Parallel cooperative meta-heuristics on the computational grid. A case study: the bi-objective Flow-Shop problem

N. Melab *, M. Mezmaz, E.-G. Talbi

Laboratoire d’Informatique Fondamentale de Lille, UMR CNRS 8022, INRIA Futurs, Dolphin-Cité scientifique-59655, Villeneuve d’Ascq cedex, France

Received 20 January 2005; received in revised form 1 December 2005; accepted 13 January 2006
Available online 11 October 2006

Abstract

In this paper, we contribute with the first results on parallel cooperative multi-objective meta-heuristics on computational grids. We particularly focus on the island model and the multi-start model and their cooperation. We propose a checkpointing-based approach to deal with the fault tolerance issue of the island model. Nowadays, existing Dispatcher–Worker grid middlewares are inadequate for the deployment of parallel cooperative applications. Indeed, these need to be extended with a software layer to support the cooperation. Therefore, we propose a Linda-like cooperation model and its implementation on top of XtremWeb. This middleware is then used to develop a parallel meta-heuristic applied to a bi-objective Flow-Shop problem using the two models. The work has been experimented on a multi-domain education network of 321 heterogeneous Linux PCs. The preliminary results, obtained after more than 10 days, demonstrate that the use of grid computing allows to fully exploit effectively different parallel models and their combination for solving large-size problem instances. An improvement of the effectiveness by over 60% is realized compared to serial meta-heuristic.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Computational grids; Multi-objective meta-heuristics; Parallelism; Cooperation; Flow-Shop problem

1. Introduction

In combinatorial optimization, meta-heuristics allow to iteratively solve in a reasonable time NP-hard complex problems. According to the number of solutions handled at each iteration, two main categories of meta-heuristics are often distinguished: evolutionary algorithms (EAs) and local searches (LSs). EAs are population-oriented as they manage a whole population of individuals, what confers them a good exploration power. Indeed, they allow to explore a large number of promising regions in the search space. On the contrary, LSs work with a single solution which is iteratively improved by exploring its neighborhood in the solution space. Therefore, they are characterized by better local intensification capabilities. On the other hand, theoretical and experimental studies have shown that the cooperation between meta-heuristics belonging either...
to the same category or to different categories improves the effectiveness (quality of provided solutions) and
the robustness of the meta-heuristics [14]. Nevertheless, as it is generally CPU time-consuming it is not often
fully exploited in practice. Indeed, experiments with cooperative meta-heuristics are often stopped without
convergence being reached. Nowadays, grid computing [7] is recognized as a powerful way to achieve high
performance on long-running scientific applications. Parallel cooperative meta-heuristics for solving real-
world multi-objective problems (MOPs) are good challenges for grid computing. However, to the best of
our knowledge very few research works have been published on that topic.

In this paper, we contribute with the first results on parallel cooperative multi-objective meta-heuristics on
computational grids. We particularly focus here on the island model and the multi-start model and their coop-
eration. These two models are presented in Section 2. The computational grid targeted in this paper is a scal-
able pool of heterogeneous and dynamic resources geographically distributed across multiple administrative
domains and owned by different organizations. The scalability and volatile nature of the grid may particularly
have a great impact on the design of grid-based meta-heuristics. The traditional parallel models and cooper-
ination mechanisms have to be re-visited and adapted to be scaled up. Moreover, these require to be fault-tol-
ent to allow long-running problem resolutions.

The design and deployment of the parallel cooperative models of meta-heuristics on computational grids
require a grid middleware that allows cooperation between parallel tasks. In this paper, we focus on the Dis-
patcher–Worker grid middlewares such as XtremWeb [6]. One major limitation of such middlewares is that
they do not allow communication between workers. One of our contributions is to propose a Linda-like
[10] coordination model and its implementation on top of XtremWeb. This is a Dispatcher–Worker middle-
ware, in which the dispatcher distributes application tasks submitted by clients to volunteer workers at their
request. In addition, the middleware provides fault-tolerance mechanisms that are costly in a highly volatile
computational grid. Indeed, a work unit is re-started from scratch each time it fails. Another contribution of
this paper is to deal with the fault-tolerance issue at the application level. We propose a checkpointing
approach for the island model.

To be validated the work has been experimented on the Bi-criterion Permutation Flow-Shop Problem
(BPFSP) [16]. The problem consists roughly to find a schedule of a set of jobs on a set of machines that min-
imizes the makespan and the total tardiness. Jobs must be scheduled in the same order on all machines, and
each machine can not be simultaneously assigned to two jobs. In [1,2], a parallel cooperative multi-objective
meta-heuristic has been proposed to solve this problem. The work has been experimented on a large problem
instance of 200 jobs on 10 machines an IBM-SP2 of 16 processors. As the implementation is not fault tolerant,
the experiments have been stopped by far before the convergence is reached. In this paper, we propose to use
the computational grid to overcome such problem. Our proposed work is based on a gridding of the island
and multi-start models and their cooperation. It allows to fully exploit the cooperation and provides clearly
better results.

The rest of the paper is organized as follows: Section 2 briefly presents the multi-objective meta-heuristics
and their associated parallel and cooperative models. Section 3 describes the characteristics of the targeted
computational grid and their impact on parallel cooperative meta-heuristics. Dispatcher–Worker grid middle-
wares are discussed and their limitations are highlighted. In Section 4, a coordination model for such family of
middlewares is proposed with an implementation on top of XtremWeb [6]. Section 5 presents the experimen-
tation of the model and its implementation through a parallel cooperative meta-heuristic applied to the Bi-
objective Permutation Flow-Shop Scheduling Problem (BPFSP), and analyzes the preliminary experimental
results. Finally, Section 6 concludes the paper.

2. Parallel cooperative meta-heuristics

2.1. Parallel local searches

LSs could be viewed as “walks through neighborhoods” meaning search trajectories through the solutions
domains of the problems at hand. The walks are performed by iterative procedures that allow to move from a
solution to another one in the solution space (see Algorithm 2.1). LSs perform particularly the moves in the
neighborhood of the current solution. The walks start from a solution randomly generated or obtained from
another optimization algorithm. At each iteration, the current solution is replaced by another one selected from the set of its neighboring candidates. The search process is stopped when a given condition is satisfied (stopping criterion). A powerful way to achieve high performance with LSs is the use of parallelism.

**Algorithm 2.1. LS skeleton pseudo-code**

```
Generate(s(0));
t := 0;
while not Termination_Criterion(s(t)) do
    m(t) := SelectAcceptableMove(s(t));
    s(t + 1) := ApplyMove(m(t), s(t));
    t := t + 1;
endwhile
```

Three parallel models of LSs are commonly used in the literature: the parallel multi-start model, the parallel moves model (or parallel exploration and evaluation of the neighborhood), and the move acceleration model (or parallel evaluation of a single solution). The parallel multi-start model consists in simultaneously launching several LSs for computing better and robust solutions. They may be heterogeneous or homogeneous, independent or cooperative, start from the same or different solution(s), configured with the same or different parameters. The parallel moves model is a low-level Farmer–Worker model that does not alter the behavior of the heuristic. A sequential search computes the same results slower. At the beginning of each iteration, the farmer duplicates the current solution between distributed nodes. Each one manages some candidates and the results are returned to the farmer. In the move acceleration model, the quality of each move is evaluated in a parallel centralized way. That model is particularly interesting when the evaluation function can be itself parallelized as it is CPU time-consuming and/or IO intensive. In that case, the function can be viewed as an aggregation of a certain number of partial functions.

### 2.2. Parallel evolutionary algorithms

Evolutionary algorithms (EAs) are stochastic search techniques that have been successfully applied to many real and complex problems. An EA is an iterative technique that applies stochastic operators on a pool of individuals (the population) (see Algorithm 2.2). Every individual in the population is the encoded version of a candidate solution. Initially, this population is generated randomly. An evaluation function associates a fitness value to every individual indicating its suitability to the problem.

**Algorithm 2.2. EA pseudo-code**

```
Generate(P(0));
t := 0;
while not Termination_Criterion(P(t)) do
    Evaluate(P(t));
    P'(t) := Selection(P(t));
    P'(t) := Apply_Reproduction_Ops(P'(t));
    P(t + 1) := Replace(P(t), P'(t));
    t := t + 1;
endwhile
```

The above pseudo-code shows the genetic components of any EA. There are several categories of EAs depending on how the individuals are coded or how each step of the algorithm works. The major classes of EAs are the genetic algorithms (GAs), evolutionary programming (EP), and evolution strategies (ESs).

Multi-objective optimization consists generally in optimizing a vector of \( nb_{obj} \) objective functions \( F(x) = (f_1(x), \ldots, f_{nb_{obj}}(x)) \), where \( x \) is an \( d \)-dimensional decision vector \( x = (x_1, \ldots, x_d) \) from some universe.
called decision space. The space the objective vector belongs to is called the objective space. $F$ can be defined as a cost function from the decision space to the objective space that evaluates the quality of each solution $(x_1, \ldots, x_d)$ by assigning it an objective vector $(y_1, \ldots, y_{nb, obj})$, called the fitness.

A multi-objective problem (MOP) may have a set of solutions known as the Pareto optimal set rather than an unique optimal solution. The image of this set in the objective space is denoted as Pareto front. Graphically, a solution $x$ is Pareto optimal if there is no other solution $x'$ such that the point $F(x')$ is in the dominance cone of $F(x)$. This dominance cone is the box defined by $F(x)$, its projections on the axes and the origin (Fig. 1).

In [5], three major parallel models for EAs are identified: the island (a)synchronous cooperative model, the parallel evaluation of the population, and the distributed evaluation of a single solution. The last model is similar to the move acceleration model presented above.

In the island model, several homo/heterogeneous EAs run simultaneously and cooperate to compute better and robust solutions. They exchange genetic stuff to improve the diversity of the search. The model aims at delaying the global convergence, especially when the EAs are heterogeneous regarding to the variation operators. The migration of individuals follows a policy defined by the following parameters: the migration decision criterion, the exchange topology, the number of emigrants, the emigrants selection policy, and the replacement/integration policy.

The evaluation of the population is necessary because it is in general the most time-consuming step of an EA. The parallel evaluation is based on the Farmer–Worker paradigm. The farmer applies the selection, transformation and replacement operations as they require a global management of the population. At each generation, it distributes the new solutions among the workers, which evaluate them and return back their fitness values. An efficient execution is often obtained particularly when the evaluation of each solution is costly.

2.3. Cooperative meta-heuristics

Combinations of different meta-heuristics often provide very powerful search methods. In [14], two levels and two modes of cooperation are distinguished: Low and High levels, and Relay and Cooperative modes. The low-level cooperation consists in replacing an internal function (e.g. an operator) of a given meta-heuristic by another meta-heuristic. In high-level cooperative algorithms, the different meta-heuristics are self-containing, meaning no direct relationship to their internal working is considered. Relay cooperation means a set of meta-heuristics is applied in a pipeline way. The output of a meta-heuristic (except the last) is the input of the following one (except the first). In the teamwork cooperation mode, several meta-heuristics evolve simultaneously and each of them performs a search in a solution space, and exchanges solutions with some others. In this paper, we address the high-level cooperation mechanism in the relay and cooperative modes.

![Fig. 1. Example of non-dominated solutions.](image-url)
In this paper, we focus only on the coarse-grained models: the island model and the multi-start model and their cooperation.

3. Multi-objective meta-heuristics on computational grids

The proliferation of research and industrial projects on grid computing is leading to the proposition of several sometimes confusing definitions of the grid concept [8,12]. As a consequence, some articles such as [4,9] are especially dedicated to the analysis of these definitions. It is thus important to clearly define the context on which a work on grid computing is focused when it is presented. In this paper, the targeted architecture is the computational grid as defined in [12]. A computational grid is a scalable pool of heterogeneous and dynamic resources geographically distributed across multiple administrative domains and owned by different organizations. The characteristics could be summarized as follows:

- **The grid includes multiple autonomous administrative domains**: the users and providers of resources are clearly identified. This allows to reduce the complexity of the security issue, however, the firewall traversal remains a critical problem to deal with. In global computing middlewares based on the large-scale cycle stealing such as XtremWeb [6], the problem is solved in a natural way as communications are initiated from “inside the domains”.
- **The grid is heterogeneous**: the heterogeneity in a grid is intensified by its large number of resources belonging to different administrative domains. The emergence of data exchange standards and Java-based technologies such as RMI allows to deal with the heterogeneity issue.
- **The grid has a large scale**: the grid has a large number of resources growing from hundreds of integrated resources to millions of PCs. The design of performant and scalable grid applications has to take into account the communication delays.
- **The grid is dynamic**: the dynamic temporal and spatial availability of resources is not an exception but a rule in a grid. Due to the large-scale nature of the grid the probability of some resource failing is high. Such characteristic highlights some issues such as dynamic resource discovery, fault tolerance, and so on.

The gridification of parallel cooperative meta-heuristics requires to take into account at the same time the characteristics and underlined issues of the computational grids and the parallel cooperative models. Some of the issues related to grids may be solved by middlewares allowing to hide their inherent complexity to the users. The number of issues that could be solved in a transparent way for the users depends on the middleware at hand. The choice of this later is crucial for performance and ease of use. In our case, recall that in the island model the islands cooperate according to a given migration topology. Maintaining a logical communication topology in a volatile environment may be complex and inefficient due to the high cost of the dynamic reconfiguration of the topology. One of the approaches allowing to deal with such issue is based on a shared space for storing the emigrant solutions between islands. The island that initiates a migration operation sends the emigrants to the shared space, and these later are stored together with the identity of their source island. Islands can also initiate immigration operations by sending requests to the shared space, and immigrants are randomly chosen from this later. In [3], it has experimentally proved that random topologies (random selection of the target islands) could be as efficient as the common topologies (ring, mesh, etc.). Grid middlewares that support such approach are Dispatcher–Worker ones such as XtremWeb [6]. In such systems, clients can submit their jobs to the dispatcher. A computational pool of volatile workers request the jobs from the dispatcher according to the large-scale cycle stealing model. Then, they execute the jobs and return back the results to the dispatcher to be collected later by the clients. The islands could be deployed as workers and the dispatcher could serve to provide the global space.

One of the major limitations of such middlewares is that they are well-suited for embarrassingly parallel (e.g. multi-parameter) applications with independent tasks. In this case, no communication is required between the tasks, and thus workers. The deployment of parallel cooperative meta-heuristics that need cross-worker/task communication is not straightforward. The programmer has the burden to manage and control the complex coordination between the workers. To deal with such problem existing middlewares must
be extended with a software layer which implements a coordination model. Several interesting coordination models have been proposed in the literature \cite{11,13}. In this paper, we focus only on one of the most popular of them i.e. Linda \cite{10} as our proposed model is an extension of this model.

4. A coordination model for dispatcher–worker middlewares

In the Linda model, the coordination is performed through generative communications. Processes share a virtual memory space called a tuple-space (set of tuples). The fundamental data unit, a tuple, is an ordered vector of typed values. Processes communicate by reading, writing, and consuming these tuples. A small set of four simple operations allows highly complex communication and synchronization schemata (see Table 1).

Nevertheless, Linda has several limitations regarding the design and deployment of parallel cooperative meta-heuristics on computational grids. First, it does not allow rewriting operations on the tuple space. Due to the high communication delays in a grid, tuple rewriting is very important as it allows to reduce the number of communications and the synchronization cost. Indeed, in Linda a rewriting operation is performed as an “in” or “rd” operation followed by a local modification and an “out” operation. The operations “in”/“rd” and “out” involve two communications and an heavy synchronization. Therefore, the model needs to be extended with a rewriting operation. Furthermore, the model does not support group operations that are useful for efficiently writing/reading Pareto sets in/from the tuple space. Finally, non-blocking operations that are very important in a volatile context are not supported in Linda. In the next section, we propose an extension of the Linda model that allows to meet these requirements.

4.1. An extended Linda model

Designing a coordination model for parallel cooperative multi-objective meta-heuristics requires the specification of the content of the tuple space, a set of coordination operations and a pattern matching mechanism. The tuple space may be composed of a set of Pareto optimal solutions and their corresponding solutions in the objective space. For the parallel island model of the multi-objective meta-heuristics, the tuple space contains a collection of (parts of) Pareto optimal sets deposited by the islands for migration. The mathematical formulation of the tuple space (Pareto Space or $PS$) is the following:

$$PS = \bigcup PO, \text{ with } PO = \{(x, F(x)), \text{ } x \text{ is Pareto optimal}\}$$

In addition to the operations provided in Linda, parallel multi-objective optimization on grids needs other operations. These operations fall into two categories: group operations and non-blocking operations. Group operations are useful to manage multiple Pareto optimal solutions. Non-blocking operations are necessary to take into account the volatile nature of computational grids. In our model, the coordination primitives are defined as in Table 2.

The update operation allows to locally update the Pareto space, and so to reduce the communication and synchronization cost. The pattern matching mechanism depends strongly on how the model is implemented, and in particular on how the tuple space is stored and accessed. For instance, if the tuple space is stored in a database the mechanism can be the request mechanism used by the database management system. More details on the pattern matching mechanism of our model are given in the next section.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>The Linda model operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>Associated role</td>
</tr>
<tr>
<td>out(tup)</td>
<td>Puts tuple into tuple-space</td>
</tr>
<tr>
<td>in(pattern)</td>
<td>Removes a (often the first) tuple matching pattern from tuple-space</td>
</tr>
<tr>
<td>rd(pattern)</td>
<td>Is the same as in(pattern) but does not remove the tuple from tuple-space</td>
</tr>
<tr>
<td>eval(expression)</td>
<td>Puts expression in tuple-space for evaluation. The evaluation result is a tuple left in tuple-space</td>
</tr>
</tbody>
</table>
4.2. Implementation on top of XtremWeb

XtremWeb [6] is a Java global computing project developed at Paris-Sud University. It is intended to distribute applications over a computational pool, and is dedicated to multi-parameter applications that have to be computed several times with different inputs. XtremWeb manages tasks following the Dispatcher–Worker paradigm (see Fig. 2). Tasks are scheduled by the dispatcher to workers only on their specific demand since they may adaptively appear (connect to the dispatcher) and disappear (disconnect from the dispatcher). The tasks are submitted by either a client or a worker, and in the latter case, the tasks are dynamically generated for parallel execution. The final or intermediate results returned by the workers are stored in a MySQL database. These results can be requested later by either the clients or the workers. The database stores also different information related to the workers and the deployed application tasks.

XtremWeb is well-suited for embarrassingly parallel applications where no communication occurs between workers, and these can only communicate with the dispatcher. Yet, many parallel distributed applications particularly parallel multi-objective meta-heuristics need cooperation between workers. In order to free the user from the burden of managing himself/herself such cooperation we propose an extension of the middleware with a software layer.

Table 2
The extended Linda model operations

<table>
<thead>
<tr>
<th>Operation(s)</th>
<th>Associated role(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, rd, out, eval</td>
<td>These operations are the same as those of Linda defined in Section 3</td>
</tr>
<tr>
<td>ing(pattern)</td>
<td>Withdrawing from PS all the solutions matching the specified pattern</td>
</tr>
<tr>
<td>rdg(pattern)</td>
<td>Reads from PS a copy of all the solutions matching the specified pattern</td>
</tr>
<tr>
<td>outg(setOfSolutions)</td>
<td>Inserts multiple solutions in PS</td>
</tr>
<tr>
<td>update(pattern, expression)</td>
<td>Updates all the solutions matching the specified pattern by the solutions resulting from the evaluation of expression</td>
</tr>
<tr>
<td>inIfExist, rdIfExist, ingIfExist and rdgIfExist</td>
<td>These operations have the same syntax than, respectively, in, rd, ing and rdg but they are non-blocking probe operations</td>
</tr>
</tbody>
</table>

Fig. 2. Global architecture of XtremWeb.
The software layer is an implementation of the proposed model composed of two parts (see Fig. 3): a coordination API and its implementation at the worker level and a coordination request broker (CRB). The Pareto Space is a part of the MySQL database associated with the dispatcher. Each tuple or solution of the Pareto Space is stored as a record in the database.

From the worker side the coordination API is implemented in Java and in C/C++. The C/C++ version allows the deployment and execution of C/C++ applications with XtremWeb (written in Java). The coordination library must be included in these programmer applications. From the dispatcher side, the coordination API is implemented in Java as a Pareto Space manager. The CRB is a software broker allowing the workers to transport their coordination operations calls to the dispatcher, and has two components: one for the worker (CRB stub) and another for the dispatcher (CRB skeleton). The role of the CRB stub is to transform the local calls to the coordination operations performed by the tasks executed by the worker into RMI calls. The role of the CRB skeleton is to transform these RMI calls into local calls to the coordination operations performed by the Pareto Space Manager. These local calls are translated into MySQL requests addressed to the Pareto Space.

To illustrate the implementation of the coordination layer on top of XtremWeb, let us consider the scenario presented in Fig. 3. The work unit performed by an XtremWeb worker calls the `ing(template)` coordination operation. In the C++ version of the coordination API, the implementation of each coordination operation makes the system call `execvp()` with appropriate parameters to plug in the CRB Stub Java object. In our scenario, the major parameters are the number `ING` designating the operation and the file `ARGS_FILE` containing the arguments specified in the `template` parameter. CRB Stub translates the `ing` local call into an RMI call to the CRB_Skeleton Java object. This latter translates the RMI call into a local call to the `ing` operation implemented in the Pareto Space Manager class. The implementation of the coordination operation consists in a MySQL select request addressed to the Pareto Space part of the XtremWeb information database.

Note that the method declarations for the coordination operations in the Pareto Space Manager class contain the Java `synchronized` keyword. Hence, the system associates an unique lock with the instance of the Pareto Space Manager class. Whenever control enters a synchronized coordination operation, other calls to a synchronized cooperation method are blocked until the Pareto Space Manager object is unlocked. In the next section, the proposed coordination model is applied to parallel cooperative multi-objective meta-heuristics.
5. Application to BPFSP and experimentation

5.1. Problem formulation

The Flow-Shop Problem is a scheduling problem [16] that has received a great attention given its importance in many industrial areas. The problem can be formulated as a set of $N$ jobs $J_1, J_2, \ldots, J_N$ to be scheduled on $M$ machines. The machines are critical resources as each machine cannot be simultaneously assigned to two jobs. Each job $J_i$ is composed of $M$ consecutive tasks $t_{i1}, \ldots, t_{iM}$, where $t_{ij}$ represents the $j$th task of the job $J_i$ requiring the machine $m_j$. To each task $t_{ij}$ is associated a processing time $p_{ij}$, and each job $J_i$ must be achieved before a due date $d_i$.

In this paper, we focus on the Bi-objective Permutation Flow-Shop Problem (BPFSP) where jobs must be scheduled in the same order on all the machines (see Fig. 4). Therefore, two objectives have to be minimized:

- $C_{\text{max}}$: Makespan (Total completion time),
- $T$: Total tardiness.

The task $t_{ij}$ being scheduled at time $s_{ij}$, the two objectives can be formulated as follows:

$$f_1 = C_{\text{max}} = \max \{s_{iM} + p_{iM} | i \in [1 \ldots N]\}$$

$$f_2 = T = \sum_{i=1}^{N} [\max(0, s_{iM} + p_{iM} - d_i)]$$

The Pareto front $PF$ associated with BPFSP may be formulated as follows:

$$\forall y, \exists x \in PF, (m(x) \leq m(y)) \text{ or } (t(x) \leq t(y))$$

where $x$ and $y$ are solutions of the MOP, and $m(x)$ (respectively, $t(x)$) is the value of $x$ corresponding to the makespan (respectively, tardiness) criterion.

5.2. A genetic–memetic algorithm for solving BPFSP

In single objective optimization, it is well known that GAs provide better results when they are hybridized with LS algorithms. Indeed, the GA convergence is too slow to be really effective without any cooperation [15]. In [1,2], a hybrid genetic–memetic algorithm named AGMA has been proposed for solving BPFSP. The simplified pseudo-code of the algorithm is presented in Algorithm 1 and illustrated in Fig. 5.

Algorithm 1. AGMA algorithm

Create an initial population

while run time not reached do

Perform a GA generation with adaptive mutation

Update $PO^*$ and $PPO^*$

if $PPO^* < \alpha$ then

Perform a generation of MA on the population (Algorithm 2)

Update $PO^*$ and $PPO^*$

end if

Update selection probability of each mutation operator

end while

AGMA combines a genetic algorithm (GA) and a memetic algorithm (MA). In this paper, we do not give the details and parameters of the two algorithms, and if needs be, the reader is referred to [1,2]. The GA uses mainly two parameters: an archive (Pareto Front) $PO^*$ of non-dominated solutions, and a progression ratio.
At each generation, these two parameters are updated. If no significant progression is noticed ($P_{po} < \alpha$, where $\alpha$ is a fixed threshold), an intensified search process is triggered. The intensification consists in applying MA (see Algorithm 2) to the current population during one generation. The application of MA returns a Pareto Front $PO^{*/C3}$ that serves to update the Pareto Front $PO^{*}$ of the GA. MA consists in selecting randomly a set of solutions from the current population of the GA. A crossover operator is then applied to these solutions and new solutions are generated. Among these new solutions only non-dominated ones are maintained to constitute a new Pareto Front $PO^{*/C3}$0. An LS is then applied to each solution of $PO^{*/C3}$0 to compute its neighborhood. The non-dominated solutions belonging to the neighborhood are inserted into $PO^{*/C3}$.

Algorithm 2. MA algorithm

\begin{algorithm}
\begin{algorithmic}
\While {MA run time not reached}
\State Select randomly a set $P$ of solutions from the current population
\State Apply the crossover on $P$ to generate a set $P'$ of new solutions
\State Compute the non-dominated set $PO^{*/C3}$ from $P'$
\While {New solutions found}
\State Create the neighborhood $N$ of each solution of $PO^{*/C3}$
\State Let $PO^{*/C3}$ be the non-dominated set of $N \cup PO^{*/C3}$
\EndWhile
\EndWhile
\end{algorithmic}
\end{algorithm}

5.3. Parallel cooperative AGMA

Different parallel models have been sketched and analyzed in Section 2. The fine-grained parallel models could not be exploited efficiently in a volatile environment due to the communication delays. In BPFSP,
the model based on parallel evaluation of each solution is fine-grained and is not likely to lead to better performance. Indeed, the evaluation of each objective has a low cost. Therefore, it is useless to evaluate in parallel the two objectives and evaluate each of them in parallel. Conversely, it is useful to exploit the following parallel models: (1) the island model that consists in performing in parallel several cooperative AGMAs; (2) the parallel evaluation of the population of each AGMA; and (3) the multi-start model that consists in applying in parallel an LS on each solution of the Pareto Front $PO^*$ in MA. The parallel evaluation of the neighborhood of each solution could not be efficient for the same reason as the parallel evaluation of each solution.

We have limited our implementation to the coarse-grained parallel models, i.e. the island model and the multi-start model. Fig. 6 illustrates the parallel cooperative AGMA exploiting these two models.

Fig. 6. Illustration of parallel AGMA.
• The island model: in our implementation (see Fig. 6), the parameters of the model are the following: the different cooperative AGMA exchange individuals selected from their archives $PO'$. At their arrival, the immigrants are merged with the local archive. Migrations occur periodically (after a fixed period of time). The migration topology is the random one, meaning the destination island is selected randomly.

• The multi-start model: the multi-start model is exploited during the execution of MA. Each solution of the Pareto Front $PO'$ computed by the algorithm represents the initial solution of an LS method which calculates its neighborhood. The different LSs are executed in parallel according to the Master-Slave model. The master, i.e. the algorithm MA merges with $PO'$ the neighborhoods returned by the different slaves and computes the new $PO''$ that contains the non-dominated solutions.

5.4. Deployment and fault tolerance

A deployment scheme may be defined as a function that consists in embedding the different components of the parallel models on the different components of the computational grid. Different deployment schemata of the island and multi-start models on XtremWeb are possible. Indeed, the AGMA algorithms of the island model can be deployed either as XtremWeb clients or workers. For the multi-start model, the master can be either a client or a worker, and the slaves are necessarily deployed as workers. For our experimentation, the deployment scheme is illustrated in Fig. 7.

The island model is deployed on XtremWeb workers, and each worker runs the AGMA algorithm. During the cooperation phase (execution of MA), the LSs initiated on the Pareto Front $PO'$ are submitted as tasks to the dispatcher that launches them on the workers at their request. The multi-start model is thus deployed on a worker and the LSs are performed by a pool of workers.

In XtremWeb, the fault tolerance issue is tackled at worker and dispatcher levels. When a worker fails the work unit being executed is re-started from scratch. If the dispatcher crashes it is re-started using its information database. The problem with such solution is that in a highly volatile environment a large amount of CPU time is wasted as the system spends its time in re-starting work units performed by the workers. For the Flow-Shop problem, unlike the AGMA algorithms the local searches are not highly CPU time consuming (few seconds). Therefore, if a worker performing an LS fails it is re-started by XtremWeb on another worker. The failures of the dispatcher are also managed by XtremWeb. The failures of workers performing the AGMA algorithm are managed in a different way. Indeed, we propose a checkpointing approach at the worker level that allows to solve more efficiently the failures of workers running the AGMA algorithm.

![Diagram](image_url)  
**Fig. 7.** Deployment scheme of parallel cooperative AGMA on top of XtremWeb.
The proposed approach is the following: each AGMA state is associated in the Pareto space with a checkpointing tuple: \([\text{agmald}, \text{gen}, \text{pop}, \text{pf}]\), where \(\text{agmald}\), \(\text{gen}\), \(\text{pop}\), and \(\text{pf}\) designate, respectively, the AGMA unique identifier, the number of performed generations, the population individuals and the Pareto front at the last checkpoint. The tuple is updated every checkpoint period (an user-parameter). In our experiments, it has been fixed to 10 min. If an AGMA is stopped on a failed worker it is re-started later by the dispatcher on another worker at this later request.

### 5.5. Experimental results

In our experiments, we consider the large-sized Taillard’s BPFSP instance 200 jobs on 10 machines that has never been solved. The parameters of the island model are fixed as the following: migrations occur every 10 min, the number of emigrants is fixed at each migration operation to 20 the size of the archive \(PO^*\) is upper than 20, and the whole archive otherwise, and the population size of each AGMA is 100.

Traditionally, the migration frequency designates a fixed number of generations. This is well adapted for homogeneous execution environments. However, in grids where computers are heterogeneous the islands evolving on more powerful machines will sweep along the other islands to converge to the same solutions. Such super-individual problem leads to a premature convergence and a loss of diversity. Conversely, islands evolving on less powerful machines will have any effect on those evolving on powerful machines (non-effect problem). In this paper, the frequency parameter represents a fixed period of time. This allows to reduce the number of migration operations performed by the islands hosted by powerful machines, and thus to limit their influence. After a migration operation on a processor is triggered the frequency is re-initialized. If the processor becomes unavailable type island is deployed on another processor and the frequency is re-initialized on that processor. As a consequence, some migrations are delayed but Table 4 shows that the island model still improves the quality of the provided solutions.

The application has been deployed during working days (non-dedicated environment) on the grid illustrated by Fig. 8. The grid is mainly composed of three education networks (with different administrative domains) belonging to three education schools: the Polytech’Lille engineering school, the IUT A technological institute and the University of Lille.

![Fig. 8. Illustration of the computational grid.](image-url)
Table 3 gives a detailed illustration of the experimentation computational grid: the PC type and its operating system, the administrative domain it belongs to, the number of PCs of that type, and the role of each PC (dispatcher or worker). There is only one PC that hosts the dispatcher, and it belongs to another (a fourth) administrative domain.

Five parallel cooperative versions of AGMA have been experimented, evaluated and compared. Note that Version 1 has been proposed in another paper [2] and has been experimented on an IBM-SP2 machine.

- **Version 0**: corresponds to a single fault tolerant AGMA with serial local searches. The parallel multi-start model is not exploited.
- **Version 1**: exploits only the island model using 10 islands on an IBM-SP2 (RS6000) machine composed of four nodes of 16 processors. Only 10 processors Power 4 (1.1 GHz CPU and 16 GB RAM) have been exploited for experiments. Note that Version 1 is not fault tolerant.
- **Version 2.0**: is the same as Version 0 except that the multi-start model is deployed in a distributed way on a pool of XtremWeb workers according to the cycle stealing paradigm (**Pull** mode, i.e. work distribution is initiated by the workers).
- **Versions 2.1 and 2.2**: are two fault tolerant versions that allow the deployment of a combination of the multi-start and the island models with, respectively, 10 and 30 islands.

These different versions allow us to evaluate the contribution of the parallel models and the grid computing to the effectiveness of hybrid meta-heuristics. First, Fig. 9 illustrates the Pareto fronts obtained with the different versions. For version 1 that corresponds to version 1.2 (2nd run over 10 runs) on the figure, the results are obtained after 24 h. Such time corresponds to the longer execution time (among 10 runs) before an application crash occurred as version 1 is not fault tolerant. The results corresponding to our own versions are obtained after 10 days 6 h and 40 min. The graphics show that the island model improves the quality of the Pareto fronts. The improvement is more significant between 1 and 10 island(s) than between 10 and 30 islands. Additional experiments are needed to determine the threshold number of islands that maximizes the quality of the provided front. Furthermore, the Pareto fronts obtained with Versions 0 and 2.0 show that the multi-start model improves the effectiveness of the local search. The use of parallelism allow to perform more local searches, and thus a better intensification of the search.

Table 4 shows the detailed improvements based on the S-metric measure realized with the parallel models on the computational grid. An S-metric [17] is defined as the volume of the search space dominated by a Pareto front. It allows to evaluate the effectiveness of a Pareto front in terms of convergence and diversity. The bi-objective solution that has served as a reference in the objective space to measure the S-metric is (57,864, 11,268). Fig. 10 illustrates the evolution of S-metric during the last 60 h for the different versions. The algorithm is converging prematurely for the serial version compared to the others.

<table>
<thead>
<tr>
<th>CPU (GHz)</th>
<th>OS</th>
<th>Domain</th>
<th>Role</th>
<th>Nbr</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4 3.06</td>
<td>Redhat.9</td>
<td>rech-info.yser.net</td>
<td>Dispatcher</td>
<td>1</td>
</tr>
<tr>
<td>P4 1.70</td>
<td>Mandrake.10</td>
<td>fil.univ-lille1.fr</td>
<td>Worker</td>
<td>24</td>
</tr>
<tr>
<td>P4 2.40</td>
<td></td>
<td></td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>P4 2.80</td>
<td></td>
<td></td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>P4 3.00</td>
<td></td>
<td></td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>AMD 1.30</td>
<td>Debian</td>
<td>students.deule.net</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>Celeron 0.80</td>
<td></td>
<td></td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>Celeron 2.00</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>Celeron 2.20</td>
<td></td>
<td></td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>P3 1.20</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>P4 1.60</td>
<td></td>
<td>iut-info.univ-lille1.fr</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>P4 2.80</td>
<td></td>
<td></td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>P4 3.00</td>
<td></td>
<td></td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>321</td>
</tr>
</tbody>
</table>

Table 3 Parameters of the computational grid

<table>
<thead>
<tr>
<th>CPU (GHz)</th>
<th>OS</th>
<th>Domain</th>
<th>Role</th>
<th>Nbr</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4 3.06</td>
<td>Redhat.9</td>
<td>rech-info.yser.net</td>
<td>Dispatcher</td>
<td>1</td>
</tr>
<tr>
<td>P4 1.70</td>
<td>Mandrake.10</td>
<td>fil.univ-lille1.fr</td>
<td>Worker</td>
<td>24</td>
</tr>
<tr>
<td>P4 2.40</td>
<td></td>
<td></td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>P4 2.80</td>
<td></td>
<td></td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>P4 3.00</td>
<td></td>
<td></td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>AMD 1.30</td>
<td>Debian</td>
<td>students.deule.net</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>Celeron 0.80</td>
<td></td>
<td></td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>Celeron 2.00</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>Celeron 2.20</td>
<td></td>
<td></td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>P3 1.20</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>P4 1.60</td>
<td></td>
<td>iut-info.univ-lille1.fr</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>P4 2.80</td>
<td></td>
<td></td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>P4 3.00</td>
<td></td>
<td></td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>321</td>
</tr>
</tbody>
</table>
In Table 4, the first row gives the values of the S-metric corresponding to the Pareto front of each version obtained after 10 days 6 h and 40 min. The second row illustrates the improvements of the Pareto front realized by the versions exploiting the parallel local search (multi-start model) alone or together with the parallel island model. The parallel multi-start model allows to perform more local searches in a fixed time compared to a serial local search. Therefore, the quality of the Pareto front in term of convergence is improved by a factor of 20%. The parallel island model combined with the parallel local search allows to improve at the same time the convergence (depth) and the diversity (width) of the Pareto front. As a consequence, the quality (S-metric) of this front is improved by a factor of 48% and 60% with, respectively, 10 and 30 islands.

Table 5 gives some execution statistics collected during the execution of the different versions. The first row shows that the fault tolerant mechanism has really been used allowing reliable executions. Each AGMA has crashed at least 10 times in average. The second row shows mainly that the parallel multi-start model allows more local searches and thus a better improvement of the quality of the obtained Pareto fronts as it was explained above. The last row gives the number of checkpointing and migration operations performed during execution. The two kinds of operations are performed together. An average of over 300 operations are performed by each island (AGMA). On the other hand, the average execution time of each operation is about 2.7 s. The total checkpointing and migration cost is thus very low: 13 min over 10 days 6 h and 40 min.

![Fig. 9. A comparison of the Pareto fronts obtained with the different versions.](image-url)
6. Conclusion and future work

The cooperation of meta-heuristics having complementary behaviors allows to enhance the effectiveness and robustness in combinatorial optimization [14]. However, its exploitation for solving real-world problems is possible only by using a great computing power. Large-scale parallelism based on the use of computational grids is recently revealed to be a good way to get at hand such computing power and fully exploit cooperation. At our best of knowledge, no research work has been published on parallel cooperative meta-heuristics on grids. Nowadays, existing Dispatcher–Worker grid computing middlewares are inadequate for the deployment of parallel cooperative applications. Indeed, these need to be extended with a software layer to support the cooperation. In this paper, we have proposed a Linda-like cooperation model that has been implemented on top of XtremWeb.

In [1,2], a cooperative meta-heuristic (AGMA) has been proposed and experimented on BPFSP. The performed experiments on large-size instances such as 200 jobs on 10 machines are often stopped without convergence being reached. The full exploitation of the cooperation needs a large amount of computational resources and the management of the fault tolerance issue. We have proposed a fault-tolerant cooperative parallel design of the AGMA combining two parallel models: the multi-start model and the island model. The algorithm has been implemented on our extended version of XtremWeb.
The first experiments have been performed on a multi-domain education network. The network is composed of 321 heterogeneous Linux PCs. The preliminary results, obtained after several execution days, demonstrate that the use of grid computing allows to fully exploit effectively different parallel models and their combination for solving large-size problem instances. In addition, the results show that the proposed checkpointing-based fault tolerance approach induces a very low overhead. Beyond the improvement of the effectiveness, the parallelism on grids allows to push far the limits in terms of computational resources. As a consequence, it permits to better evaluate the benefits and limitations of the cooperation.

In the future, the focus will be on the cooperation of meta-heuristics with exact methods to provide exact Pareto fronts. The meta-heuristics will serve to provide better bounds to reduce the exploration cost of the exact methods.

**Acknowledgement**

This work is supported by the French Government through the national joint grid project GGM of ACI-Masse de Données.

**References**


