimal solution of the obtained subproblem considering the set of jobs $\jmath$ and machines M_k and M_l. B.J. Lageweg et al obtained thus the lower bound (with $O(m^2nlogn)$ complexity) which we used in our work :

$$LB(\jmath) = \max_{1\leq k<l\leq m} \{P^*_{Ja}(\jmath, M_k, M_l) + \min_{(i,j)\in\jmath^2, i\neq j} (r_{i,k} + q_{j,l})\} \tag{11.4}$$

11.6.3 Experiments

The studied problem instances are those of E.Taillard [37]. We treated the benchmarks: *ta_20_5_2*, *ta_20_5_3*, *ta_20_10_1*, *ta_20_10_2* and *ta_100_5_1*[4]. Parameters of *ParallelGA* are fixed as follows: 500 individuals in the population, the size of each subpopulation is fixed to 20, migrations occur every 10 generations with 10 migrants.

Table 11.1. Experimentation hardware platform

Site	CPU characteristic x number of CPU / node	Number of nodes	Number CPUs
Lille	AMD Opteron 248, 2.2 GHz x 2	70	140
Lyon	AMD Opteron 246, 2.0 GHz x 2	55	110
Nancy	AMD Opteron 246, 2.0 GHz x 2	35	70
Orsay	AMD Opteron 246, 2.0 GHz x 2	290	580
Rennes	Intel Xeon 5148 LV, 2.33 GHz x 2	60	120
	AMD Opteron 246, 2.0 GHz x 2	90	180
	AMD Opteron 248, 2.2 GHz x 2	50	100
Nice	AMD Opteron 246, 2.0 GHz x 2	55	110
	AMD Opteron 275, 2.2 GHz x 2	50	100
Total		775	**1510**

We made large scale deployment of the application (more than 1500 processors) gathered on six geographically distributed sites located at (*Lille, Rennes, Orsay, Nice, Lyon and Nancy*) belonging to the French grid GRID'5000 [2]. The experimentation hardware platform characteristics are presented in Table 11.1. As we said previously, our objective is the development of hybrid algorithm

[4] *ta_i_j_k*: a Taillard benchmark with i: number of jobs, j: number of machines and k: the instance number.

Table 11.2. Some obtained execution times

Number of Processors	Deployment Time	$ta\_20\_5\_2$	$ta\_20\_5\_3$	$ta\_20\_10\_1$	$ta\_20\_10\_2$	$ta\_100\_5\_1$
06 (1/5)	15	(3) 4	(492) 401	(1815) 1287	(277) 234	-
20 (5/15)	46	(10) 8	(409) 391	(170) 129	(111) 98	-
50 (15/35)	112	(16) 11	(277) 263	(100) 97	(59) 51	-
100(20/80)	234	17	193	81	50	-
200(40/160)	504	-	151	77	-	-
300(60/240)	713	-	152	-	-	7h [5572]
600(100/400)	1949	-	-	-	-	6h 57min [5571]
1500(300/1200)	4186	-	-	-	-	6h 57min [5571]

to solve exactly complex instances of benchmarks with large scale deployment. Here, we made this preliminary deployment just to show that the application is scalable and can be used in this sense.

We made other deployments of our application on 6, 20, 100, 200 and 300 processors on GRID'5000. In this experiments a portion of workers are assigned to *ParallelGA* computation and the rest to *ParallelBB*. As shown in Section 11.4.3, when *GAWorkers* terminate their calculation parts they join *BBWorkers*, so after they terminate, *ParallelBB* exploits the whole pool of workers. The application was launched in P2P mode where all processes run with the lowest priority to reach one of P2P Computing characteristic which is the exploitation of idle CPU cycles. Table 11.2 shows the execution times obtained by our hybrid application compared to an older version of P2P non hybrid parallel branch and bound algorithm [10]. The old times are presented in the table in parentesis. In the first column we have the number of used processors (i/j): i number of $GAWorkers$ and j number of $BBWorkers$.

The first point we notice is that the hybridization improves efficiency. Comparing with the non hybrid method, practically, all benchmarks are solved more efficiently when using hybridization. For example, the exact resolution of the benchmark $ta\_20\_10\_01$ on 20 machines took 129 seconds using the hybrid algorithm whereas it was solved in 170 using non hybrid version. This same benchmark was solved in 97 seconds whereas it took 100 sec. The only exception is when solving $ta\_20\_05\_02$ it was solved three times more quickly on 6 machines than on 20 and 5 times more efficient more than on 50. This can be explained if we take a look on the situation of the solution regarding the space of solutions. It was found in the 3^{rd} node of the solutions tree, this means that the six machines was sufficient to find the solution in short time, and the 50 machines take an additional time to manage tasks and free all workers deployed.

In the second column we have deployment times (deployment time includes, detection and handling of nodes and distribution of tasks). For small instances of

benchmarks, we don't need a large scale deployment, $ta\_20\_5\_2$ was exactly solved on 50 machines in 11 seconds whereas the deployment time is 112 seconds. This is not the case for large instances where the deployment time is insignificant compared to time of resolution. Though, instance $ta_{100\_5\_1}$ wasn't exactly solved, but we remark easily that this time is negligible. The calculation of these two instances wasn't terminated, values in the table represent the time of calculation and reached upper bound between square brackets.

11.7 Conclusions

Using exact methods for the resolution of COPs, such as B&B which is the most used for an exact resolution of these problems, is very important. However, their use on applications of industrial size is possible only by the use of a great computational power. Hybridization and large scale parallelism based on the use of Grid Computing or P2P Computing proves today a potential tool which offers such power. Several factors have to be taken into account for a better parallelization of these methods, for their implementations on a Peer-to-Peer systems such as ProActive and for a better exploitation of the computing power. (1) A study and a good choice of a suitable model of parallelism; (2) A good management of the knowledge generated by these algorithms; (3) Exploitation of all the tools that P2P middlewares offers for controlling of the computational network.

In this chapter, we developed a parallel branch-and-bound algorithm hybridized with a parallel genetic algorithm for resolution of COPs on a Peer-to-Peer system. We applied it to the Permutation Flow-Shop problem which is a reference problem in this area. We chose a high level parallelism of branch-and-bound and a coarse grained parallel genetic algorithm. In this direction, we developed *ParallelBB*, *ParallelGA* and a hybrid version based on *master/worker* paradigm which is a most appropriate technique for the development of scientific applications dedicated to an intensive computing on large scale systems. The performances of the algorithm were improved with the knowledge sharing between the workers. This was realized by the use of the *master/worker* paradigm with direct communications between workers.

We implemented the peer-to-peer version of our algorithms on top of ProActive and we took benefits from the maximum of its functionalities. We took advantage of the communication groups and the asynchronous methods invocation in single direction for the knowledge sharing between workers and master. We used the listeners and daemons in order to take into account the new arrivals, their detection and the management of their connections. Finally, we used the future active objects for collecting of computation results. The experiments made on a P2P network managed by ProActive showed the interest of collaborative work between nodes of the computation network as well as the importance of hybridization. We have shown the ability of our application to support scalability and dynamic availability of peers in the network.

As a perspective, we project to extend our work to other type of COPs (Quadratic Assignment Problems QAP and Quadratic three dimensional Assignment Problems Q3AP). In addition, we plan to improve the performances of ParallelBB with: (1) The load balancing of the tasks generated by the algorithm so that they become more equitable; (2) The production of several forms of granularity of tasks and their distribution to the corresponding station (Calculator, PC, laptop...).

References

1. http://proactive.objectweb.org
2. http://www.grid5000.fr
3. Adamidis, P.: Review of parallel genetic algorithms bibliography. Technical report, Aristotle University of Thessaloniki, Thessaloniki, Greece (1994)
4. Aida, K., Futakata, Y., Hara, S.: High-performance parallel and distributed computing for the bmi eigenvalue problem. In: Proc. the 16th IEEE International Parallel and Distributed Processing Symposium, pp. 71–78 (2002)
5. Aida, K., Natsume, W., Futakata, Y.: Distributed Computing with Hierarchical Mast-Worker Paradigm for Parallel Branch-and-Bound Algorithm. In: IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), p. 156 (2003)
6. Aida, K., Osumi, T.: A case Study in Running a Parallel Branch and Bound Application on the Grid. In: IEEE, Symposium on Applications and the Internet (SAINT 2005), January 2005, pp. 164–173 (2005)
7. Alba, E., Luque, G., Talbi, E.-G., Melab, N.: Meta-heuristics and parallelism. In: Parallel Meta-heuristics, pp. 79–104. John Wiley & Sons, Chichester (2005)
8. Anderson, D.P., Cobb, J., Corpela, E., Lepofsky, M., Werthimer, D.: SETI@Home: An Experiment in Public-Resource Computing. Communications of the ACM 45(11), 56–61 (2002)
9. Baduel, L., Baude, F., Caromel, D., Contes, A., Huet, F., Morel, M., Quilici, R.: Grid Computing: Software Environments and Tools. In: Programming, Deploying, Composing, for the Grid. Springer, Heidelberg (2006)
10. Bendjoudi, A., Melab, N., Talbi, E.-G.: A Parallel P2P Branch-and-Bound Algorithm for Computational Grids. In: Seventh International Workshop on Global and Peer-to-Peer Computing. IEEE/ACM International Symposium on Cluster Computing and the Grid (IEEE/ACM CCGRID), May 14-17, pp. 749–754 (2007)
11. Gendron, B., Crainic, T.G.: Parallel Branch-and-Bound Algorithms: Survey and Synthesis. Operations Research 42(06), 1042–1066 (1994)
12. Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lantéri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.-G., Touche, I.: Grid 5000: a large scale and highly reconfigurable experimental Grid testbed. Intl. Journal of High Performance Computing Applications 20(4), 481–494 (2004)
13. Bourbeaua, B., Crainica, T.G., Gendron, B.: Branch-and-bound parallelization strategies applied to a depot location and container fleet management problem. Parallel Computing (26), 27–46 (2000)
14. Cahon, S., Melab, N., Talbi, E.-G.: ParadisEO: A Framework for the reusable Design of Parallel and Distributed Meta-heuristics. Journal of Heuristics 10(3), 357–380 (2004)

15. Caromel, D., Klauser, W., Vayssière, J.: Towards seamless computing and metacomputing in Java. In: Fox, G.C. (ed.) Concurency Practice and Experience, vol. 10, pp. 1034–1061. Wiley & Sons, Ltd., Chichester (1998)
16. Mathieu, C., Caromel, D., di-Costanzo, A.: Peer-to-peer for computational grids: Mixing clusters and desktop machines. Parallel Computing Journal on Large Scale Grid (to appear, 2007)
17. Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York (1991) BU: 511.6 HAN
18. Fedak, G., Germain, C., Néri, V., Cappello, F.: XtremWeb: A Generic Global Computing System. In: Proceedings of the first International Symposium on Cluster Computing and the Grid (CCGRID 2001), p. 582. IEEE, Los Alamitos (2001)
19. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
20. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman & Co., New York (1979)
21. Goodman, E.D.: An Introduction to GALOPPS v3.2. Technical report, 96-07-01, CARAGE, I.S. Lab. Dpt. of C.S and C.C.C.A.E.M., Michigan State University, East Lansing, MI (1996)
22. Gordon, V.S., Whitley, D.: Serial and Parallel Genetic Algorithms as Function optimizers. In: Forrest, S. (ed.) Proceedings of the Fifth International Conference on Gentic Algorithms, pp. 177–183. Morgan Kaufmann, San Francisco (1993)
23. Goux, J., Kulkami, S., Linderoth, J., Yoder, M.: An enabling framework for master-worker applications on the computational grid. In: IEEE Symposium and High Performance Distributed Computing (HPDC 2000), vol. 9, p. 43 (August 2000)
24. Herrera, F., Lozano, M.: Gradual distributed real-codeed genetic algorithm. IEEE Transaction in Evolutionary computation 4, 43–63 (2000)
25. Holland, J.H.: Adapation in natural and artificial systems. The university of Michigan Press, Ann Arbor (1975)
26. Lenstra, J.K., Lageweg, B.J., Rinnooy Kan, A.H.G.: A General boundind scheme for the permutation flow-shop problem. Operations Research 26(1), 53–67 (1978)
27. Johnson, S.M.: Optimal two and three-stage production schedules with setup times included. Naval Research Logistis Quarterly 1, 61–68 (1954)
28. Jackson, J.R.: An Extension of Johnson's results on Job-Lot Scheduling. aval Research Logistis Quarterly 3(3) (1956)
29. Mitten, L.G.: Sequencing n jobs on two machines with arbitrary time lags. Management Science (1959)
30. Lin, S.-C., Punch, W., Goodman, E.: Coarse-Grain Parallel Genetic Algorithms: Categorization and New Approach. In: IEEE Symposium on Parallel and distributed Processing, pp. 28–37. IEEE computer Society Press, Los Alamitos (1994)
31. Péridy, L., Pinson, E., Rivreau, D.: Elimination Rules for the Flow-Shop and Permutation Flow-Shop Problems. Technical report, Institut de Mathématiques Appliquées, Université Catholique de l'Ouest (May 27, 2002)
32. Mejia-Olvera, M., Cantu-Paz, E.: DGENESIS-software for the execution of destributed genetic algorithms. In: Proceedings XX Conference Latinoamerica de Informatica, pp. 935–946 (1994)
33. Melab, N., Talbi, E.-G., Mezmaz, M., Wei, B.: Parallel Hybrid Multi-objective Meta-heuristics on P2P Systems. In: Olaru, S., Zomaya, A.Y. (eds.) Handbook of Bioinspired Computing, pp. 649–663. CRC Press, Boca Raton (2006)

34. Milojicic, D., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., Xu, Z.: HP Laboratories Palo Alto. Peer-to-Peer Computing. Technical Report HPL-2002-57, HP (March 8, 2002)
35. Mitten, L.G.: Branch-and-bound Methods: General formulation and properties. Operations Research 18, 24–34 (1970)
36. Schollmeier, R.: A definition of Peer-to-Peer networking for the classification of Peer-to-Peer architectures and applications. In: 2001 International Conference on Peer-to-Peer Computing (P2P 2001), Linko pings Universitet, Sweden. IEEE, Los Alamitos (2001)
37. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operations Research 64, 278–285 (1993)
38. Tanaka, Y., Sato, M., Hirano, M., Nakada, H., Sekiguchi, S.: Performance evaluation of firewall compliant globus-based wide-area cluster system. In: IEEE Symposium on High-Performace Distributed Computing (HPDC), vol. 9, p. 121 (2000)
39. Trienekens, H.W.J.M.: Parallel Branch and Bound on an MIMD System. Report 8640/A, Econometric Institute, Erasmus University Rotterdam (1989)
40. Trienekens, H.W.J.M., Bruin, A.: Towards a Taxonomy of Parallel Branch and Bound Algorithms. Report EUR-CS-92-01, Dept. of Computer Science, Erasmus University, Rotterdam (1992)
41. Yang, M.K., Das, C.R.: A Parallel Branch-and-Bound Algorithm for MIN-Based Multiprocessors, pp. 222–223. ACM, New York (1991)