Genetic algorithms for scheduling in a CPU/FPGA architecture with heterogeneous communication delays

Fadel Abdallah, Camel Tanougast, Imed Kacem, Camille Diou, Daniel Singer

Version of Record: https://www.sciencedirect.com/science/article/pii/S0360835219304644
Manuscript_ac85528152b614b0f13e63a87b89aea3

Genetic algorithms for scheduling in a CPU/FPGA architecture with heterogeneous communication delays

Fadel Abdallah[a], Camel Tanougast[a], Imed Kacem[a,*], Camille Diou[a], Daniel Singer[a]

[a] *Université de Lorraine, LCOMS, EA 7306, 7 rue Marconi, Metz, F-57070,France*

Abstract

In this paper we study a CPU/FPGA heterogeneous architecture scheduling problem (often referred as Multi-Processors System on Chip or MPSoC) with communication delays' constraints. In this context, we propose two approaches based on genetic algorithms. Their main goal is to run in the MPSoC an application, which is described by a given data flow graph. The aim is to minimize the schedule length (makespan). Computational experiments are conducted to evaluate the proposed algorithms. The obtained results show that the two approaches are often capable of finding optimal or near optimal solutions for the studied problem while improving significantly the running time compared to existing works.

Keywords: Genetic Algorithm, Task Scheduling Problem, Combinatorial Optimization, Metaheuristics, Makespan (schedule length), Linear Programming, Heterogeneous system, Communication delays

*corresponding author. Tel.: +33 387547136; fax: +33 387547307.
Email addresses: fadel.abdallah@univ-lorraine.fr (Fadel Abdallah), camel.tanougast@univ-lorraine.fr (Camel Tanougast), imed.kacem@univ-lorraine.fr (Imed Kacem), camille.diou@univ-lorraine.fr (Camille Diou), daniel.singer@univ-lorraine.fr (Daniel Singer)

Genetic algorithms for scheduling in a CPU/FPGA architecture with heterogeneous communication delays

Abstract

In this paper we study a CPU/FPGA heterogeneous architecture scheduling problem (often referred as Multi-Processors System on Chip or MPSoC) with communication delays' constraints. In this context, we propose two approaches based on genetic algorithms. Their main goal is to run in the MPSoC an application, which is described by a given data flow graph. The aim is to minimize the schedule length (makespan). Computational experiments are conducted to evaluate the proposed algorithms. The obtained results show that the two approaches are often capable of finding optimal or near optimal solutions for the studied problem while improving significantly the running time compared to existing works.

Keywords: Genetic Algorithm, Task Scheduling Problem, Combinatorial Optimization, Metaheuristics, Makespan (schedule length), Linear Programming, Heterogeneous system, Communication delays

1. Introduction

The parallel machine (identical, uniform and unrelated) scheduling problems for minimizing the schedule length or the maximum completion time (i.e., the makespan) are very important combinatorial problems, which have been proven to be NP-hard [9, 10, 11, 12, 13, 14, 16, 17]. Actually, the literature shows that there are several real and industrial problems for which the main question can be reduced as a parallel machine scheduling problem. As an example, we can cite the different applications in supply chain, transportation, aviation and bicycle sharing systems' problems [4, 13, 15]. As a consequence, a huge literature exists on this topic and on the related applications. Several approaches have been investigated: mathematical models [15, 16], heuristic algorithms [12, 14, 16, 17, 18], genetic algorithms [5, 15, 11] and others. Despite their interest, most of these existing works do not take into account the trend of computing technologies. Consequently, they cannot be directly applied to heterogeneous parallel computing systems where setup times are not included in the job processing times, or when heterogeneous communication delays exist, with a strong impact on the makespan performance. Indeed, the evolution of applications (such as signal and image processing) in terms of complexity and the need for systems' flexibility have progressively led the Integrated Circuit (IC) designers to elaborate reconfigurable heterogeneous architectures based on systems-on-chip. Usually, these embedded systems are composed of Central Processing Units (CPUs), communication on chip resources and Field Programmable Gate Array (FPGA) [6, 7]. The main goal of these heterogeneous computing architectures is to meet various application requirements by combining logic and software cores (real-time, high-rate computation, and so on.) and to achieve a better performance by minimizing the running time. Indeed, an effective way to ensure a high-performance and real-time execution is to distribute the computations on a heterogeneous reconfigurable architecture based on logic programmable areas and processor cores (i.e., CPUs, Digital Signal Processing (DSP), and so on.). These Multi-Processors Systems-on-Chip (MPSoC) including FPGA chips provide flexibility and better performance compared to CPUs, up to 10 times [8].

From the theoretical point of view, the mapping and scheduling problems on heterogeneous architectures with communication delay constraints are well-known as NP-hard problems [22]. Moreover, the applications with real-time criteria on the reconfigurable MPSoC still need more effective solutions in terms of computation time. Given this challenge, recent works have tried to find efficient solutions for these mapping and scheduling problems by using different tools and models. Some of them have been adopted to solve the scheduling problem on homogeneous multiprocessor computer systems such as heuristic approaches [24], evolutionary approaches [25] or hybrid methods [26]. Similarly, mathematical models have been proposed for the same problem to optimize the order of computations by using linear programming (LP), which can be solved by CPLEX [19, 28]. Unfortunately, the literature considering the same heterogeneous scheduling problem taking into account the access cost and rate communications is reduced to few mathematical models. More precisely, a specific model has been proposed by taking into account the communication delays in a heterogeneous implementation structure. Thereby, a linearization is performed on communication constraints to get a linear model without any other variables. Consequently, this model has been improved by minimizing variables in order to get the reduced linear model and then to allow a significant reduction of its size. However, the main drawback of these methods is the large running time, which is required to explore the search space. On the opposite side, heuristic methods have the advantage of achieving quickly an acceptable solution by limiting the exploration to a reduced part of the search space. Among them, we find the meta-heuristics, which have been proven to be effective in solving several machine schedul-

ing problems. Most of them are inspired by natural processes [22] such as but not limited to simulated annealing [32], Tabu search [33], genetic algorithms [34], ant colony [35] and particle swarms [9, 36]. Despite the interest of these approaches, we observe that no meta-heuristic has been proposed in the literature to solve the scheduling problem we address in this paper [20] (i.e., considering resources with heterogeneous architecture and communication delay constraints). Thus, this work can be an interesting attempt to investigate the performance of meta-heuristics for solving the studied problem. Our choice is restricted to the genetic algorithms and it is motivated by different reasons. Indeed, genetic algorithms (GAs) [23, 27] are defined as a class of robust stochastic search algorithms that are used to solve various optimization problems [15]. According to some surveys, the genetic algorithms can produce feasible solutions of equivalent or better performances compared to other meta-heuristic techniques for similar optimization problems [5]. These performances include the solution quality and the computation time. To summarize, the problem addressed in this paper is challenging and it is motivated by real applications. It takes into account the heterogeneous architecture and the communication delay constraints. Only few works exist on this topic and they are focused on mathematical models (see [20] and [19]). These models' capacities are limited to small instances. Taking into account the practical advantage of the genetic algorithms as robust meta-heuristics, it appears that their investigation can be very helpful to deal efficiently with large instances of the studied problem.

In this paper, we focus on metaheuristic approaches for solving the considered scheduling problem. The proposed method aims at providing optimal or near optimal solutions in a short running time while ensuring the specific performance criteria such as makespan, load balancing and so on. In our case study, the main challenge is to assign and schedule the tasks on the available resources in the heterogeneous architecture, in order to optimize the makespan performance criterion. Such a consideration is motivated by different applications' requirements (real time delivery, workload balancing, energy consumption, etc.) [21]. The originality of our approaches is mapping and scheduling tasks to the available resources in the FPGA/CPUs architecture in order to ensure the minimization of the makespan criterion considering one shared communication bus.

The remainder of this paper is organized as follows. Section 2 illustrates and presents the considered target reconfigurable heterogeneous architecture and the problem formulation based on a reduced mathematical model. Section 3 describes the proposed genetic algorithms for the considered scheduling problem taking into account the mentioned constraints. Experimental results aiming to evaluate the efficiency of the proposed approaches in terms of makespan and computation time are presented and discussed in Section 4. Finally, some general conclusions and further research directions are summarized in Section 5.

2. Problem description and existing models

This section is intended to give an overview about the main problem and to describe it precisely. The notations of the task graph and the target architecture are presented by considering one specific example of our problem.

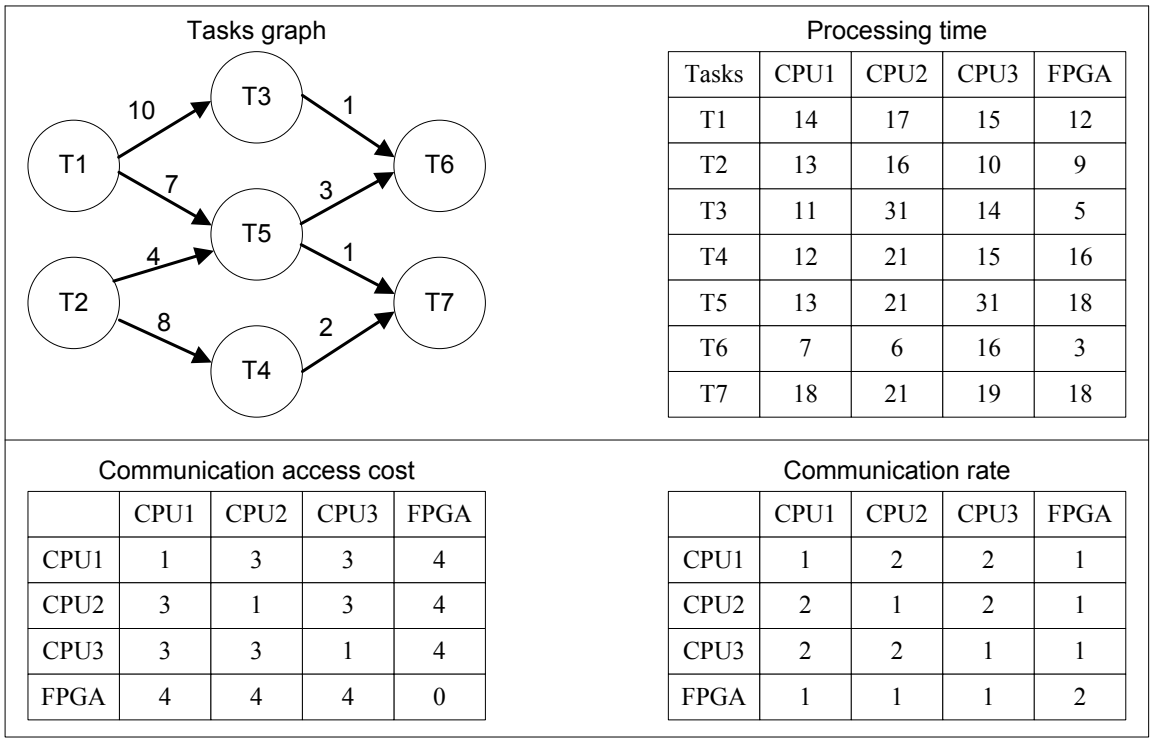
2.1. Position and description of the problem

The main problem is how to map the tasks of an application described in a data flow graph (DFG) form into a heterogeneous CPU/FPGA architecture [19]- [20].

The heterogeneous architecture we consider in this paper is a computing system, which consists of a set of CPUs cores and one FPGA chip linked by different means of communication. Thereby, the CPU cores are linked to each other by a shared memory while FPGA communicates with the CPU cores via one shared bus. An illustrative example is presented in Fig. 1 giving an overview of the problem. The first part of the Fig. 1 presents an application that consists of seven tasks to be executed on three CPUs and one FPGA where the processing time required for each task in each computing unit is also mentioned in processing time table in Fig. 1. The annotation DFG allows us to consider the condition of the data dependencies. For instance, the link from tasks T2 to T4 indicates that the task T4 requires a data size of 8 from task T2, before starting the execution. The time cost to exchange this amount of data depends on the connection between the both processing units where are assigned the tasks. The second part of Fig. 1 specifies "Communication Access cost" and "Communication rate" tables giving respectively the rate and time unit access costs. These tables define the total fixed time required to start the communication between the processing units (CPUs 1 to 3 and FPGA), and the data communication rate (number of data units exchanged per unit of time) between the processing units.
Fig. 2 shows two solutions of the considered scheduling problem (see Fig. 1) thanks to Gantt charts. In these schemes, we have highlighted the communication effects versus the scheduling solutions. We note that in "Solution 1" the delay to send eight data units from T2 to T4 in the FPGA is 0 + 8/2 = 4 time units (access + communication costs). The running time of the application (tasks graph in the Fig. 1) obtained for both solutions (Solution 1 and Solution 2) in the considered architecture based on three CPUs and one FPGA is 54.5 and 59 time units, respectively. This difference depends on the mapping of application on the target heterogeneous architecture. Indeed, to find the best solution, the mapping should be very effective.

In the remainder of this paper, we assume that the tasks are not preemptive, and we consider the following common notations:

- N: Set of n tasks;
- M: Set of m processing units (CPUs/FPGAs);
- $G = (N, A)$: A given directed acyclic graph, where N is the set of tasks and A is the set of arcs representing the precedence between tasks, i.e. (i,j) in A means that task i must be performed before task j;



Processing time

Tasks	CPU1	CPU2	CPU3	FPGA
T1	14	17	15	12
T2	13	16	10	9
T3	11	31	14	5
T4	12	21	15	16
T5	13	21	31	18
T6	7	6	16	3
T7	18	21	19	18

Communication access cost

	CPU1	CPU2	CPU3	FPGA
CPU1	1	3	3	4
CPU2	3	1	3	4
CPU3	3	3	1	4
FPGA	4	4	4	0

Communication rate

	CPU1	CPU2	CPU3	FPGA
CPU1	1	2	2	1
CPU2	2	1	2	1
CPU3	2	2	1	1
FPGA	1	1	1	2

Figure 1: Illustrative description of the considered scheduling problem.

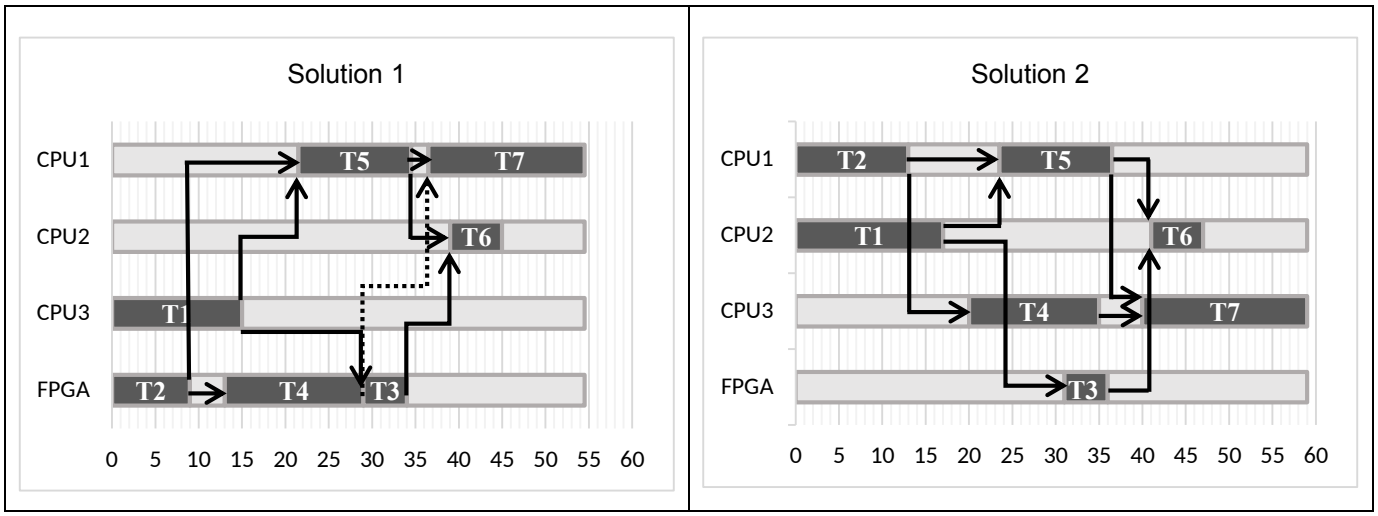


Figure 2: Proposition of two solutions for the same scheduling problem.

- *Pred (i)*: Set of tasks that precede task i;
- N^+: Set of tasks with no successors (output of graph G);
- *P(i)*: Set of tasks j such that i and j belong to the same path;
- t_{ik}: Processing time of task i on processing unit k;
- $c_{ik,jl}$: Cost of direct communication between task i on processing unit k and task j on processing unit l;
- F_k: Set of tasks that should not be assigned to the processing unit k;

The communication depends on the hardware architecture used such as shared memory, shared bus, hierarchical bus, network on chip, Ethernet links, and so on. The network architecture has also an impact on the communication. Generally, the communication cost is non-linear and will be defined as follows: $c_{ik,jl} = a_{kl} + \frac{d_{ij}}{r_{kl}}$ where a_{kl} is the fixed communication cost between processing units k and l, d_{ij} is the size of data sent from the task i to task j, and r_{kl} is the communication rate between the two processors k and l. We assume that $a_{kl} = a_{lk}$ and $r_{kl} = r_{lk}$.
For the decision part, the assignments and starting times are associated to variables δ_{ij}, x_{ij} and s_i. These decision variables

will be used in the mathematical model of the next subsection. Finally, it is important to mention that the objective is to minimize the completion time of the last task or the makespan (C_{max}).

2.2. Ait El Cadi et al.'s model [19]

The studied problem can be associated to the reduced linear model proposed by Ait El Cadi et al. [19]. This model will be compared to the algorithms proposed in the next sections of this paper. Note that the reduced model is interesting and it could handle in average the instances with size up to 50 tasks and 5 CPU/FPGA units in few seconds. The correctness proof of this model is detailed in Ait El Cadi et al. [19]. Moreover, different improvements have been introduced the mentioned reference. Thus, the comparison of our algorithms to the performances of this reduced model seems to be an important and interesting attempt. The reduced linear model is shortly described in Appendix A. More details on its proof and its performance can be found in Ait El Cadi et al. [19]..

3. Proposed approaches for the scheduling problem

Many surveys report the results of comparisons between various meta-heuristic techniques for different optimization problems (GAs, ant colony, particle swarm, simulated annealing and tabu search,...), and these comparative surveys show that GAs provide good trade-off between the rapid convergence and the computation time [38]. These references are related to different optimization problems (circuit partitioning, path planning, scheduling, etc.) and they affirm that GAs can produce solutions equivalent or better than the other metaheuristic approaches while needing shorter computation times [38, 39, 40]. Therefore, this observation motivates naturally the exploitation of genetic algorithms to solve the considered scheduling problem in this paper.

Our goal is then concentrated on the adaption of the genetic algorithms in order to implement an effective scheduling algorithm for the considered problem. For this reason, we aim to run an application presented in the form of a data flow graph into a heterogeneous architecture in order to find the best compromise between minimizing the makespan (C_{max}) and the computation time (Time) required by the algorithm. In this section, after briefly presenting the standard genetic algorithms, we detail two proposed approaches; the *Genetic Algorithm Approach* (*GAA*) and the *Modified Genetic Algorithm Approach* (*MGAA*) focusing on the makespan minimization. In the remainder of the paper, we also consider the following common notations:

- *Succ (i)*: Set of successor tasks of task i;
- N^-: Set of tasks with no predecessors (i.e., the input of graph G);
- *FTP(k)*: Completion time of the last task performed on processor k;
- *AFT(i)*: Actual completion time of task i;

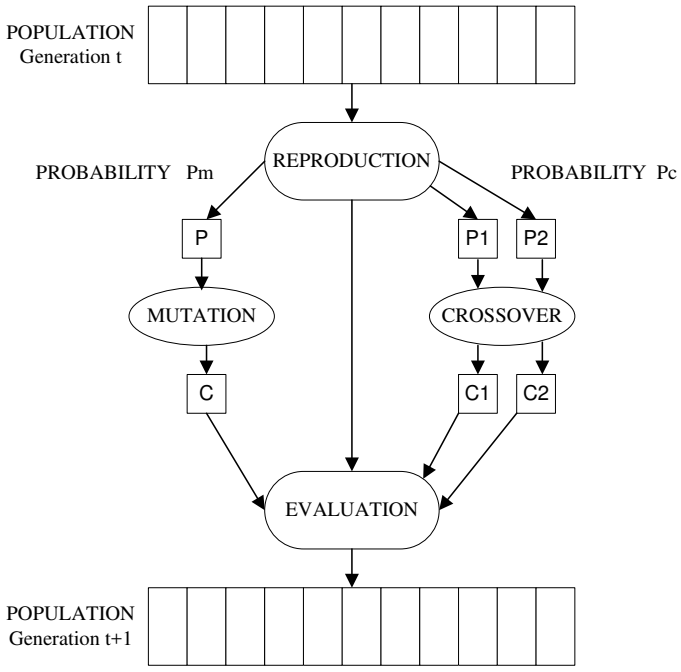


Figure 3: Flowchart of a standard GA.

- *PopSize*: Number of chromosomes in the population (size of the population);
- *NG*: Number of generations, used as a stopping criterion for the genetic algorithms (GAs);

3.1. Standard GAs

GAs were originally proposed by Holland in 1975 [29]. In discovering good solutions to difficult problems, these algorithms, based on the the biological evolution process, have been rapidly and successfully applied to solve the combinatorial optimization problems in Operations Research and learning problems in the field of Artificial Intelligence [29]. Thus, GAs may be particularly useful in the following areas: Optimization (functions, planning, scheduling, etc.), Learning (classification, prediction, robotics, etc.) , Automatic Programming (LISP programs, cellular automata, etc.) or study of the living and real world (economic markets, social behavior, immune systems, etc.) Portmann, introduces and explains how to apply these algorithms to scheduling problems [31]. A standard GA is based on populations of solutions. Initially, a population is created by a certain heuristic or random procedure. Then, the GA generates at each iteration other solutions, which tend to be better, by combining some chromosomes, i.e., solutions, using genetic operators for selection, crossover and mutation. In the context of the application of GAs to combinatorial optimization problems, an analogy is developed between an individual (chromosome) in a population and a feasible solution of the considered problem. Currently, GAs are based on the same fundamental algorithm structure as shown in Fig. 3.

We start by creating a random initial population of individuals (chromosomes). To pass from a generation t to the next

generation t+1, the following three operations are applied to the individuals of the population of generation t. Parents P1 and P2 are selected according to their adaptation levels (or the values of the associated solutions). The crossover operator is applied to them with a probability Pc (usually around 0.6) and it generates pairs of children C1 and C2. Other elements P are selected according to their adaptations (or their solutions' values) and the mutation operator is applied to them with a probability Pm (Pm is generally much lower than Pc). This mutation leads to the mutated individuals C. The adaptation level of children (C1, C2) and mutated individuals C are then evaluated before insertion into the new population.

To summarize, the different steps of a standard GA can be listed as follows:

- Generation of the initial population.
- Selection.
- Crossover and mutation.
- Replacement by the new population.

3.2. Genetic algorithm approach (GAA)

The GAA approach inherits almost the same steps of a standard GA. Fig.4 illustrates the different typical phases of GAA. After the generation of the initial population, the standard operators are successively performed in order to obtain the best chromosomes corresponding to the smallest makespan. The following subsections explain the concepts and describe the ordered GAA steps as depicted in Fig. 4.

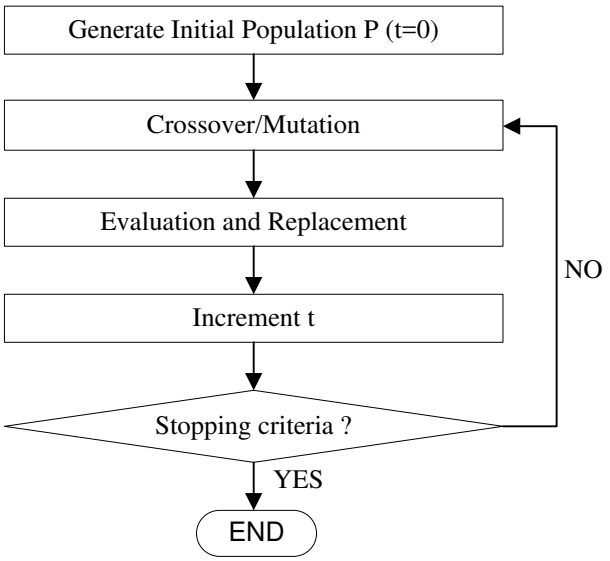


Figure 4: GAA steps.

3.2.1. Chromosomes and assignment strategies of tasks

In the case of our considered scheduling problem, the chromosome represents a combination of tasks and processors as shown in Fig. 5. More precisely, each task in the first part of the chromosome is linked to the corresponding processor assignment. For example, task T_j is processed on CPU 2 (index 3). Therefore, we consider that a chromosome is composed of

number of tasks = 8 and number of processors = 3

	1	2	3	4	5	6	7	8
Task List	T_h	T_i	T_j	T_l	T_n	T_m	T_s	T_k
Processor List	1	1	2	1	2	1	3	2

Figure 5: An example of solution representation.

a sequence of n tasks (Task List) and an assignment (Processor List). In addition, the sequence should be feasible in order to respect the dependencies between tasks. Consequently, a sequence is considered to be feasible if and only if:

1. Each task belongs to one and only one position in this sequence (Task List).
2. The order of tasks in a given sequence must respect the data dependencies between them, such as: if T_i precedes $T_j \Rightarrow T_i$ is mentioned before T_j in the sequence.

In order to measure each individual adaptation, at every iteration of the genetic algorithm (GAA or MGAA), the evaluation function calculates the makespan of the corresponding chromosome in the current population. The proposed pseudocode of the evaluation function is given in Algorithm 1. In this approach, we assign the tasks of a sequence on the heterogeneous architecture according to one of two following strategies:

Algorithm 1: Evaluation of chromosome.

Input : Task List TL and Processor List PL
`/* Chromosome = Task List + Processor List */`
Output: Makespan (C_{max})

1 **for** $p = 0; p < m; p$++ **do**
2 $FTP(p) = 0$; `/* rest FTP for all processors */`
3 **end**
4 **for** $c = 0; c < n; c$++ **do**
5 $i = \mathrm{TL}[c]$; `/* i is the task indexed by c */`
6 $k = \mathrm{PL}[c]$; `/* k is the processor indexed by c */`
7 val = 0;
8 **if** $Pred(i) \neq \emptyset$ **then**
9 val = $max_{j \in Pred(i)} \{AFT(j) + c_{jl,ik}\}$ `/* l is the index of processor where the task j is assigned */`
10 **end**
11 $FTP(k) = max\{FTP(k), \mathrm{val}\} + t_{ik}$;
12 $AFT(i) = FTP(k)$;
13 **end**
14 $C_{max} = max_{p \in [1,m]}\{FTP(p)\}$;

- Strategy A: Usually, we assign the tasks according to their indices/positions in the sequence. Thus, a task with the index I of a given sequence is assigned to the processor (I-1) [m] +1, where [ ] refers to the modular arithmetic. This strategy is illustrated in Fig. 6 which shows an example of sequence assignment by considering 8 tasks in an architecture of four processors (m = 4). Thus, the task T_k is assigned to the processor (8-1) [4] +1 = 4.

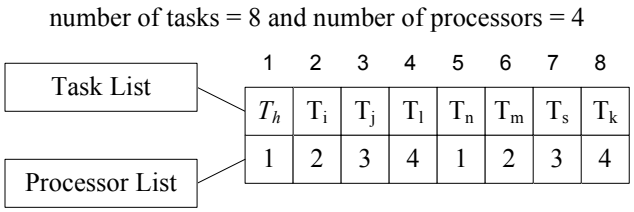


Figure 6: Encoding of chromosome for the GAA under the strategy A.

- Strategy B: this strategy consists simply in assigning each task in the sequence, according to its order of appearance, to its "best" processor, on which the task completion time is minimized.

3.2.2. Initial population

The choice of the first population is an important parameter in the success of the search strategy. Typically, several methods exist to define this initial population, which represents the starting point for the constitution of future generations such as random selection, heuristics, or a combination of heuristics and random solutions [1, 2].

Algorithm 2: Generates the initial population randomly.

Input : Positive integer *PopSize* and empty population P.
Output: Population P

```
1  k = 0
2  Generate a first chromosome;
3  while k< PopSize do
4      if feasible chromosome then
5          if chromosome does not exist in the population
             P then
6              chromosome k accepted;
7              k = k+ 1;
8          else
9              Denied chromosome;
10         end
11     else
12         Denied chromosome;
13     end
14     Generate another chromosome;
15 end
```

The selection of the population size is decisive for the efficiency of the algorithm. As an example, a large population increases the chances of finding optimal or near optimal solutions, but causes an increase of the computation time. Indeed, the choice of the population size is still an open problem and it requires to be fixed by experimental tests.

The solutions in the initial population are randomly chosen in our study. Algorithm 2 describes the proposed pseudo-code that creates the initial population containing *PopSize* chromosomes (*PopSize* feasible sequences).

3.2.3. Crossover operator for Task List

The crossover operator is the most important step in the GAs. It can effectively explore the search space in order to ensure the search intensity. There are several modes of crossovers: 1-point crossover, 2-points crossover, uniform crossover, crossover 1.X, etc. In our case, we are interested to implement our approach by using the crossover 1.X where the cross between two feasible chromosomes (parents) leads to two feasible chromosomes (children). The principle of this operator is described as follows:
Let E1 and E2 be two feasible chromosomes (children) obtained by crossing two feasible chromosomes (parents) P1 and P2, and let p be a randomly selected crossover point (see Fig. 7):

- The child E1 receives the same genes as P1 between position 1 and position p; the rest will be completed according to the order of the missing genes in P2.
- The child E2 receives the same genes as P2 between position 1 and position p; the rest will be completed according to the order of the missing genes in P1.

We consider a problem of six tasks with the following precedence constraints: $\{1 \rightarrow 6\}$, $\{2 \rightarrow 4\}$, $\{5 \rightarrow 3 \rightarrow 6\}$ to illustrate the Crossover 1.X operator in Fig. 7.

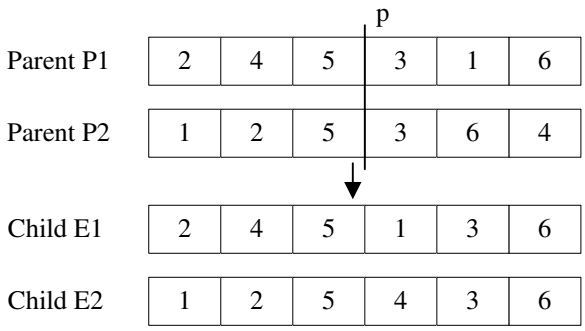


Figure 7: Crossover 1.X operator.

3.2.4. Mutation operator for Task List

There are many ways to apply the mutation to a given chromosome. In our case, due to the precedence constraints, we propose a mutation operator that allows us to create feasible chromosomes as shown in Fig. 8 considering the same case used to explain the Crossover 1.X operator. The required steps for this operator are the following:

- Determination of the set of pairs of genes that can be exchanged between them in a given chromosome without losing the feasibility of the modified chromosome. For example, the set E for Parent 1 is E= {(2,5), (2,1), (5,1), (5,4), (1,4), (1,3), (4,3)}.
- Among the pairs found in the set E, take a random element e= (a, b) and exchange the two genes "a" and "b" in the same chromosome A to obtain a new feasible chromosome A'. For instance, if we consider Parent 1 (feasible chromosome) and assume that the pair (5,4) is selected, then we get the new feasible chromosome (Child 1) as described in Fig. 8.

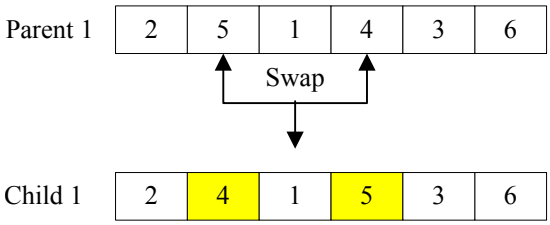


Figure 8: Mutation operator.

3.2.5. Evaluation and replacement

The application of the two previous operators (*PopSize/2* crossover and *PopSize* mutation) to the current population P(t) in the previous step leads to a population P'(t). This step allows us to constitute the population of the next generation P(t+1) from the parents P(t) and the children P'(t) of the current generation (see Fig. 9). A fraction of the population is replaced by its offspring in each generation. Specifically, an evaluation phase based on Strategy A or Strategy B is integrated to calculate the makespan for all the chromosomes of the populations P(t) and P'(t). Then, we select the best *PopSize* chromosomes among them to constitute the population P(t+1) of the next generation.

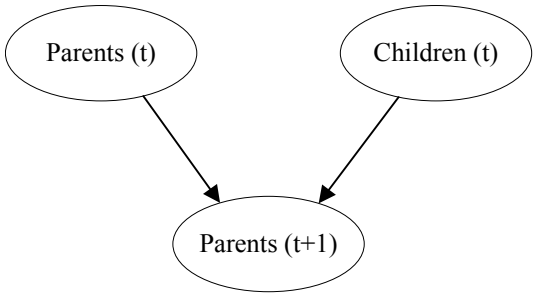


Figure 9: Evaluation and replacement.

3.2.6. Stopping criterion

Different stopping criteria of the algorithm can be selected as follows:

- The popular one is to fix the number of iterations (NG) to be performed, in order to find a solution within a limited computation time.

Algorithm 3: Generates *Nb* different feasible sequences.

Input : Positive integer *Nb*, Three empty sets *A*, *B* and *T*
Output: Set of sequences *T*

```
/* Create the first feasible sequence */
K = 0;
while N⁻ ≠ ∅ do
    /* Add all the tasks with no predecessors to the sequence Seq */
    Select a task i from N⁻;
    Seq[k] = i;
    N⁻ = N⁻ \ {i};
    K = K + 1;
end
I = 0;
while I < K do
    A = Succ(T[I]);
    while A ≠ ∅ do
        Select a task i from A;
        B = Pred (i);
        J = 0
        V = 0;
        while J < K do
            if i = Seq[J] then
                V = -1;
                break;
            end
            if Seq[j] ∈ B then
                V = V + 1;
            end
            J = J + 1;
        end
        if V == Card(B) then
            Seq[k] = i;
            K = K + 1;
        end
        A = A \ {i};
    end
    I = I + 1;
end
T = T ∩ {Seq}; /* The first feasible sequence Seq is added to the empty set T. */
Count = 1;    /* Count is the cardinality of the set T */
while Count < PopSize do
    /* Creation the PopSize-1 feasible sequences remaining using the mutation operator of GAA (see Fig. 8)    */
    Create a sequence Seq' by applying the mutation operator of GAA to the sequence Seq;
    if Seq' ∉ T then
        T = T ∩ {Seq'};
        Count = Count + 1;
    end
end
```

- The algorithm can be stopped when the population no longer changes or when the change is not significant.
- The algorithm can be stopped when chromosomes converge to one or more satisfactory solutions.
- The algorithm is run for a predetermined period and it gets the result.

In our study, the algorithm is performed for a fixed number of iterations and it returns the best obtained chromosome as the final solution. However, it is not obvious to fix this number of generations.

3.3. Modified genetic algorithm approach (MGAA)

Although the previous proposed approach can produce a promising solution within a reasonable amount of time, it has some limitations. This is due to the fact that for every task sequence there is only one mapping to the target architecture by Strategy A or Strategy B. To overcome this problem, we have proposed a new modified version of GAA called Modified Genetic Algorithm Approach (MGAA). This approach exploits further the search space to in order to find better solutions. More precisely, we explore the possible assignments for each feasible sequence to find a good mapping and hence to get a satisfactory couple (Tasks List, Processor List). Consequently, a solution or an individual is encoded by an N*2 matrix to provide a set of tasks and a random assignment.

Fig.10 illustrates the different typical phases of the MGAA. First, we create *Nb* different feasible sequences in a pre-treatment step. To create feasible sequences, we consider the algorithm 3. More precisely, the feasible sequences are randomly created. For this purpose, we create a first sequence by a uniform search based on the Breadth-First Search (BFS) method [41]. We have considered this blind search because we seek to obtain one first feasible sequence with no information about the search space. Then, the mutation is randomly performed. Indeed, the creation of the *PopSize*-1 remaining sequences uses randomly the mutation operator of GAA as described in Fig. 8. We create a thread for each sequence among these *Nb* sequences (*Nb* threads executed in parallel having the same steps and parameters, *PopSize* and *NG*), which allows us to explore some good assignments for each sequence. At the end of these *Nb* threads, we obtain *Nb* solutions (one solution for each thread). The best solution among these *Nb* solutions is selected as a final solution for this approach. We note here that the phases of "Evaluation and Replacement" and "Stopping criterion" are already explained except the evaluation of a chromosome, which is performed according to the sequence of tasks and the problem constraints. The remaining phases of MGAA are detailed in the next subsections.

3.3.1. Initial population

The initial population is created with *PopSize* random chromosomes (solutions) where each chromosome has the same feasible sequence Seq and a random assignment (detailed in Algorithm 4).

Algorithm 4: Generates the initial population randomly.

Input : Positive integer *PopSize*, feasible sequence Seq and empty population P
Output: Population P

```
1 t = 0;
2 while t < PopSize do
3 |   Generate a random assignment A for Seq;
4 |   Add the solution S = (Seq, A) to the population P;
5 |   t = t + 1;
6 end
```

3.3.2. Crossover and mutation operators for Processor List

The crossover and mutation operators are applied in this approach to the assignment properties of chromosomes (no modification in the sequences of tasks). We use the "Crossover point" to cross two parents' chromosomes, which leads to two new children chromosomes. An illustrative example of this operator for an instance of six tasks and four processing units is shown in Fig.11. A single crossover point on both parents is selected. Child 1 is produced by taking the initial part of the assignments in Parent 1 and the remaining part from Parent 2. Similarly, for Child 2, the initial part is taken from the assignment part of Parent 2 and the rest is taken from the assignments contained in Parent 1. Childs 1 and 2 take the same sequence of tasks (Task List) as in parents 1 or 2.

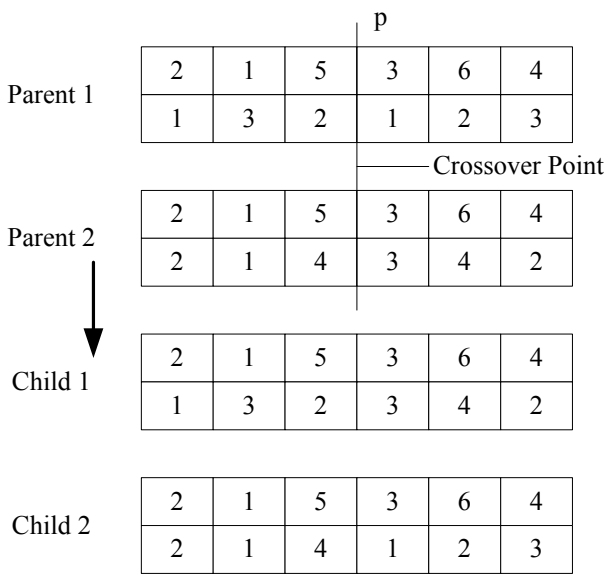


Figure 11: one-point crossover operator.

The mutation operator used in this approach aims at selecting a random cell from the assignment part of a chromosome and it consists in assigning a random value between 1 and m. Fig. 12 presents an example of one mutation where $n = 6$ and $m = 4$.

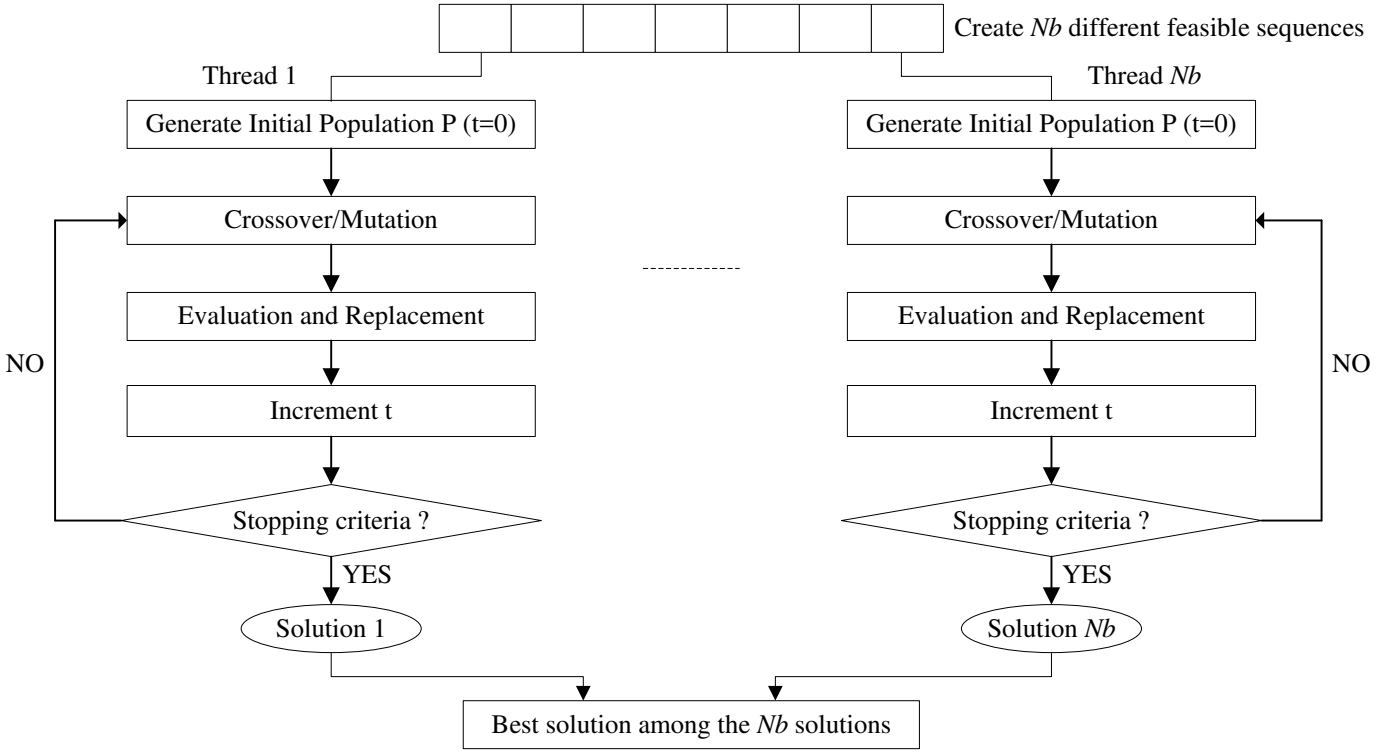


Figure 10: Steps of MGAA.

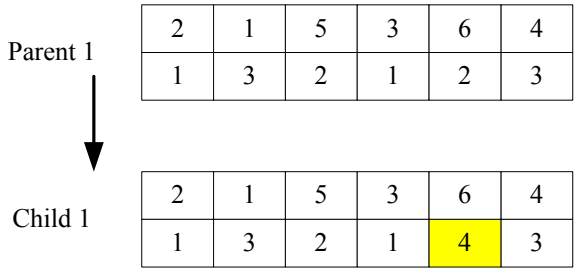


Figure 12: Point mutation operator.

4. Computational results

The results of our experiments are presented in this section in order to evaluate and to compare the two proposed approaches GAA and MGAA. In addition, a comparison of our algorithms with the reduced mathematical model proposed in [19] is performed to demonstrate the effectiveness of our work, in particular in terms of computation time.
All the computational tests are conducted on a computer equipped with 8 Intel processor i7-4700MQ cores and 8GB RAM memory. The operating system is a 64-bits Windows 8 Pro. To solve the mathematical model, described in Section 1, we used the IBM ILOG CPLEX V12.6 Optimization Studio and JAVA language. On the other hand, we implemented GAA and MGAA by using the C language.

4.1. Instances

In order to compare the performance of GAA, MGAA and the reduced integer linear model, we use 7 datasets (sets of n tasks, where n varies from 5 to 30, and the sets of edges have their cardinality values from 4 to 67). These datasets are described in Table 1, in which we list the name and the description of each Dataset. The value m is the number of the processing units; n represents the number of tasks to be scheduled and $|A(G)|$ is the number of edges in the corresponding graph G. In this set of data, the network communication is described by two matrices ($m \times m$), as depicted in Fig. 1, which contains the values of the communication rate and the fixed costs. The instances are defined by three matrices generated randomly: (1) The first one is $n \times m$ matrix for the processing times, as illustrated in Fig. 1. It gives for each task its processing time if it is performed on a given processing unit. (2) The second one is the adjacency matrix of $n \times n$. (3) The third $n \times n$ matrix contains the amount of the exchanged data between tasks. For each dataset (according to their number of tasks and number of edges), we randomly generate ten instances with a number of processing units m equal to 4 (three CPUs and one FPGA), i.e., we consider for our experiences a total 70 instances in overall. We use the same network description for all the instances of our datasets. From the ten instances of each dataset, we calculate the average values of makespan $\overline{C_{max}}$ and computation time $\overline{Time}$.

4.2. Effect of the population size and the number of iterations (generations)

The determination of GA and MGA parameters (*PopSize*, number of iterations/generations *NG*, number of threads *Nb*) in order to improve both computation time and solution quality is

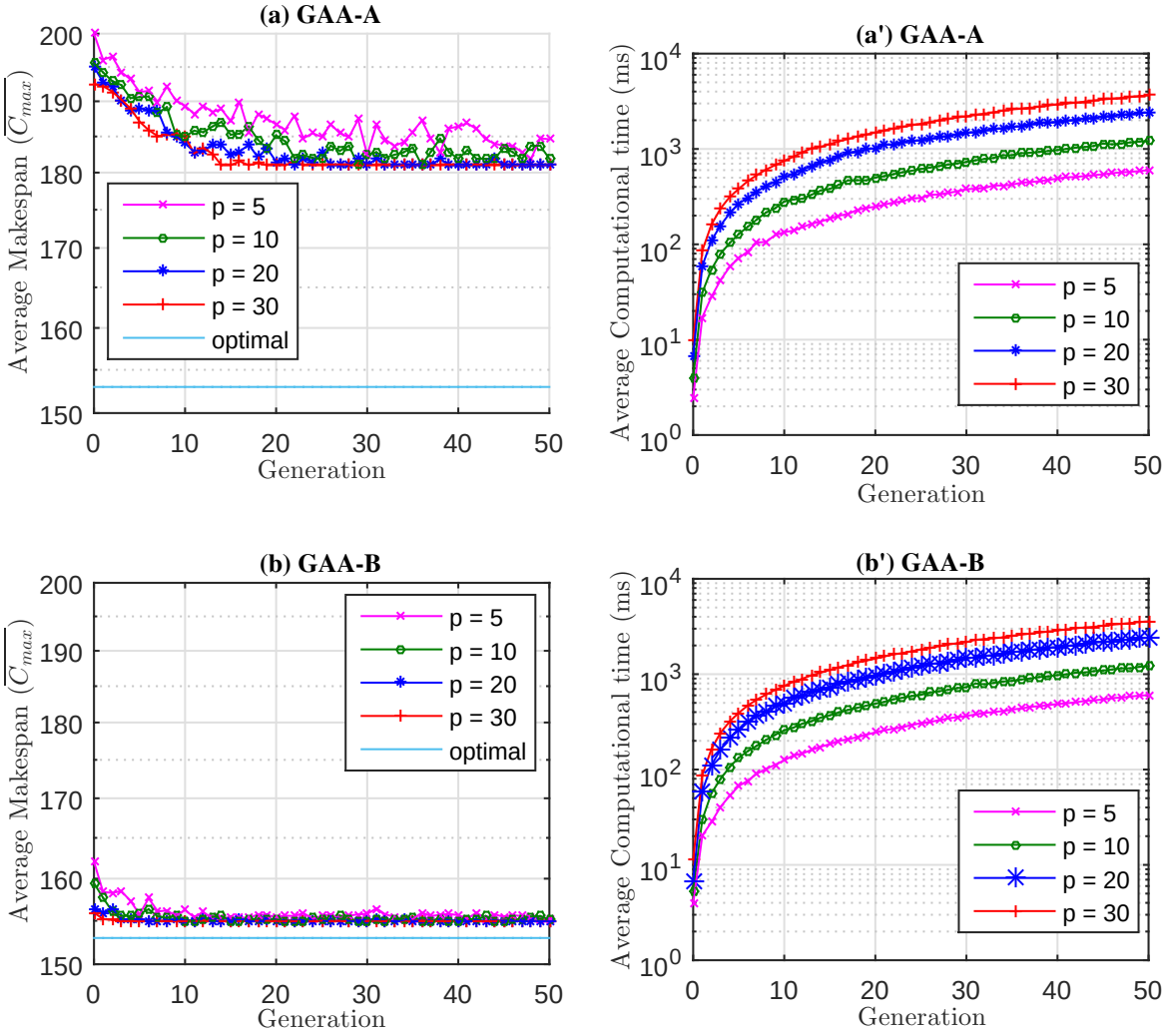


Figure 13: GAA performance for various populations' sizes *PopSize*.

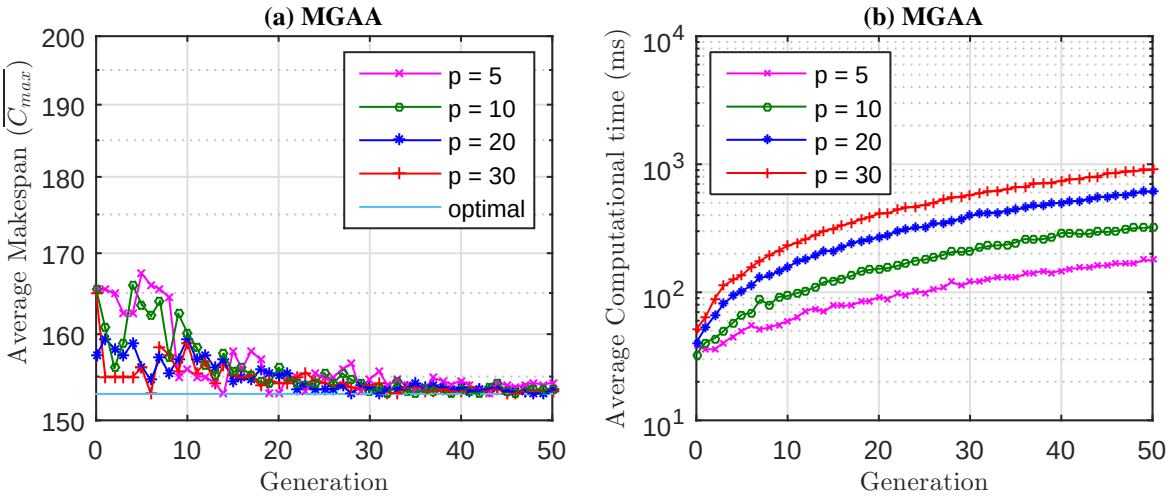


Figure 14: MGAA performance for various populations' sizes *PopSize*.

Table 1: Datasets used to compare the proposed approaches and the integer linear model.

Dataset	m	n	$\|A(G)\|$
Dataset 1	4	5	4
Dataset 2	4	10	18
Dataset 3	4	15	10
Dataset 4	4	15	25
Dataset 5	4	20	29
Dataset 6	4	25	30
Dataset 7	4	30	67

not a trivial task. To be able to compare the effect of the population size and the number of generations on our proposed algorithms, different population sizes (from 5 to 30 chromosomes in the population) are used, and the maximum number of generations are chosen from 0 (initial generated solutions without running the algorithm) to 50. The number *Nb* of threads used in MGAA is *Nb* = 15. Because of their stochastic characteristics, our proposed algorithms were run 10 times for each population size and generations' number. Figures 13 and 14 show respectively the results in terms of makespan and average computation times for GAA and MGAA. In these figures the same problem instance was considered based on 15 tasks and 20 edges with a number of processing units m equal to 4 (3 CPUs and 1 FPGA). We note here that the optimal makespan for this problem is equal to 153.00 time units. Figures 13 and 14 contain two plots. One represents the average values of makespan $\overline{C_{max}}$ (see Figures 13(a) and 13(a')) for the considered population sizes, and the other one shows the average results observed for the computation time.

The results presented in the figures 13(a), 13(b) and 14(a) show that increasing the population size from 5 to 30 chromosomes allows us to improve significantly the makespan values and that fewer generations are required to converge. In Fig. 13(a), only 14 generations are required to reach the best average value of makespan ($\overline{C_{max}}$ = 181.00) obtained with the GAA considering the "Strategy A" (GAA-A) when *PopSize* = 30, while 22, 29 and more than 50 generations are respectively required when *PopSize* = 20, *PopSize* = 10 and *PopSize* = 5. The same observation can be made in Fig. 13(b) for the GAA considering Strategy B (which is denoted by GAA-B), where only 3 generations are required to reach the a makespan average value of 155.00 when *PopSize* = 30, while 6, 10 and more of 50 generations are required to reach a same average value when *PopSize* = 20, *PopSize* = 10 and *PopSize* = 5, respectively. Otherwise, the optimal makespan value of 153.00 is obtained with MGAA for all the population sizes (from 5 to 30) as shown in Fig. 14(a).

Moreover, increasing the population size or the number of generations causes an increase of the computation time (see Figures 13(a'), 13(b') and 14(b)). Indeed, when the population size rises, it causes an increase of the required computation and memory times, which can be a problem for the large-scale tests. To overcome this problem, we choose the best associated value pair of makespan and computation time as one solution among all the results obtained with each approach. More precisely, for example, the best pair of makespan and computation time average values in the GAA-A is $\overline{C_{max}}$ = 181.00 time units, $\overline{CPUtime}$ = 703.9674 ms which corresponds to the following coordinates (number of generations *NG* = 29, population size *PopSize* = 10) in the figures 14(a) and 14(a'). Similarly, in the figures 14(b), 14(b') and 15, the two best pairs of makespan and computation time in terms of average values in the GAA-B and MGAA are ($\overline{C_{max}}$ = 155.00 time units, $\overline{CPUtime}$ = 139.2433 ms) and ($\overline{C_{max}}$ = 153 time units, $\overline{CPUtime}$ = 70.2007 ms), respectively.

Considering the experimental results, we can clearly confirm that the performances of GAA and MGAA in terms of computation time and solution quality depend mainly on the used parameters. In the rest of this paper, in order to provide a better trade-off between computation time and makespan, we apply the proposed approaches for all instances 10 times for each population size and generations' numbers by considering the population sizes from 5 to 30 chromosomes, a number of iterations NG from 0 to 200, and a number *Nb* of threads used in MGAA equal to 80.

4.3. Comparison with the reduced mathematical model

In this subsection, we present a comparison between the results obtained from our proposed approaches GAA-B and MGAA with the reduced mathematical model, previously mentioned (Ait El Cadi et al.'s model [19]). This comparison aims to confirm the performances of our proposed GAs compared to the exact reduced model.

Table 2 gives the comparison results between GAA-A and the reduced mathematical model in terms of $\overline{C_{max}}$, $\overline{Time}$ and the improvement rates ($Imp1$ and $Imp2$) expressed in percentages and representing the improvements of $\overline{C_{max}}$ and $\overline{Time}$ between the mathematical model and GAA-A. We observe that GAA-A provides a better performance in terms of running time. Indeed, we obtained a significantly reduced average computation time (with an improvement in average computation time greater than 53 %) while the obtained makespan values decrease in average about 24.3 %. Hence, the GAA-A does not provide better solutions compared to reduced mathematical model but it is much faster than this exact model in terms of computation time. In summary, the GAA-A gives reasonable solutions for the largest instances in a very short computation time.

Table 2: Comparison between GAA-A and the reduced mathematical model (average values).

Dataset	Reduced model		GAA-A		Improvement (%)	
	$\overline{C_{max}}$	$\overline{Time(s)}$	$\overline{C_{max}}$	$\overline{Time(s)}$	$Imp1$	$Imp2$
Dataset 1	63.80	0.0843	94.25	0.0092	-47.72	89.09
Dataset 2	123.10	0.2996	175.69	0.1946	-42.72	35.05
Dataset 3	81.85	3.7363	97.28	1.7486	-18.85	53.20
Dataset 4	123.30	1.2931	156.47	1.3989	-26.90	-8.18
Dataset 5	130.55	9.0510	147.90	5.9955	-13.29	33.76
Dataset 6	131.95	33.8473	145.85	8.2772	-10.53	75.55
Dataset 7	181.30	565.9699	199.50	23.2937	-10.04	95.88

Similarly, Table 3 shows the comparative results between GAA-B and the reduced mathematical model. We observe that

the GAA-B ensures a good performance compared to the reduced mathematical model. Indeed, the obtained makespan values by GAA-B are in average close to the obtained values of the reduced model and for all the datasets (with a relative gap less than 3.89 %). Moreover, considering all datasets, the average running times of GAA-B is clearly lower than those given by the reduced mathematical model (with an improvement rate at least greater than or equal to 42.35 % in all the cases). These results demonstrate the advantage of GAA-B compared to the reduced model in terms of computation time. Hence, the GAA-B provides satisfactory approximate solutions to our problem in a very reasonable running time.

Table 3: Comparison between GAA-B and the reduced mathematical model (average values).

Dataset	Reduced model		GAA-B		Improvement (%)	
	$\overline{C_{max}}$	$\overline{Time(s)}$	$\overline{C_{max}}$	$\overline{Time(s)}$	$Imp1$	$Imp2$
Dataset 1	63.80	0.0843	64.70	0.0053	-1.41	93.71
Dataset 2	123.10	0.2996	125.30	0.0992	-1.79	66.89
Dataset 3	81.85	3.7363	85.03	1.1959	-3.89	67.99
Dataset 4	123.30	1.2931	124.51	0.7455	-0.98	42.35
Dataset 5	130.55	9.0510	133.38	3.5829	-2.17	60.41
Dataset 6	131.95	33.8473	136.35	4.8654	-3.33	85.63
Dataset 7	181.30	565.9699	186.65	13.4329	-2.95	97.63

Table 4 compares the performances of the reduced mathematical model and MGAA in terms of $\overline{C_{max}}$, $\overline{Time}$ and the improvement rates ($Imp1$ and $Imp2$), as before. The obtained average running time from MGAA is clearly very less than the obtained values from the reduced model in all cases (improvement rate greater than or equal to 67.43 %). In addition, the MGAA provides optimal solutions in most of the cases. We note that for the instances with a large number of tasks, MGAA provides similar solutions compared to those of the mathematical reduced model. Moreover, in the case of Dataset 7, the obtained average running time from the reduced model is 565.9699 seconds. This value decreases to 8.4624 seconds when using MGAA, which means an improvement of 98.50 %. These results confirm the practical advantage of MGAA with respect to the mathematical reduced model, in terms of running time. Hence, MGAA is much faster than the reduced model, and it provides solutions close to the optimal one for solving our scheduling problem.

Table 4: Comparison between MGAA and the reduced mathematical model (average values).

Dataset	Reduced Model		MGAA		Improvement (%)	
	$\overline{C_{max}}$	$\overline{Time(s)}$	$\overline{C_{max}}$	$\overline{Time(s)}$	$Imp1$	$Imp2$
Dataset 1	63.80	0.0843	63.80	0.0053	0.00	93.71
Dataset 2	123.10	0.2996	123.10	0.0446	0.00	85.11
Dataset 3	81.85	3.7363	82.23	0.5562	-0.46	85.11
Dataset 4	123.30	1.2931	123.50	0.4211	-0.16	67.43
Dataset 5	130.55	9.0510	130.70	1.1997	-0.11	86.75
Dataset 6	131.95	33.8473	133.70	2.7778	-1.33	91.79
Dataset 7	181.30	565.9699	182.95	8.4624	-0.91	98.50

4.4. Comparison results between GAA and MGAA

In this sub-section, the performance analysis of GAA and MGAA is presented and compared. We describe the numerical results obtained by testing the two algorithms on the dataset defined in Table 3.

Table 5 compares the two strategies A and B of GAA. This comparison is carried out as before in terms of the same measures $\overline{C_{max}}$, $\overline{Time}$, $Imp1$ and $Imp2$. The results confirm the conclusion in the previous subsection and GAA-B outperforms GAA-A.For example, in the case of Dataset 2, the makespan value obtained in average by GAA-A is $\overline{C_{max}}$ = 175.69 time units within a computation time of 0.1946 seconds. These values decreased when using the GAA-B to $\overline{C_{max}}$ = 125.30 time units within a running time of 0.0992 seconds. Therefore, the improvement rate in terms of $\overline{C_{max}}$ is 28.68 %, and in terms of running time is 49.02 %, and these improvements are in favor of GAA-B. In addition, we found that the results obtained by GAA-B are better than those obtained by the GAA-A in all the datasets. Based on the obtained results, we can confirm that GAA-B is more efficient and faster than GAA-A.

Table 5: Comparison between GAA-A and GAA-B (average values).

Dataset	GAA-A		GAA-B		Improvement (%)	
	$\overline{C_{max}}$	$\overline{Time(s)}$	$\overline{C_{max}}$	$\overline{Time(s)}$	$Imp1$	$Imp2$
Dataset 1	94.25	0.0092	64.70	0.0053	31.35	42.39
Dataset 2	175.69	0.1946	125.30	0.0992	28.68	49.02
Dataset 3	97.28	1.7486	85.03	1.1959	12.59	31.61
Dataset 4	156.47	1.3989	124.51	0.7455	20.43	46.71
Dataset 5	147.90	5.9955	133.38	3.5829	9.82	40.24
Dataset 6	145.85	8.2772	136.35	4.8654	6.51	41.22
Dataset 7	199.50	23.2937	186.65	13.4329	6.44	42.33

Table 6 depicts the comparison between MGAA and GAA-B (the best strategy of GAA) according to the same criteria $\overline{C_{max}}$, $\overline{Time}$ and improvement rates ($Imp1$ and $Imp2$), defined as in the previous analysis. We notice that the makespan average and the running time are reduced with MGAA in all the cases. In particular, we can observe that in the case of Dataset 5, the obtained makespan value by GAA-B is in average equal to 133.38 time units and the results are obtained within a computation time of 3.5829 seconds. These values decrease when using MGAA to $\overline{C_{max}}$ = 130.70 time units within a running time of 1.1997 seconds. Thus, the improvement rates in favor of MGAA are 2.01 % (for the makespan) and 66.52 % (for the running time). To conclude, MGAA can provide very good approximate solutions little bit better than those obtained by GAA-B, but in a much shorter computation time.

In summary, the experimental results allow us to rank the proposed methods. Indeed, MGAA gives better solutions that GAA-B in terms of the makespan values and in particular the computation time required to solve a problem. On the other hand, GAA (under its assignment strategy B) provides satisfactory solutions within a reasonable computation time compared to the exact mathematical model. Therefore, our contributions based on these adapted GAs provide good trade-off between computation time and makespan compared to existing exact mathematical models, which can be of a real practical inter-

Table 6: Comparison of GAA-B and MGAA (average values).

Dataset	GAA-B		MGAA		Improvement (%)	
	$\overline{C_{max}}$	$\overline{Time(s)}$	$\overline{C_{max}}$	$\overline{Time(s)}$	$Imp1$	$Imp2$
Dataset 1	64.70	0.0053	63.80	0.0053	1.39	0.00
Dataset 2	125.30	0.0992	123.10	0.0446	1.76	55.04
Dataset 3	85.03	1.1959	82.23	0.5562	3.29	53.49
Dataset 4	124.51	0.7455	123.50	0.4211	0.81	43.51
Dataset 5	133.38	3.5829	130.70	1.1997	2.01	66.52
Dataset 6	136.35	4.8654	133.70	2.7778	1.94	42.91
Dataset 7	186.65	13.4329	182.95	8.4624	1.98	37.00

est in the context of limited computing resources in embedded systems.

5. Conclusions and perspectives

In this paper, we proposed adapted GAs for solving an important scheduling problem in CPU/FPGA architectures under heterogeneous communication delays' assumptions, with the aim of minimizing the makespan. We proposed two algorithms (GAA and MGAA) and we compared them to an existing reduced mathematical model. The results of MGAA demonstrated that this approach is better than GAA. In addition, it provides a significant enhancement compared to other existing exact methods such as the reduced mathematical model, in terms of the computation time required to solve a problem. Nevertheless, GAA provides good trade-off between computation time and makespan compared to the same exact mathematical model. To conclude, the proposed MGAA is a very promising approach that allows us to obtain optimal or near optimal solutions in a short running time. This approach can be easily and effectively adapted to other similar optimization problems to further maximize the computational performance. In particular, MGAA is suitable for scheduling problems dedicated to embedded heterogenous parallel architectures, which is still open and where the high performance in a real-time context with limited resources (computational units, power consumption, logic area, etc.) is required.

As a perspective, further comparisons between MGAA to other metaheuristic techniques seem to be interesting in order to find some improvement possibilities. Then, the hybridization of MGAA with other methods can be investigated.

References

[1] Maffioli, F. (1986). Randomized algorithms in combinatorial optimization: A survey.*Discrete Applied Mathematics*, 14(2), 157-170.

[2] Maringer, D. (2006). *Portfolio Management with Heuristic Optimization*. (1st ed.). Advances in Computational Management Science, New York: Secaucus, NJ.

[3] Kadri, A. A, Kacem, I. & Labadi, K. (2016). Branch-and-bound algorithm for solving the static rebalancing problem in bicycle sharing systems. *Computers and Industrial Engineering*, 95, 41-52.

[4] Kadri, A. A, Labadi, K. & Kacem, I. (2015). An integrated Petri net and GA-based approach for performance optimisation of bicycle sharing systems. *European Journal of Industrial Engineering*, 9(5), 638-663.

[5] Kacem, I. (2013). Genetic Algorithms for Solving Flexible Job Shop Scheduling Problems. In B. Jarboui, P. Siarry & J. Teghem (Eds.), *Metaheuristics for Production Scheduling* (pp. 19-44). Book Wiley-ISTE.

[6] Tanougast, C., & Killian, C. (2014). Optimization of a Reliable Network on Chip dedicated to partial reconfiguration. *2nd International Conference on Control, Decision and Information Technologies (CoDIT), IEEE Control System Society and IEEE Computer Society* , 778-782.

[7] Killian, C., Tanougast, C., Monteiro, F., & Dandache, A. (2014). Smart Reliable Network-on-Chip. *IEEE Transactions on Very Large Scale Integration Systems*, 22(2), 242-255.

[8] Tanougast, C., Berviller, Y., Brunet, P., Weber, S., & Rabah, H. (2003).Temporal partitioning methodology optimizing FPGA resources for dynamically reconfigurable embedded realtime system. *Microprocessors and Microsystems, Elsevier*, 27(3), 115-130.

[9] Kashan, A. H., & Karimi, B. (2009). A discrete particle swarm optimization algorithm for scheduling parallel machines. *Computers and Industrial Engineering*, 56(1), 216-223.

[10] Senthilkumar, P. & Narayanan, S. (2010). Literature Review of Single Machine Scheduling Problem with Uniform Parallel Machines. *Intelligent Information Management*, 2(8), 457-474.

[11] Zhou, H., Feng, Y. & Han, L. (2001). The hybrid heuristic genetic algorithm for job shop scheduling. *Computers and Industrial Engineering*, 40(3), 191-200.

[12] Kalczynski, P. J. & Kamburowski, J. (2005). A heuristic for minimizing the makespan in no-idle permutation flow shops. *Computers and Industrial Engineering*, 49(1), 146-154.

[13] Hamzadayi, A. & Yildiz, G. (2016). Event driven strategy based complete rescheduling approaches for dynamic m identical parallel machines scheduling problem with a common server. *Computers and Industrial Engineering*, 91, 66-84.

[14] Hojati, M. (2016). Minimizing maske-span in 2 stage disassembly flow shop scheduling problem. *Computers and Industrial Engineering*, 94, 1-5.

[15] Tari, F. G. & Haschemi, Z. (2016). A priority based genetic algorithm for nonlinear transportation costs problems. *Computers and Industrial Engineering*, 96, 86-95.

[16] Behnamian, J., Fatemi Ghomi, S.M.T., Jolai, F. & Amirtaheri, O. (2012). Minimizing makespan on a three-machine flowshop batch scheduling problem with transportation using genetic algorithm. *Applied soft computing*, 12(2), 768-777.

[17] Lee, W. C., Chuang, M. C. & Yeh, W. C. (2012). Uniform parallel-machine scheduling to minimize makespan with position-based learning curves. *Computers and Industrial Engineering*, 63(4), 813-818.

[18] Kim, B. K. & Kim, Y. D. (2011). Heuristic algorithms for assigning and scheduling flight missions in a military aviation unit. *Computers and Industrial Engineering*, 61(4), 1309-1317.

[19] Ait El Cadi, A., Souissi, O., Ben Atitallah, R., Belanger, N., & Artiba, A. (2015). Mathematical programming models for scheduling in a CPU/FPGA architecture with heterogeneous communication delays. *Journal of Intelligent Manufacturing (JIM)*. doi: 10.1007/s10845-015-1075-z.

[20] Souissi, O. (2014). *Path planning on a high performance heterogeneous CPU/FPGA architecture*, PhD in University of Valenciennes and Hainaut-Cambresis, France.

[21] Golub, M., & KASAPOVIC, S. (1999). Scheduling Multiprocessor Tasks with Genetic Algorithms. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 10(8), 825-837.

[22] Garey, M., & Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*, A series of books in the mathematical sciences. San Francisco, Calif.: W. H. Freeman and Co. pp. x+338. ISBN 0-7167-1045-5. MR 519066.

[23] Mitchell, M. (1996). *An Introduction to Genetic Algorithms*, Cambridge, MA: MIT Press.

[24] Kasahara, H., & Narita, S. (1984). Practical multiprocessing scheduling algorithms for efficient parallel processing. *IEEE Transactions on Computers*, 33(11), 1023-1029.

[25] Hou, E.S.H., Ansari, N., & Hong, R. (1994). A Genetic Algorithm for Multiprocessor Scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2), 113-120.

[26] Ahmad, I., & Dhodhi, M.K. (1996). Multiprocessor scheduling in a genetic paradigm. *Parallel Computing*, 22(3), 395-406.

[27] Mitchell, T. 1997. *Machine Learning*. McGraw-Hill Science/Engineering/Math.

[28] ILOG AMPL CPLEX System version 9.0 User's Guide, September 2003.
[29] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press.
[30] Melanie, M., (1996). *An Introduction to Genetic Algorithms*. Cambridge, MA: The MIT Press.
[31] Portmann, M.-C. (1996). *Genetic algorithm and scheduling: a state of the art and some propositions*. In Proceedings of the workshop on production planning and control, pages I-XIV, Mons, Belgium.
[32] Jin, F., Song, S., & Wu, C. (2009). A simulated annealing algorithm for single machine scheduling problems with family setups. *Computers & Operations Research*, 36(7), 2133-2138.
[33] Choobineh, F.F., Mohebbi, E. & Khoo, H., (2006). A multiobjective tabu search for a single-machine scheduling problem with sequencedependent setup times. *European Journal of Operational Research*, 175(1), 318-337.
[34] Chou, F.-D. (2009). An experienced learning genetic algorithm to solve the single machine total weighted tardiness scheduling problem. *Expert Systems with Applications*, 36(2), 3857-3865.
[35] Liouane, N., Yahia, H., & Borne, P. (2008). Multi-objective Scheduling onto Heterogeneous Processors System Using Ant System and Fuzzy Logic Controller. *Studies in Informatics and Control*, 17(1), 95-106.
[36] Anghinolfi, D., & Paolucci, M. (2009). A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 193(1), 73-85.
[37] Diana, S., Ganapathy, L., & Pundir, A.K. (2013). An Improved Genetic Algorithm for Resource Constrained Project Scheduling Problem. *International Journal of Computer Applications*, 78(9), 34-39.
[38] Arora, T. & Gigras, Y. (2013). A survey of comparison between various meta-heuristic techniques for path planning problem. *International Journal of Computer Engineering & Science*, 3(2), 62-66.
[39] Manikas, T. W. & Cain, J. T. (1996). Genetic algorithm vs. Simulated annealing: a comparison of apporaches for solving the circuit partitioning problem, *Computer science and engineering* , SMU digital repository, Technical report, 96-101.
[40] Sadegheih, A. (2006). Scheduling problem using genetic algorithm, simulated annealing and the effects of parameter values on GA performance. *Applied Mathematical Modelling*, 30(2), 147-154.
[41] Knuth, D. E. (1997). *The art of computer programming*. (3rd ed., Vol. 1). Boston: Addison-Wesley.

APPENDIX A: Ait El Cadi et al.'s reduced model [19]

$$\min C_{max} \tag{.1}$$

subject to:

$$\sum_{k\in\{l\in M/i\in F_l\}} x_{ik} = 1 \quad \forall i \in N \tag{.2}$$

$$s_i + \sum_{k=1}^{m} t_{ik}x_{ik} \leq C_{max} \quad \forall i \in N^+ \tag{.3}$$

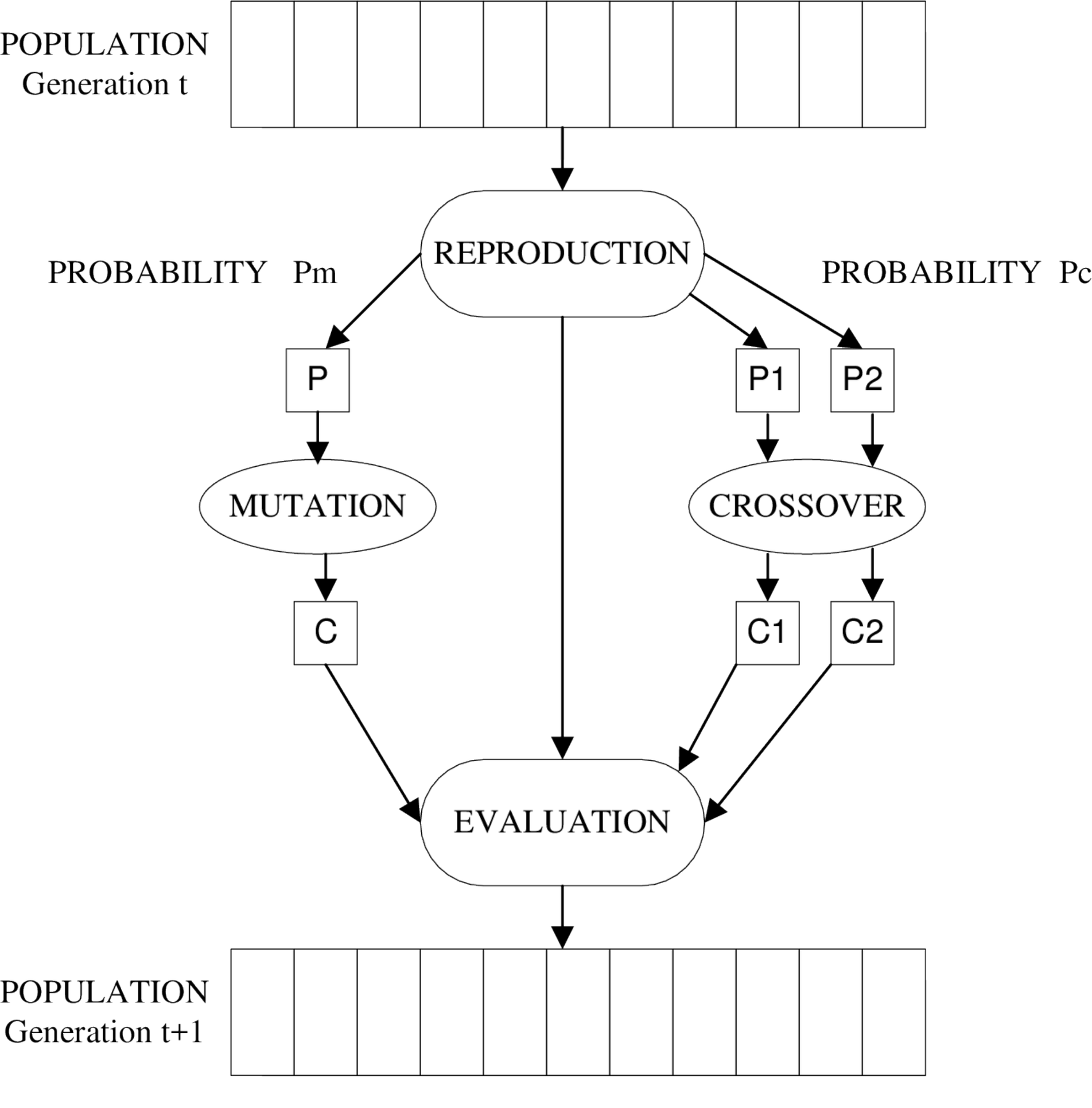
$$\begin{aligned} s_i + t_{ik}x_{ik} + c_{ik,jl}(x_{jl} + x_{ik} - 1) \leq s_j \quad \forall k,l \in M, \\ \forall j \in N\backslash F_l, \forall i \in Pred(j)\backslash F_k \end{aligned} \tag{.4}$$

$$\begin{aligned} s_i + t_{ik} - s_j \leq B(3 - x_{ik} - x_{jk} - \delta_{ij}) \quad \forall k \in M, \\ \forall i \in N\backslash F_k, \forall j \in N\backslash(P(i) \cup F_k) \end{aligned} \tag{.5}$$

$$\begin{aligned} s_j + t_{jk} - s_i \leq B(2 - x_{ik} - x_{jk} + \delta_{ij}) \quad \forall k \in M, \\ \forall i \in N\backslash F_k, \forall j \in N\backslash(P(i) \cup F_k) \end{aligned} \tag{.6}$$

$$x_{ik} \in \{0,1\}; s_i \in R^+ \quad \forall k \in M, \forall i \in N\backslash F_k \tag{.7}$$

Variable δ_{ij} could be seen as the decision that consists in performing task i before j ($\delta_{ij} = 1$) or not ($\delta_{ij} = 0$). For the assignment part, variable $x_{i,k}$ is binary and it indicates when $x_{i,k} = 1$ that task i is assigned to processor k. Otherwise, we have $x_{i,k} = 0$. The starting time of task i is defined by variable s_i for every i. It is worthy to note that the set *P(i)* of tasks j sharing the same path with i in G is computed as a pre-processing phase. It is based on the breadth-first search (BFS) method [19]. Moreover, to avoid increasing the number of binary variables, variable x_{ik} is not used for any i belonging to F_k (i.e., any task i that cannot be assigned to processing unit k) in all the constraints of this model.


POPULATION
Generation t
REPRODUCTION
PROBABILITY Pm
PROBABILITY Pc
P
P1
P2
MUTATION
CROSSOVER
C
C1
C2
EVALUATION
POPULATION
Generation t+1