


AI-Based Methods to Resolve Real-Time Scheduling for Embedded Systems: A Review

Fateh Boutekkouk, ReLaCS2 Laboratory, University of Oum el Bouaghi, Algeria*

 <https://orcid.org/0000-0003-0398-4597>

ABSTRACT

Artificial intelligence is becoming more attractive to resolve nontrivial problems including the well-known real-time scheduling (RTS) problem for embedded systems (ES). The latter is considered as a hard multi-objective optimization problem because it must optimize at the same time three key conflictual objectives that are tasks deadlines guarantee, energy consumption reduction, and reliability enhancement. In this paper, the authors firstly present the necessary background to understand the problematic of RTS in the context of ES. Then they present enriched taxonomies for real-time, energy, and fault-tolerance-aware scheduling algorithms for ES. After that, they survey the most pertinent existing works of literature targeting the application of AI methods to resolve the RTS problem for ES, notably constraint programming, game theory, machine learning, fuzzy logic, artificial immune systems, cellular automata, evolutionary algorithms, multi-agent systems, and swarm intelligence. They end this survey with a discussion putting the light on the main challenges and the future directions.

KEYWORDS

Artificial Intelligence, Embedded Systems, Energy Consumption, Real-Time Scheduling, Reliability

1. INTRODUCTION

Embedded systems (ES) have penetrated our life to a point where we cannot ensure our daily business, assignments and chores in their absence. These systems have undergone a dramatic increase in functionality and omnipresence in such a way no one can negate their remarkable influence on our behaviors, habits and even our convictions.

Whatever their architectures type (i.e. centralized Vs. distributed), the used technologies (i.e. wired, wireless, optical) and the application fields (i.e. automotive, avionics, space, robotics, health care, military, entertainment and so on), ES have some common decisive requirements among others the real time constraints, the reduced energy consumption and the reliability assurance.

First, ES are qualified as Real Time (RT) systems. A RT system is any system where its correction depends not only on the results of computations, but also on the time instants at which these results become available. In other term, a real-time system is responsible for delivering logically correct computations within the predefined deadlines.

DOI: 10.4018/IJCINI.290308

*Corresponding Author

This article published as an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

Depending on the severity of the timing constraint (i.e. deadline), RT systems can be hard (i.e. critical), soft, firm or any combination of them. RT systems typically incorporate a RTOS (Real Time Operating System) kernel. The latter is responsible of many vital activities for ES such as tasks scheduling, Input/output management, memory management, security and so on. Since ES are often battery-dependent, their design have to minimize their power dissipation and energy consumption accordingly. Energy-aware design methodologies of ES are becoming popular in the ES terminology. The other important characteristic is the reliability.

In its large sense, reliability means the capacity of the system to continue its functioning even in the absence of faults. Reliability includes many aspects or attributes such as availability, safety (i.e. functional correction), data integrity, etc. Each attribute has a set of means to realize it. For example, availability can be achieved through faults tolerance. The latter has many mechanisms among them the spatial redundancy (i.e. hardware redundancy) and the temporal redundancy (i.e. the re-execution of the same task or code). Faults tolerant or reliability-aware methodologies are also being an inherent part of the ES jargon.

Contrary to traditional ES, nowadays, ES are becoming more complex, more open and networked and integrate intelligent parts that can function in hostile, dynamic environments (probably with uncertain or partial knowledge) autonomously, simulating a bit of some human intellectual activities such as reasoning, learning, memorization, perception, decision-making, self-adaptation, and self-optimization. On the other hand, the exponential progress in the hardware technology conducting to the appearance of multi-core and parallel computing on chip, very high performance processing and reconfigurable hardware render embedding AI in ES possible. Of course, this coupling between AI and ES is not trivial at all, since the two fields have different philosophies. While AI deals with more complex cognitive, theoretically with unlimited resources tasks, ES are by nature reactive and have limited resources. The integration of AI into ES leads to the emergence of what we call 'intelligent embedded systems' (IES).

IES design is a hot research topic investigating the application of the most famous AI models and methods as Artificial Neural Networks (ANN), Reinforcement learning, multi agent systems (MAS), swarm intelligence, Genetic Algorithms, fuzzy logic, constraint programming, game theory, cellular automata, and Artificial Immune Systems (AIS) to ES design while meeting the temporal, the energetic and the reliability requirements in addition to the cost constraint and end-users goals satisfaction.

This paper is interested in the application of AI to resolve the well-known problem of RT scheduling for embedded systems. The possibility of coupling AI with RT systems was discussed earlier (Musliner et al., 1994) and a few existing works had yet been interested in the application of expert systems to resolve the traditional jobs scheduling problem (MacCarthy & Jou, 1995). In turn, (Laalaoui & Bouguila, 2014) presented a non-exhaustive survey on the application of some AI methods to the static RT scheduling.

With regard to the application of AI methods to resolve the RT scheduling problem for ES, it is stated that existing surveys on this topic are not exhaustive enough, hence the need to produce a new exhaustive survey with additive knowledge and some novel insights on the use of AI methods to resolve and optimize RT scheduling for embedded systems taking into account the energy consumption and the reliability. The survey paper is organized as follows: first, ES are introduced while showing their features and classification, the definition of intelligent embedded systems and the RT scheduling problem rationalization and possible taxonomies. A more exhaustive taxonomy of RT scheduling algorithms for ES is also presented enriching previous taxonomies by adding some important criteria. After that, the application of AI methods to resolve the RT scheduling problem for ES is studied based on previous research articles, conferences papers and surveys. Last sections are devoted to a discussion putting the light on some main challenges and future directions.

2. EMBEDDED SYSTEMS

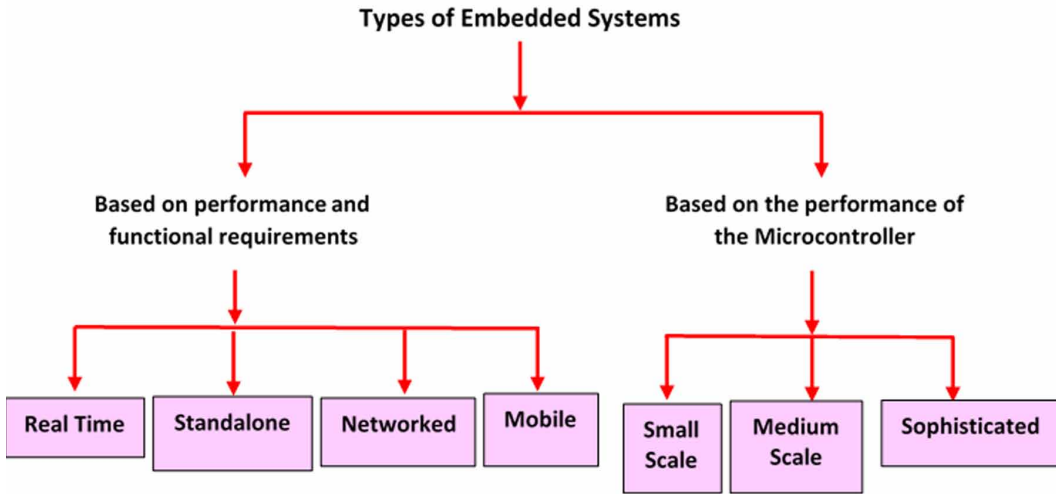
Traditionally defined, an embedded system (ES) is merely a limited-resources computing system, which is integrated into a larger system as a software part, a hardware part or a conjoint software/hardware part and which interacts with the physical world or the external environment continually via sensors/actuators to accomplish a certain task. Sometimes we refer to ESs as a special case of cyber physical systems (CPS). ES should meet a set of conflictual objectives notably timing deadlines, reduced energy consumption, small memory footprint, small weight, low cost, and reliability.

Conventional ES are centralized, simple and closed systems that interact with a fully specified environment via sensors/actuators and assumed to achieve some well-defined tasks under the supervision of a micro-controller functioning as a simple feedback loop. With the ever increasing in Integrated Circuit density integration and ICT (Information and Communication technologies), ES are being evolved in order to respond to the new increasing requirements in terms of multi-applications execution support, high performance computing, large scaling, scalability, autonomy, self-adaptation, security and at the same time to support the next-future technologies as wireless/optical communication, sustainable energy, IoT (internet of things), big data, and cloud computing. Hence, the necessity to develop new design methodologies or to tune and boost conventional methodologies in order to cope with these emerging issues.

Typically, an ES includes a RTOS kernel. The latter plays the role of an intermediate between the hardware layer and the application layer. The RTOS assures the most vital activities of an ES especially the RT scheduling, inputs/outputs and memory management and protection.

Figure 1 shows a possible classification of embedded systems (Classification-of-embedded-systems, n.d). In this classification, two main criteria are used: the system performance and functional requirements, and the performance of the microcontroller. Based on the first criterion, ES are classified into four categories that are Standalone ES, Real time ES, Networked ES, and Mobile ES. Based on the second criterion, ES can be classified into three main categories: Small scale ES, Medium scale ES, and Sophisticated ES. Standalone ES refer to ES working by themselves (i.e. do not require a host system). Examples for the standalone-embedded systems are digital cameras and video game consoles. Networked ES are the distributed version of the traditional centralized ES. The connected network can be LAN, WAN or the IoT (Internet of Things). The connection can be wired or wireless. Mobile ES are ES, which are used in portable embedded devices like cell phones. Small Scale ES are ES with a microcontroller of 8 or 16 bits. Medium Scale ES are ES with a microcontroller of 16 or 32 bits, RISCs or DSPs. Sophisticated ES are ES with more sophisticated hardware/software components as ASIPs, IPs, or configurable processors. They are used to implement complex applications following a Hw/Sw Codesign approach. Sophisticated ES are implemented as complex SOC (System On Chip), NOC (Network on Chip) or WiNOC (wireless NOC). The latter paradigm is a promising solution to mitigate the large delay and high power dissipation issues offered by the conventional NOC. Of course, there exist other classifications of ES considering other criteria as the application domain, the target SOC and NOC architectures. In principle, most ES are RT and require an RT scheduler to assure that all or a certain percentage of system tasks meet their deadlines while minimizing the energy consumption and maximizing the system reliability. The classification in figure 1 can in its turn be refined. For example, RT ES can be further subdivided into three main sub-classes: Hard (critical), Soft, and firm. In hard RT systems, the consequences of missing a deadline can be catastrophic. In soft and firm RT systems, the results are relatively tolerable but for firm systems a degradation in quality of services is very potential. Networked ES can be also divided into two sub-classes: wired ES and wireless ES. Each one can be further partitioned into many categories with regard to the scope of the network. For more details, one can refer to (Boutekkouk, 2019a).

Figure 1. Classification of ES



3. INTELLIGENT EMBEDDED SYSTEMS

(Elmenreich, 2003) identified some potential reasons for using an intelligent solution for ES among them dependability, efficiency, autonomy, easy modelling, maintenance costs and insufficient alternatives. Intelligent embedded systems (IES) are ESs having the capacity of reasoning about their external environments even in the presence of uncertainty and adapt their behavior accordingly. IES have some main characteristics such as self-learning, self-optimizing and self-repairing. We can say that IES are the fruit of coupling between embedded computing and Artificial Intelligence (AI). Therefore, IES can include knowledge-based technology. Recently, the use of AI techniques in ES has proliferated. In terms of ES, this gives rise to the possibility of developing systems that can learn from their environment and that can change their own control programs to adapt to new situations. Intelligent WSN (wireless sensors networks), intelligent vehicles, and robots are becoming very popular. IES applications are growing more and more covering a large spectrum of domains such as farming, smart buildings, education and transport.

4. REAL TIME SCHEDULING

Real time Scheduling (RTS) is a decisive activity in ES design. It can be defined as the process of assigning dates of execution start to system computational and communicational tasks such that deadlines are met totally or partially. This definition is sound when scheduling is performed on a system with only one computational (i.e. one processor) or communicational resource (i.e. one bus). For a system with multiple resources, the RTS has to take into consideration the allocation or mapping of tasks to system resources (spatial mapping) in addition to scheduling (temporal mapping). The RT scheduler is generally based on the so called the canonical model of real time tasks (Lee et al., 2007). As it is shown in figure 2, each task in this model is defined by a set of primary timing parameters. These parameters include:

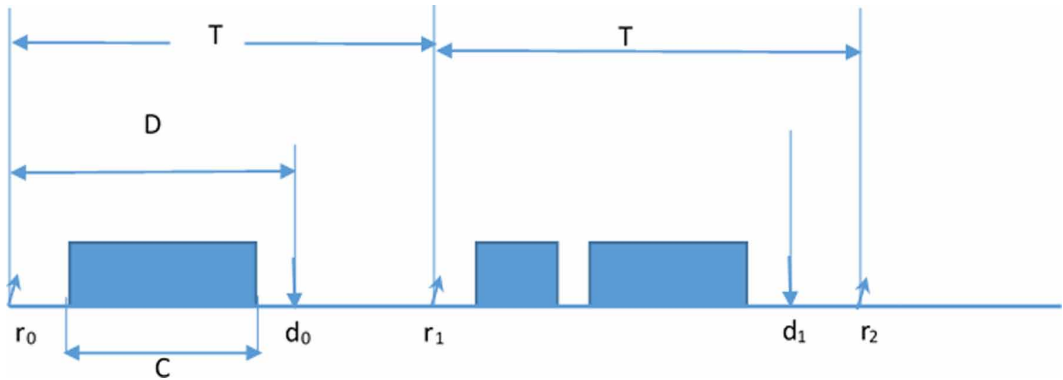
- r, task release time referring to the triggering time of the task execution request.
- C, task worst-case computation time, when the processor is fully allocated to it.
- D, task relative deadline, referring to the maximum acceptable delay for its processing.

- T , task period (valid only for periodic tasks).

When the task has hard real-time constraints, the relative deadline allows computation of the absolute deadline $d = r + D$.

The successive release times are request release times at $r_k = r_0 + kT$, where r_0 is the first release and r_k the $k + 1^{\text{th}}$ release; the successive absolute deadlines are $d_k = r_k + D$

Figure 2. Canonical model for real time tasks



Periodic tasks request times are known a priori and repeated at regular time intervals (i.e. periods). In order to simplify schedulability analysis, the LCM (Least common multiple) of all tasks periods is often considered. Sporadic task request times are not known a priori, but it is assumed that a minimum interval exists between two successive requests. Aperiodic tasks have no such constraint on their request times. Usually, the request times follow some probabilistic laws. Tasks can be independent or have precedence, synchronization, and mutual exclusion constraints between them. Tasks can be mapped to processors in a preemptive or non-preemptive manner. Preemptive means that the running task can be interrupted at any time to assign the processor to another ready task, whereas non-preemptive means that, a task once started executes to completion before relinquishing the processor.

The problem of scheduling tasks with precedence and synchronization constraints on a set of processors is NP-complete and heuristics are typically used to obtain a feasible schedule.

A dynamic or on-line scheduler makes its scheduling decisions at run time whereas, a static or off-line scheduler generate a feasible schedule that is guaranteed to meet the timing constraints of all tasks at design time. Static scheduling is more suited to critical systems with periodic tasks.

RT scheduler can execute on only one processor or over a set of homogeneous or heterogeneous processors. Multiprocessor scheduling has to solve the allocation problem that consists in deciding on which processor a task should execute, and the priority problem, that is, when a task should execute. In global scheduling, there is only one tasks queue and tasks migration is allowed, whereas, in partitioned scheduling, each processor has its own tasks queue and no migration is allowed. By separated RT scheduling, we mean multiprocessor RT scheduling that treats the allocation and scheduling sub-problems separately (i.e. sequential manner). Simultaneous RT scheduling treats the two sub-problems concurrently.

Separating the two sub-problems render the problem more simpler but at the prize of sub-optimality. The simultaneous treatment makes the problem more complex but leads to optimal results. With the appearance of networked ES, RT scheduling becomes distributed. Consequently, each node of the distributed system has its own local scheduler.

A significant number of RT scheduling algorithms have been developed over last decades and classified following a set of criteria. The latter include mainly: tasks periodicity (periodic vs aperiodic), dependency (dependent vs. independent), the number of processors (mono vs. multiprocessor), tasks priority calculation (fixed vs dynamic) and online vs. offline scheduling.

RT scheduling is probabilistic, if it applies some laws of probability to select tasks or to compute some timing parameters (Tidwell, 2011). Some probabilistic RT scheduling algorithms model the problem as queues theory. RT scheduling is imprecise, if its decisions regarding tasks selection or priorities calculation is based on a partial, vague or imprecise data. In some cases, intermediate or partial results from task computations can be used instead of more precise results when a real-time system suffers failures or transient overloads.

RT scheduling is adaptive if it can adjust its parameters dynamically to respond to some events (i.e. faults) or to meet deadlines. For instance, in some situations, the RT scheduler has to adjust the period of a task (Zhou et al., 2017).

Traditional RT scheduling is concerned merely with the timing issue; however, with the emerging of battery-dependent systems, the RT scheduling should minimize the energy consumption too. Moreover, the RT scheduling has to enhance the reliability of the system especially for critical systems.

Energy-aware RT scheduling algorithms tend to minimize the energy consumption in embedded systems (processors, buses, memories) while meeting all tasks deadlines or a subset of the set of deadlines (Niu & Quan, 2006).

The problem of reducing energy consumption while meeting tasks deadlines is not trivial since time and energy decreasing are conflictual objectives. For this reason and in order to find a good tradeoff, energy-aware algorithms have been developed. Figure 4 shows a possible classification of energy-aware RT scheduling algorithms. Previous taxonomies are enriched by adding the class of algorithms applying the technology of energy harvesting.

As shown in figure 4, energy-Aware RT Scheduling algorithms are traditionally classified regarding whether they minimize the dynamic power, the static power or both two, whether they use nonrenewable or renewable energy technology and whether they run a single core or multi-cores CPUs (Bambagini et al., 2016).

Most energy-aware algorithms minimize dynamic power that is related to the amount of the switching activity in the hardware resource (i.e. the processor), its supply voltage and clock frequency. With the continuing reduction of the transistor dimension, static power becomes more significant and cannot be neglected anymore. Static power is mainly related to the current leakage. Recent algorithms take into account both power types. Nevertheless, for static power minimization, the algorithms resort to low levels energy models and simulators. Other algorithms make some abstractions in order to simplify the power estimation but at the prize of estimation impreciseness. Energy-aware algorithms can minimize temperature. In order to do so, they generally resort to some thermal models to estimate the generated temperature at hardware components. In the case of multicores processor architectures, the objective of temperature-aware algorithms is rather to balance the distribution of temperature between the different cores so the hotspot of the processor is avoided (Ahmeda et al., 2011).

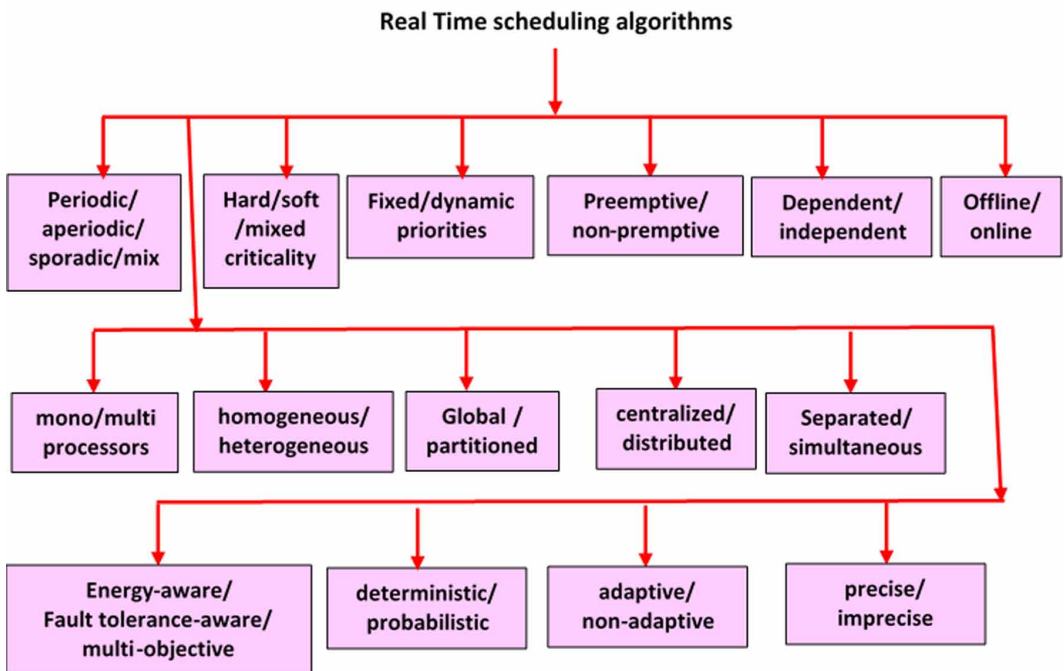
Single core algorithms are first classified along the DVFS (Dynamic Voltage Frequency Scaling) and DPM (Dynamic Power Management) dimensions. DVFS-based algorithms are based on the principle of scaling the supply voltage and the clock frequency of the processor dynamically to minimize the processor energy. Each processor has different speed (voltage) levels at which it can execute (interval of discrete or continuous values).

DPM-based energy management techniques selectively place system components into low-power states when they are idle at runtime. A power managed system can be modeled as a power state machine, where each state is characterized by the power consumption and the performance. In addition, state transitions have power and delay cost. DVFS algorithms are classified according to the type of slack (the unused CPU time) that they reclaim for scaling speed to save energy. Specifically, the algorithms that exploit only the static slack consider the residual processor utilization in the worst-

case execution, whereas those that reclaim the dynamic slack take advantage of the difference between the worst-case and the actual execution time of the jobs. Various real-time DVFS techniques have been studied, among them the presented static, cycle-conserving (CC), Look Ahead (LA), dynamic reclaiming algorithm (DRA), dynamic reclaiming-one task extension (DR-OTE), and aggressive speed adjustment (AGR) algorithms are the most important.

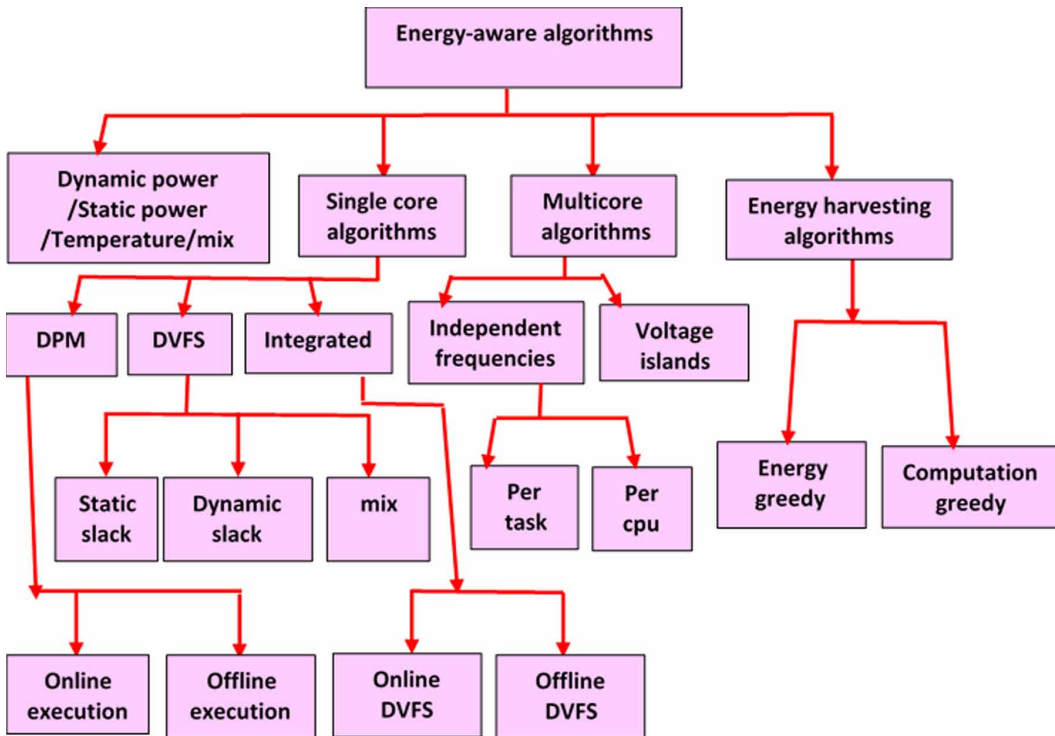
DPM algorithms are classified as offline and online approaches. The algorithms that use both DVFS and DPM techniques are designated as integrated algorithms. These algorithms are further divided according to when the task speed assignment decisions are made, that is, either offline or online. Multicore algorithms are classified according to the flexibility in the DVFS support provided by the platform. If the hardware allows setting a different frequency for each core, the DVFS algorithms are classified as Independent Frequencies, whereas if a single frequency is shared among a subset of cores, the algorithms are classified as Voltage Islands

Figure 3. Classification of RT scheduling algorithms for ES



The DVFS multiprocessor algorithms for independent frequencies can be further distinguished between approaches that assign frequencies to cores independently of the running tasks (Per-CPU algorithms) and those that compute a frequency for each task and use it for the core executing that task (Per-Task algorithms). ES which are powered by a renewable energy source are qualified as energy harvesting systems. Uncertainty of energy availability in energy harvesting systems makes the problem of task scheduling more challenging. In general, energy-harvesting algorithms can be broadly divided into energy-greedy and computation-greedy classes depending on the actions they take when the energy level is low. Under that condition, energy-greedy algorithms exploit the available slack in the system by procrastinating tasks and charging the battery as much as possible. In contrast, the computation-greedy algorithms give priority to execute the pending workload, and charge the battery

Figure 4. Classification of energy-aware algorithms



only for the shortest time, which guarantees the execution of the next computational unit (Moser et al., 2006; Bambagini, 2014; Chandarli, 2014; Housseyni et al., 2016).

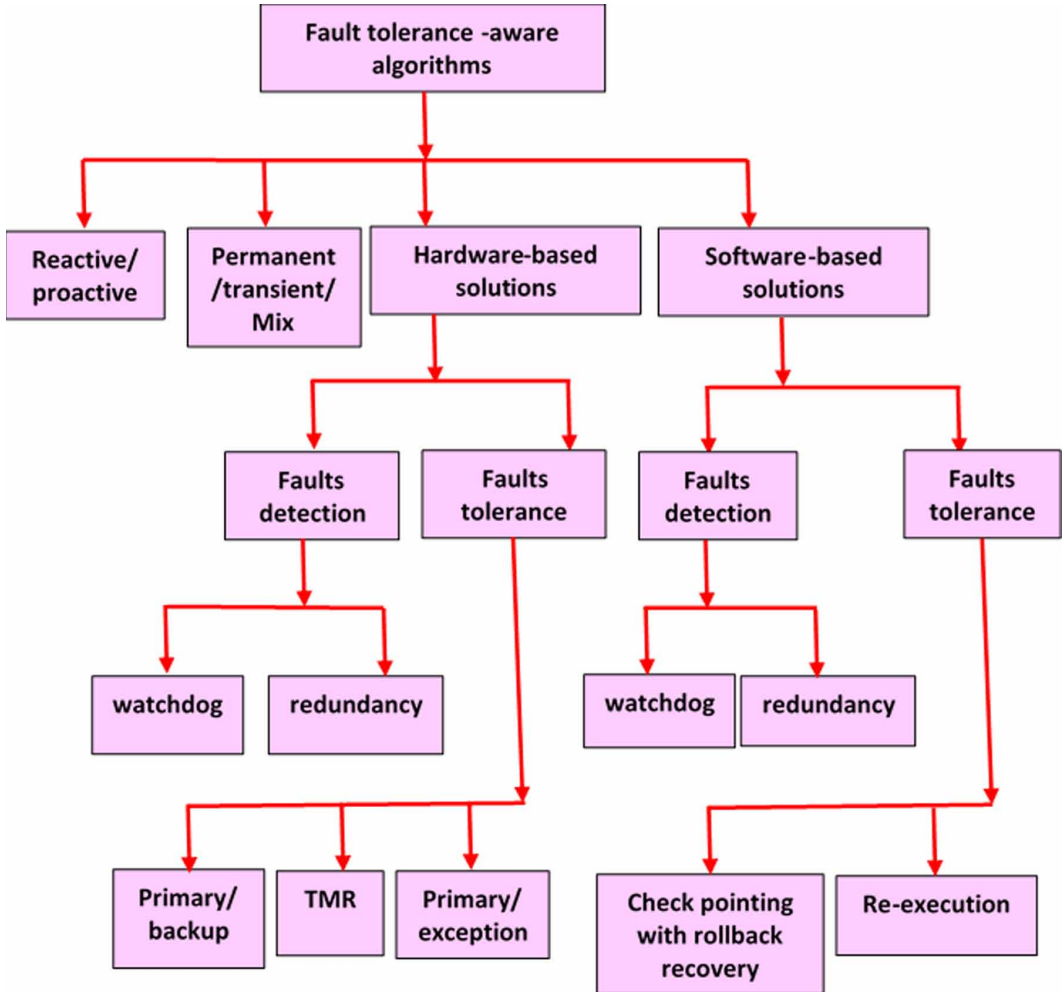
Faults tolerant-aware RT scheduling has to guarantee the functional and timing correctness even in the presence of hardware and software faults. Faults are generally classified into three main categories: permanent, transient, and intermittent. Permanent faults do not die away with time. They are caused by total failure of the computational unit and remain until they are repaired as the affected unit is replaced (hardware redundancy). Transient faults are temporary malfunctioning of the computational unit. They die away after some time. Intermittent faults are repeated occurrences of transient faults. Many Faults tolerant-aware algorithms have been proposed to deal with the different kinds of faults. Most of these algorithms deal with transient faults that occur at the processor and bus levels.

In order to achieve fault tolerance, the first requirement is that faults have to be detected. In general, there exists two main techniques for faults detection: watchdog and redundancy.

Watchdog timer monitors periodically the execution time of programs or transmitted data, whether it exceeds a certain limit. Redundancies can be classified into two categories hardware-based redundancy and time-based redundancy. Hardware-based redundancy methods attempt to tolerate transient faults by copy-executions of each original task on another separated hardware. These methods can be classified into three main categories: TMR (Triple Modular Redundancy), PB (Primary/Backup), and PE (Primary/Exception). In TMR, the critical components are replicated three times and error checking is achieved by comparing results after completion. In this scheme, the overhead is always on the order of the number of copies running simultaneously. In PB, the tasks are assumed to be periodic and two instances of each task (a primary and a backup) are scheduled on a uni-processor system. The main idea behind this technique is that the backup of a task need not execute if its primary executes successfully and that no resource conflicts occur between the two versions of any task. One of the restrictions of this approach is that the period of any task should be

a multiple of the period of its preceding tasks. It also assumes that the execution time of the backup is shorter than that of the primary. PE is the same as PB method except that exception handlers are executed instead of backup programs. In general, hardware redundancy is avoided as far as possible, due to limited resources. Software (or Temporal) redundancy is more cost-efficient to handle transient faults. One possible approach is to schedule critical tasks multiple times and perform voting of the results (re-execution).

Figure 5. Classification of faults tolerant-aware algorithms



Another common technique is to insert checkpoints into the software and rollback the execution from a safe state in case faults are detected. For real-time applications, temporal redundancy must be used with utmost care, since the overhead in time may lead to deadline violations. Faults-tolerant algorithms can be either reactive in the sense they handle the faults only when they occur. Whereas, cognitive algorithms can predict and plan in priori the handling of faults (Kandasamy et al., 2000; Mottaghia & Zarandi, 2014; Han, 2015; Fan & al., 2017; Barkahoum & Hamoudi, 2019).

Some typical examples of intelligent RT embedded systems scheduling applications include outpatient clinics (OPCs), smart transport and spacecraft.

In the context of OPCs, intelligent RT scheduler that schedules patients and resources based on the actual status of departments is crucial particularly when the patient demand is high and patient arrivals are random. Generally, OPCs systems are push systems where scheduling is based on average demand prediction and is considered for long term (monthly or bimonthly). Often, planning and actual scenario vary due to uncertainty and variability in demand and this mismatch results in prolonged waiting times and under-utilization of resources (Munavalli et al., 2020).

In the field of smart transport, the focus of the smart transportation industry has been shifting towards the research and development of smart cars with autonomous control while promoting safe driving which is one of the crucial concerns in autonomous smart cars. The major issue for the better provision of safe driving is real time tasks scheduling and an efficient inference system for autonomous control. In such systems, an optimal control system consists of an intelligent part which can be implemented as ANN or an expert system with a knowledge base and a control unit; where the knowledge base contains the data and thresholds for rules and the control unit contains the functionality for smart vehicle autonomous control. The intelligent RT scheduler provides an efficient way of controlling smart cars in different scenarios such as heavy rainfall, obstacle detection, driver's focus diversion etc., while ensuring the practices of safe driving, timing constraints respect and energy consumption (Sehrish et al., 2019).

Spacecraft operations have been a major area of application for intelligent RT scheduling.

the use of an automated intelligent scheduler will assist to create observations of both targeted geographical regions of interest and general mapping observations while respecting spacecraft constraints such as data volume, observation timing, visibility, lighting, season, and science priorities. Numerous space missions have used automated planning and scheduling on the ground to enable significant operational efficiencies. For instance, Europa Clipper has been a mission concept under study by NASA for a spacecraft to fly to the Jovian planet-moon system in order to study the icy moon Europa. The Europa Clipper concept considers a number of possible science instruments, including a radar to study the ice shell and subsurface properties, and infrared instrument to study surface composition, a topographic imager to gather high resolution images of surface features, and an ion and neutral mass spectrometer to investigate Europa's trace atmosphere during flybys. In order to achieve these goals, an automated intelligent RT scheduler is implemented. The intelligent scheduler functions on three steps namely instrument definition, campaign generation, and target selection. In the first step, the spacecraft and instruments must be defined along with the constraints that may impact how and when data can be collected. To generate valid schedules, the interactions between the instruments and the spacecraft should be modeled as well as how the instruments interact with each other. In the second step, campaigns are generated to represent the constrained and prioritized requests of the scientists. In order to collect relevant data for a particular scientific campaign, constraints are made on both internal and external conditions. Then, a priority is assigned to each campaign to enable the scheduler to make the best choice when different observation types are feasible. Finally, the last step is to select the best tradeoff between feasible observations (Rabideau & al., 2015).

4. RESOLVING THE RT SCHEDULING PROBLEM FOR EMBEDDED SYSTEMS USING AI.

The RT scheduling problem for ES can be defined as a Multi-Objective Optimization (MOO) problem under constraints. It is qualified to be NP-hard, makes it difficult to devise conventional approaches. As an alternative, AI-based approaches seem to be promising solutions. RT scheduling optimization for ES has to take into account three conflictual objectives that are deadlines respect, energy consumption minimization and reliability.

This poses a big challenge because lowering the voltage to reduce energy consumption reduces the processor clock frequency that means increasing task execution time, which can lead to no guarantee of task deadlines. Furthermore, lowering the voltage to reduce energy consumption has been shown to increase the number of transient faults. Even the redundancy mechanism (i.e. hardware redundancy) to tolerate faults will increase the energy consumption although it decreases the overall execution time. For a nontrivial MOO problem, no single solution exists, that simultaneously optimizes each objective and there exists a (possibly infinite) number of Pareto optimal solutions. A solution is called no dominated or Pareto optimal if none of the objective functions can be improved in value without degrading some of the other objective values. In the literature, there are several theoretical options to solve a MOO problem. Many methods convert the original problem with multiple objectives into a single-objective. This is called a scalarized problem. MOO methods can be generally partitioned into four classes (Multi-Objective Optimization, n.d):

- The “no preference” methods in which decision maker (DM) is expected to be available, but an unbiased tradeoff solution is identified without preference information.
- The “a priori” methods in which preference information is first asked from the DM and then a solution best satisfying these preferences is found.
- The “a posteriori” methods in which a representative set of Pareto optimal solutions is first found and then the DM must choose one of them.
- The interactive methods in which the DM is allowed to iteratively search for the most preferred solution. In each iteration, the DM is shown Pareto optimal solution(s) and describes how the solution(s) could be improved. The information given by the DM is then taken into account while generating new Pareto optimal solution(s) to study in the next iteration.

AI includes a set of methods and models to address complex problems of the real world application including the RT scheduling problem.

By intelligent RT scheduling, it is meant any conventional RT scheduling which incorporates an AI method or model. Figure 6 shows a classification of AI methods which have largely been used to resolve RT scheduling. These models comprise mainly: Constraint Programming (CP), Game Theory (GT), Artificial Immune Systems (AIS), Cellular Automata (CA), Machine learning including Artificial Neural Networks (ANN) and Reinforcement learning, Fuzzy logic, Multi-Agent Systems (MAS), Evolutionary Algorithms (EA), and Swarm intelligence including ants colony, bees colony and PSO.

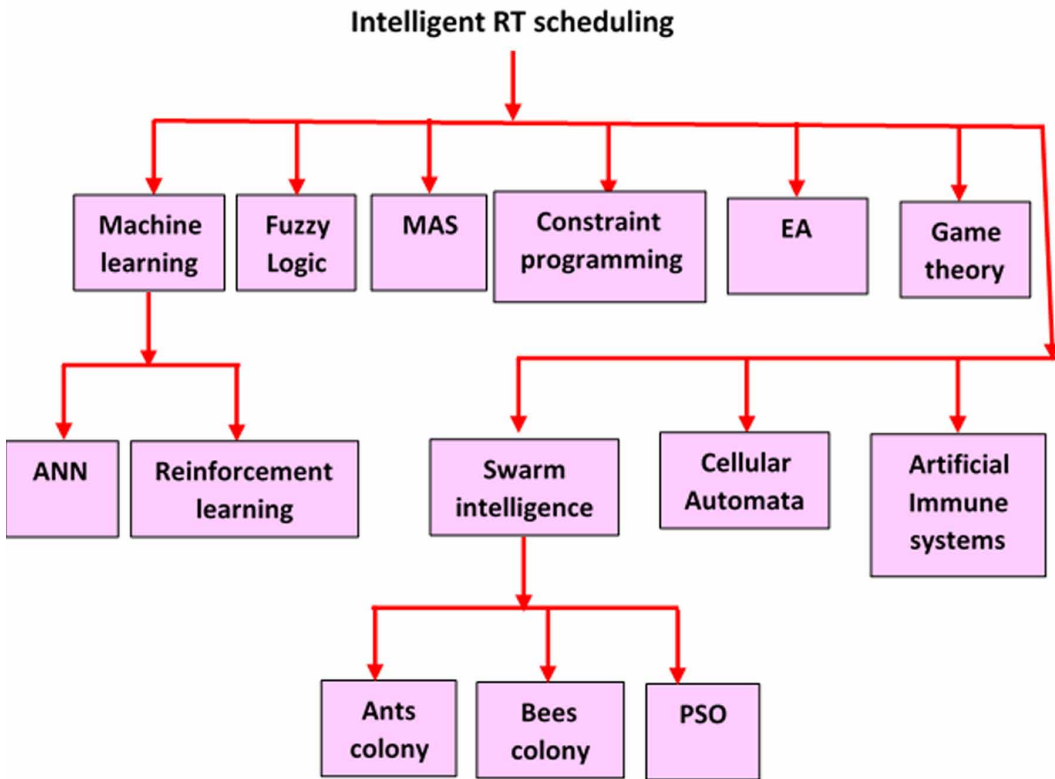
4.1. Constraint Programming

Due to the large variety of constraints that must be fulfilled by a RT scheduler, the constraint programming (CP) paradigm seems to be justifiable. In this context, the RT scheduling problem can be formulated as a constrained satisfaction problem (CSP).

CP is so popular, because it helps to specify in a declarative and generic fashion relevant constraints. Furthermore, CP offers a powerful search space reduction using constraint propagation that can detect infeasible branches in the search tree earlier and this is triggered every time a new constraint is added to the model. One big advantage of CP is the possibility to create and integrate new heuristics using the available meta-heuristics (Szymanek et al., 2000). CP can be applied also in solving combinatorial optimization problems by expressing the objective function as a constraint variable and then iteratively, solve the same problem with an increasing tighter bound on this variable. Basically, CP encompasses two steps that are:

1. Formulate the problem in terms of variables and constraints.
2. Find a feasible or an optimal assignment of the variables such that the constraints are satisfied.

Figure 6. Classification of AI methods used to resolve RT scheduling



In CP, it is well known that the time to seek for feasible solutions represents the major performance bottleneck. The search must be guided by a search strategy that aims at quickly directing the search towards good solutions without losing the completeness of CP. A search strategy is related to the order of choice of variables with their values.

Standard CP optimization method is based on branch and bound (B&B) algorithm. The latter can be successfully applied to small and middle size problems, but for large and complex problems with heterogeneous constraints, more sophisticated optimization methods are required. Dynamic Constraint Programming (DCP) is a version of CP where dynamic (i.e. at execution time) addition/retraction of constraints are possible. Two main classes of methods can be distinguished in DCP: proactive and reactive methods. Proactive methods propose to build robust solutions that remain solutions even if changes occur. On the other hand, reactive methods try to reuse as much as possible previous reasoning and solutions found in the past. They avoid restarting from scratch and can be seen as a form of learning. One of the main methods currently used to perform such learning is a justification technique that keeps trace of inferences made by the solver during the search. Such an extension of constraint programming is called explanation-based constraint programming.

In her PhD thesis, (Eklin, 2004) coped with periodic and dependent tasks RT scheduling problem taking into account the energy consumption minimization for ES on heterogeneous multiprocessor architecture with a shared bus using CP.

Firstly, she proposed a taxonomy of typical constraints that appear in RT embedded systems design. A set of heuristics to reduce the search time of the optimization algorithm had also proposed. These heuristics use some design space reduction techniques such as symmetries exclusion. In addition,

the author integrated a power model in order to minimize the energy consumption while meeting the deadlines. She applied the DVFS technique.

The constraint model uses many variables:

For a task: The start time, the execution time, the blocking time, the execution node (processor), the speed level and the consumed energy. For a message: the start time, the transmission delay.

Some additional assumptions are made:

- A task invocation is not allowed to migrate to another node during preemption,
- A task invocation will run at a single speed level during its entire execution.

For a schedule of length lcp , each task invocation is treated as a separate task.

A possible CP-based formulation of the RT scheduling problem is illustrated in figure 7.

Where:

$start(\tau \frac{k}{i})$ is the start time of the task

$node(\tau \frac{k}{i})$ is the execution node

$level(\tau \frac{k}{i})$ is the speed level

$start_message(\tau \frac{k}{i}, \tau \frac{l}{j})$ is the start time of the message transmission

The work of (Hladik et al., 2005) applied dynamic constraint programming (DCP) to solve the problem of distributed preemptive RT scheduling taking into account fault-tolerance including periodic tasks with fixed priorities and hard constraints. The processors are considered homogeneous. (i.e. they have the same speed) and are fully connected to a network using a token ring protocol. The presented method follows logic Benders-based decomposition. The latter can be seen as a form of learning from mistakes. It is a solving strategy that uses a partition of the problem among its variables: x, y . The strategy can be applied to a problem of this general form:

$$P : M \inf(x) + cy$$

$$s.t : g(x) + Ay \geq a \quad \text{with : } x \in D, y \geq 0$$

The master problem considers only a subset of variables x and the sub-problem (SP) tries to complete the assignment on y and produces a Benders cut added to the master problem.

The author separated the allocation problem (master problem) from the schedulability (sub-problem) one. The allocation and resource constraints is solved by means of DCP tools, whereas schedulability and timing constraints is checked with specific real-time scheduling analyses. In other term, the master problem is solved with CP yields a valid allocation and the subproblem checks the schedulability of this allocation finding out why it is unschedulable and designing a set of constraints, named nogoods which rules out all the assignments which are unschedulable for the same reason. The main idea is to learn from the schedulability analysis how to re-model the allocation problem and reduce the search space.

Two classes of constraints are defined:

- Resources constraints including memory capacity, processor utilization factor and network use.
- Allocation constraints that are imposed by the system architecture. These constraints include residence (a task needs some specific resource which is only available on specific processors),

Figure 7. RT scheduling problem formulation using CP

$$\begin{aligned}
 & \text{minimize } f \\
 & \text{subject to} \\
 & \text{speed} \\
 & \forall \tau_i^k : \text{execution_time}(\tau_i^k) = \left\lceil \frac{c(i, \text{node}(\tau_i^k), 1) \cdot f(\text{node}(\tau_i^k), 1)}{f(\text{node}(\tau_i^k), \text{level}(\tau_i^k))} \right\rceil \\
 & \text{deadlines} \\
 & \forall \tau_i^k : \text{start}(\tau_i^k) + \text{execution_time}(\tau_i^k) \leq \text{deadline}(i, k) \\
 & \text{non-preemptiveness} \\
 & \forall \tau_i^k : \text{disjoint2}([\text{start}(\tau_i^k), \text{execution_time}(\tau_i^k), \text{node}(\tau_i^k), 1]) \\
 & \text{affinity} \\
 & \forall \tau_i^k : \text{node}(\tau_i^k) = \text{node}(\tau_i^k) \\
 & \text{communication} \\
 & \forall \tau_i^k, \tau_j^l \text{ with communication constraints:} \\
 & \text{delay}(\tau_i^k, \tau_j^l) = \text{message_size}(i, j) \cdot c_{\text{speed}} \cdot B_{i,j}^{k,l} \\
 & B_{i,j}^{k,l} \Leftrightarrow \text{node}(\tau_i^k) \neq \text{node}(\tau_j^l) \\
 & \text{start}(\tau_i^k) + \text{execution_time}(\tau_i^k) + \text{delay}(\tau_i^k, \tau_j^l) \leq \text{start}(\tau_j^l) \\
 & \text{start_message}(\tau_i^k, \tau_j^l) \geq \text{start}(\tau_i^k) + \text{execution_time}(\tau_i^k) \\
 & \text{start_message}(\tau_i^k, \tau_j^l) + \text{delay}(\tau_i^k, \tau_j^l) \leq \text{start}(\tau_j^l) \\
 & \text{serialized}([\text{start_message}(\tau_i^k, \tau_j^l)], [\text{delay}(\tau_i^k, \tau_j^l)]) \\
 & \text{energy} \\
 & \forall \tau_i^k : \text{energy}(\tau_i^k) = \left\lceil \frac{\text{execution_time}(\tau_i^k) \cdot p(\tau_i^k)}{f_{\text{max}}} \right\rceil \\
 & p(\tau_i^k) = a(\text{node}(\tau_i^k)) \cdot \text{volt}(\text{node}(\tau_i^k), \text{level}(\tau_i^k))^2 \\
 & f(\text{node}(\tau_i^k), \text{level}(\tau_i^k)) \\
 & \forall \eta_p \text{ with energy constraints: } \sum_{\forall \tau_i^k} B_{i,p}^{k,l} \cdot \text{energy}(\tau_i^k) \leq \text{max_energy}(p) \\
 & B_{i,p}^{k,l} \Leftrightarrow \text{node}(\tau_i^k) = p \\
 & \text{resources} \\
 & \forall \rho_r \forall \tau_i^k \text{ that use } \rho_r : \\
 & \text{cumulative}([\text{start}(\tau_i^k)], [\text{execution_time}(\tau_i^k)], [\text{amount_used}(i, r) \\
 & \text{capacity}(r)]) \\
 & \text{cumulative}([\text{start}(\tau_i^k) \cdot B_{i,r}^k], [\text{execution_time}(\tau_i^k) \cdot B_{i,r}^k] \\
 & [\text{amount_used}(i, r) \cdot B_{i,r}^k], \text{capacity}(r)) \\
 & B_{i,r}^k \Leftrightarrow \text{node}(\tau_i^k) = \text{node}(r) \\
 & \text{node}(\tau_i^k) \in \{\text{node}(r)\} \\
 & \text{locality} \\
 & \forall \tau_i^k \text{ with locality constraints:} \\
 & \text{node}(\tau_i^k) = p \\
 & \text{integer} \\
 & \text{node}(\tau_i^k) \in [1, m] \\
 & \text{level}(\tau_i^k) \in [1, v] \\
 & \text{start}(\tau_i^k) \in [\text{period}(i) \cdot (k - 1), \text{period}(i) \cdot k] \\
 & \text{start_message}(\tau_i^k, \tau_j^l) \in [0, \text{lcp}]
 \end{aligned}$$

co-residence (several tasks should be assigned to the same processor), and exclusion (tasks that be replicated for fault-tolerance cannot be assigned to the same processor).

To solve the allocation problem, different basic search strategies based on some criteria have been defined and compared. These criteria are:

- mindomain (the search strategy chooses the variable with the smallest domain size),
- maxmemory (the search strategy chooses the variable with the biggest memory need),
- maxutilization (the search strategy chooses the variable with the biggest processor utilization), and
- learning (the search strategy uses the nogoods learned during the resolution).

4.2. Game Theory

Game theory (GT) is a multi-player decision theory and any game must specify the players of the game, the information and actions available to each player at each decision point, and the payoffs for each outcome. The players participate in a Game in order to get maximum benefit by selecting a reasonable action. Equilibrium is a key concept in game theory. As optimization problems seek to optimal solutions, a game looks for equilibrium. The latter defines a stable state in which either one outcome occurs or a set of outcomes occur with known probability.

Game theory includes two main branches that are non-cooperative and cooperative (Game Theory, n.d).

A game is cooperative if the players are able to form binding commitments externally enforced through contract law. A game is non-cooperative if players either cannot form agreements or if all agreements need to be self-enforcing. Cooperative games are often analyzed through the framework of cooperative game theory, which focuses on predicting which coalitions will form, the joint actions that groups take, and the resulting collective payoffs.

In the context of RT scheduling problem, GT can be applied by considering each task as a selfish agent free to select its own processor. Every processor declares its scheduling policy in advance, and this induces a simultaneous-move game between the tasks. The strategy of a task consists of choosing the processor on which it will be executed. Each task wants to optimize its cost function and the game reaches an equilibrium where no task can optimize its cost function by migrating to another processor. The goal of a system designer is to design a scheduling policy on each processor such that ineffectiveness resulting from the selfish behavior is as minor as possible (Kulkarni, 2015).

(Ahmad & Ranka, 2008) formulated the RT scheduling problem on heterogeneous multi-cores architecture as a cooperative game theory problem to minimize the energy consumption and the makespan (the time required to execute all the tasks) simultaneously, while maintaining deadline constraints. The RT scheduling is based on a scenario where it is assumed that the schedule of the parallel application that minimizes the execution time (makespan) on the multi-cores architecture is known and the objective is to find a new schedule (with allowable reduction in schedule length) that tries to minimize the energy consumption of the entire architecture using the DVS technique. Hence, the objective is to optimize the cumulative performance rather than to satisfy individual cores. The paper showed that for such RT scheduling problem, a simple cooperation for joint resource allocation was significantly better than no cooperation. So the cores of the architecture acting as players can benefit only if the overall cores can benefit from the execution of tasks. This collective benefit can be achieved very efficiently via the concept of NBS (Nash Bargaining Solution). NBS is a solution to a game in which players use bargaining interactions to demand a portion of some entity. The interactions continue until a resolution is met and all the players achieve their demands. The remarkable property of the NBS is that it guarantees pareto-optimality and fairness. By converting the energy-aware RT scheduling problem into a cooperative game theory solution, the authors proved to guarantee pareto-optimal solutions in mere $O(nm \log(m))$ time (where n is the number of tasks and m is the number of cores).

(Wu, 2012) addressed the question of how to reduce the temperature difference between embedded processor multicores against real-time guarantee. For this end, they proposed a generalized tit-for-tat based corporative energy-aware scheduling game for multicore systems namely GTFTES (Generalized Tit-For-Tat Energy-aware Scheduling). In the proposed game, game players are the cores competing for different tasks whose number is always lower than the number of the processor cores. In the proposed scheduling scenario, each core submits a price that a task should pay for the executing service (in terms of temperature amount, which is estimated based on the ATMI thermal model, and deadlines guarantee). Naturally, the core bidding the lowest bids will win the auction. The auctioneer (the scheduler) firstly decides the winning cores, and then assigns the tasks to the cores. The auctioneer charges each core the harm they cause to other cores, and ensures that the optimal strategy for a task is to bid the value of the cores. Each player (core) decides to cooperate or retaliate

according to the hardness factor h . Specifically, after each round of the game, each player will calculate the proportion that the players who cooperate (this proportion is specified as hardness h). If his over a predefined value, the player will decide to cooperate in the next game round. Otherwise, he will choose to retaliate without considering the power status of the processor. Simulations results showed that the proposed game could reduce the temperature difference between different groups of cores, which effectively avoids the local hotspot of a processor. (Abdeyazdan et al., 2013) applied the idea of Nash equilibrium in game theory for static tasks graph pre-scheduling on homogeneous multiprocessor architecture. Given a game with strategy sets for players, a pure Nash equilibrium is a strategy profile in which each player deterministically plays her chosen strategy and no one has an incentive to unilaterally change her strategy. The proposed algorithm tries to establish a trade-off between time and energy. In order to minimize the energy consumption, the number of processors should be minimized but while minimizing the makespan, the number of available processors may be increased. To determine the optimal number of processors, the algorithm consider each level of a task graph as a selfish player attempting to get suitable number of processors to execute its tasks in parallel. The overall benefit of applying the processors is maximized when the Nash equilibrium is reached among the levels. Afterward, considering the determined number of processors, the algorithm again applies the Nash equilibrium concept to determine the appropriate merging of tasks with their parents. In this case, each task is considered as a player and the best merging is the one it minimizes tasks earliest start time.

4.3. Artificial Immune Systems

Artificial immune systems (AIS) are a class of computationally intelligent, rule-based machine learning systems derived from the principles inspired by the human immune system. The till-known function of AIS is to protect the host organism against attack by invading pathogens, and that the immune system is comprised of several interacting subsystems which are closely linked with the endocrine and the central nervous systems. Some views that are more radical suggest that the immune system is part of a larger cognitive system that has a fundamental role in the maintenance of the body and the preservation of homeostasis (Artificial Immune Systems, n.d).

Traditionally, the immune system is thought to comprise a series of layers, each providing protection against a specific type of pathogenic attack (Lay, 2009).

Several AIS algorithms for combinatorial optimization have been designed and most of them are based on the clonal selection and immune network principles.

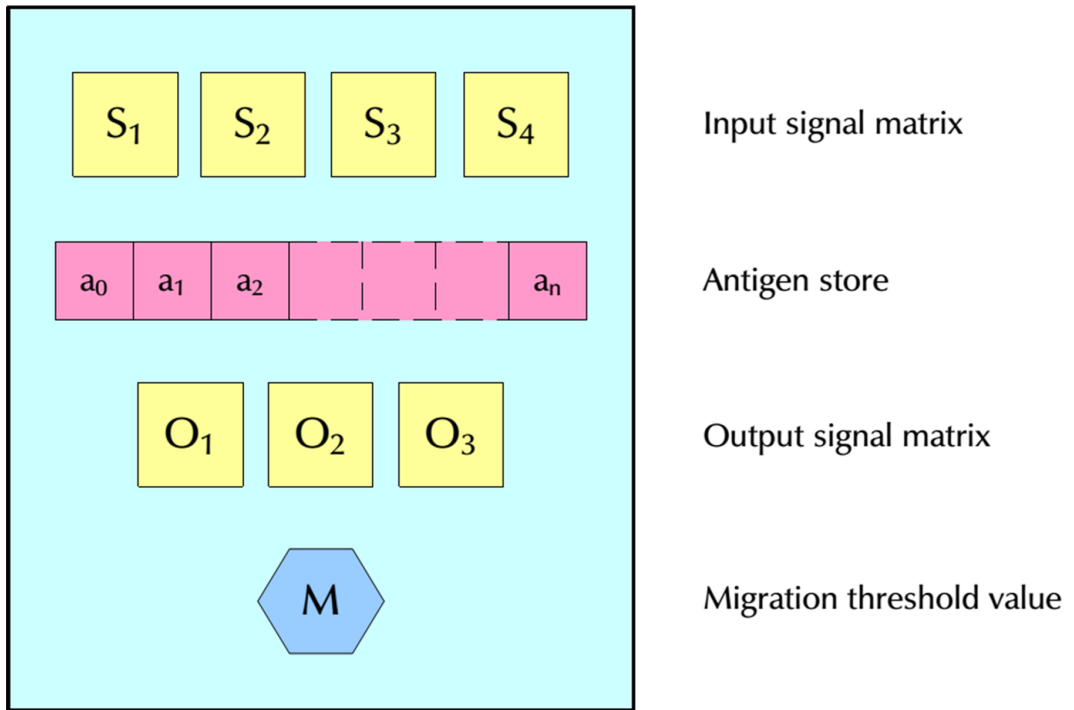
One of the most known AIS algorithms for optimization is the DCA (Dendritic Cell Algorithm), an algorithm derived from the operation of dendritic cells (DC).

DCs collect antigens, which are then presented to T-cells in a lymph node, along with information about the concentrations of cytokine signals associated with cell death, either necrotic or apoptotic. The information provided by DCs is used to determine whether the initiation of an immune response against those antigens is necessary.

The general function of the DCA is to provide an indication of danger levels associated with different parts of the system. Based on the idea of a DC detecting chemical signals, the DCA makes use of virtual DCs and virtual signals derived from various aspects of the system being monitored. As shown in figure 8, each DC takes the form of a data structure, consisting of a matrix of input signals, a matrix of output signals, an antigen store and a migration threshold.

DCA functioning is based around a regular cell update cycle. During each update cycle, each DC belonging to the population updates its input signal matrix and antigen store by evaluating the values passed to it from the environment and then use some or all of the input signals, weights them according to a set of weighting values and combines them to produce the appropriate output value for its required output signals. Thus, the algorithm can be adjusted to favor some specific signals or categories of signals.

Figure 8. Data structure of single DC



For example, a signal can be assigned a high weighting value if it always indicates the presence of danger in the system. This high weight will enable the presence of this signal to render the DC mature. Each new DC is considered immature and has a maximum lifespan defined as a number of cell update cycles. At the end of this lifecycle, if this DC could not reach maturity, it will be transformed into the semi-mature state.

Nevertheless, if at any point in its lifecycle the value of a DC's output signal exceeds a specific migration threshold, it will be transformed into the mature state, after which it presents its antigen and signal values to a lymph node structure to derive a measure of danger for that antigen. In the framework of a PhD thesis, (Lay, 2009) suggested some modification on the conventional DCA, so it will be possible to apply it to reduce tasks deadlines overrun problem and hence improve the reliability in a non-preemptive and fixed-priorities tasks RT scheduling. The DCA should be executed in parallel with the system, which it is monitoring, such that it is able to predict or detect overruns as the system operates. The proposed algorithm is shown in figure 9. In each cycle, the DCA monitoring component chooses a random pool of DCs from the overall pool, which are to be evaluated in that cycle. It then invokes the task scheduler, and for each task currently in the run queue, it determines the values for the DC input signals based on the state of the task at that point in the simulation, and updates the signal and antigen matrices of each selected DC. Once the input signals have been processed for all the tasks in the run queue, the output signals for all the selected DCs are then evaluated, and the lifecycle counter for each selected DC is incremented. For each DC, the output signal value is compared against the DC's maturation threshold, and if the output signal value is greater than the maturation threshold, the DC becomes mature. The DC becomes semi-mature if the lifecycle counter is greater than the lifecycle threshold. Once a DC has reached maturity or semi-maturity, its antigen is evaluated, and the DC is reset and returned to the population.

The random selection of a subset of DCs in each cycle and the non-deterministic characteristics of the task execution, guarantees a continuous evaluation of the status of the run queue throughout the simulation.

In the context of the deadline-overrun problem, the DCA is applied with three signal categories:

- PAMP (Pathogen Associated Molecular Patterns) which corresponds to the actual overrun occurring when the task completion time is greater than the task deadline.
- Danger which corresponds to a potential overrun occurring when at any point from the task release to completion, the worst-case response time is greater than the time to deadline.
- Safe, when at all points from the task release to completion, the worst-case response time is less than the time to deadline.

The behavior of the DCA depends strongly on its input parameters especially the weights and the threshold values.

Figure 9. DCA pseudocode, amended for application to the deadline overrun problem

```
initialise cell population;  
for each clock cycle loop  
  select x DCs from DC population  
  
  for each task in run queue loop  
    compute input signals;  
    for each selected cell loop  
      update signal matrix;  
      update antigen matrix;  
    end loop;  
  end loop;  
  
  for each selected cell loop  
    increment cycle count;  
    compute output signal;  
    if output signal > migration threshold then  
      register danger output;  
      reinitialise cell;  
    else if cycle count >= lifecycle threshold then  
      register safe output;  
      reinitialise cell;  
    end if;  
  end loop;  
  
end loop;
```

4.4. Cellular Automata

Cellular Automaton (CA) is a computation model used to model and simulate behaviors of complex dynamic and parallel systems. CA can be seen as a grid of cells together form a neighborhood. Each cell has a state and can change its state according to its neighborhood state. This is called a local transition rule. By applying local rules on cells synchronously or asynchronously, the CA changes its global state and some complex emerging behaviors can be produced. This is a primary inherent characteristic of CA. In its simplest form, a cellular automaton is defined as $CA = (D, S, N, \xi, F)$ where: D is the grid dimension, S is the set of cells states, ξ is the local transition function and F is the global transition function (Cellular automaton, n.d).

Many authors have been interested in applying CA to solve the traditional multiprocessor scheduling problem. But only few works targeting the application of CA to solve RT scheduling problem for ES. Existing literature works differ according to how the neighboring, the cell states and the local transition rules are defined and how they are evolved.

It is well known that one of the difficulties in using CA is the exponentially increasing number of rules with increasing number of processor and neighborhood radius. For this reason, it is not wonder if we find many authors have been combine optimization metaheuristics such as genetic algorithms and their variants notably quantum inspired genetic algorithms or other swarm based techniques to solve the combinatorial explosion of rules in CA.

The first application of CA to solve the problem of scheduling is due to (Seredynski, 1998).

The author proposed to use cellular automaton (CA) as a tool for designing distributed scheduling algorithms for allocating parallel program tasks in multiprocessor systems. To do so, a program graph is considered as a CA containing elementary automata interacting locally according to some rules. The proposed algorithm has two phases: in the first phase, it tries to discover effective rules for the CA by a genetic algorithm. In the second phase, for any initial allocation of tasks in a multiprocessor system, the CA-based scheduler attempts to find an allocation minimizing the total execution time of the program in a given system topology. The proposed approach was validated for a number of program graphs scheduled in a two-processor system. (Swiecicka et al., 2006) presented a CA-based multiprocessor scheduler working in three modes that are learning, normal operating, and reusing. In the learning mode, knowledge about solving a given instance of the scheduling problem is extracted and coded into CA rules. A genetic algorithm is used to discover the most suitable CA rules. Discovered rules in the learning mode are used in the normal operating mode by CA-based scheduler for automatic scheduling without a calculation of a cost function, an optimal or suboptimal solution of the scheduling problem for any initial allocation of program tasks in the multiprocessor system. In the third mode, previously discovered rules are reused with support of an artificial immune system (AIS) to solve new instances of the problem.

(Ghafarian et al., 2009) proposed a scheduler that uses a two dimensional evolving CA based on ant colony optimization method to find optimal response time for some of well-known precedence task graph in the multiprocessor scheduling area.

(Agrawal & Rao, 2012) presented an irregular CA to find an energy-aware schedule. The rules for cellular automata are learned using a genetic algorithm. To solve the scheduling problem using CA, the tasks graph with precedence constraints and the architecture graph have to be mapped to the CA domain. An elementary task of the program is mapped to a cell in the CA space. The CA uses a neighborhood of size 5, which includes two parents and two children of the task, and the task itself. The state of the cell specifies the component to which the task is assigned. Initially, tasks are assigned randomly to the components. Then according to the rules and the neighborhood, CA evolve sequentially to reach a state, which gives a near-optimum schedule. In a more recent work (Boutekkouk, 2015), a CA-based solution is proposed to solve multi-cores energy-aware RT scheduling problem with periodic and independent tasks using the DVS (Dynamic Voltage Scaling) technique. The proposed CA is a two dimensional grid where each cell represents the information of the task allocation on a processor with a certain frequency mode. Thus, a cell state is a triplet (task, processor, frequency mode). The

user has the choice to choose between two scheduling policies that are DM (Deadline Monotonic) and EDF (Earliest Deadline First) when more than one task is allocated to the same processor. The author defined a set of local transition rules in order to simulate the CA. Such transition rules can for instance change the allocation, the priority or the frequency mode of a task. These rules try to optimize power consumption, balancing usage ratios of processors and minimize the number of tasks missing their deadlines. By applying these rules continually, some emerging behaviors or some good configurations with minimal power consumption are observed.

In the example of figure 10, there are five processors (p0... p4) and twenty tasks (T0... T19).

Each processor is characterized by a color and three frequency modes that are High frequency mode (H), Middle frequency mode (M), and Low frequency mode (L).

Figure 10. Example of a CA state in the first period using EDF

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19
P0	H												M			L	L			
P1				H							H									
P2			M																	
P3					M		H		M	M				M	L			H		L
P4		M			M			M											H	

4.5. Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning concerned with how software agents have to take actions in an uncertain, potentially complex environment in order to maximize the notion of cumulative reward. RL does not need labelled input/output pairs be presented and sub-optimal actions to be explicitly corrected. Instead, the focus is on finding a balance between exploration and exploitation of the current knowledge (Reinforcement learning, n.d).

Among efficient application of RL is to solve MDP (Markov Decision Process) without explicit specification of the transition probabilities (i.e. probabilities are not known in priori), in this case, the values of the transition probabilities are needed in value and policy iteration. Reinforcement learning can also be combined with function approximation to address problems with a very large number of states. Informally, a MDP is a discrete time stochastic control process, used for probabilistic modeling of decision making. More formally, an MDP (Markov decision process, n.d) is a quadruplet (S, A, P_a, R_a) where

S is a finite or infinite set of states,

A is a finite or infinite set of actions,

$P_a(s, s')$ is the probability that action a in state s at time t will lead to state s' at time $t+1$,

$$P_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$$

$R_a(s, s')$ is the immediate reward (or expected immediate reward) received after transitioning from state s to state s' due to action a . The core problem of MDPs is to find a policy for the decision maker: a function π that specifies the

action $\pi(s)$ that the decision maker will choose when in state s . Once a Markov decision process is combined with a policy in this way, this fixes the action for each state and the resulting combination behaves like a Markov chain.

The goal is to choose a policy π that will maximize some cumulative function of the random rewards, typically the expected discounted sum over a potentially infinite horizon:

$$E\left[\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_t + 1)\right]$$

Where we choose $a_t = \pi(s_t)$ (i.e. actions given by the policy) and the expectation is given over

$$s_t + 1 \sim P_{a_t}(s_t, s_t + 1)$$

Where γ is the discount factor satisfying

(Glaubius et al., 2010) considered the problem of learning near optimal RT scheduling when the system model is not known using MDP. In contrast to classical real-time scheduling approaches that are based on worst-case execution time (WCET) analysis, this work assumes that each task's duration obeys some underlying but unknown stationary distribution. Thus, RL suits well this situation. The tasks model consists of N tasks that require mutually exclusive use of a single common resource. Each task consists of an infinite sequence of jobs. Furthermore, many assumptions are made:

- inter-task job durations are independently distributed,
- intra-task job durations are independently and identically distributed,
- The scheduling is supposed not preemptive and each duration distribution must have bounded support on the positive integers.

The goal is to schedule jobs in order to preserve temporal isolation.

More formally, the RT scheduling problem is modeled as an MDP over a set of *utilization* states X . each state x is an n -vector $(x_1 \dots x_n)$ where each x_i is the total number of quanta during which task T_i occupied the shared resource since system initialization.

$$\tau(x) = \sum_{i=1}^n x_i$$

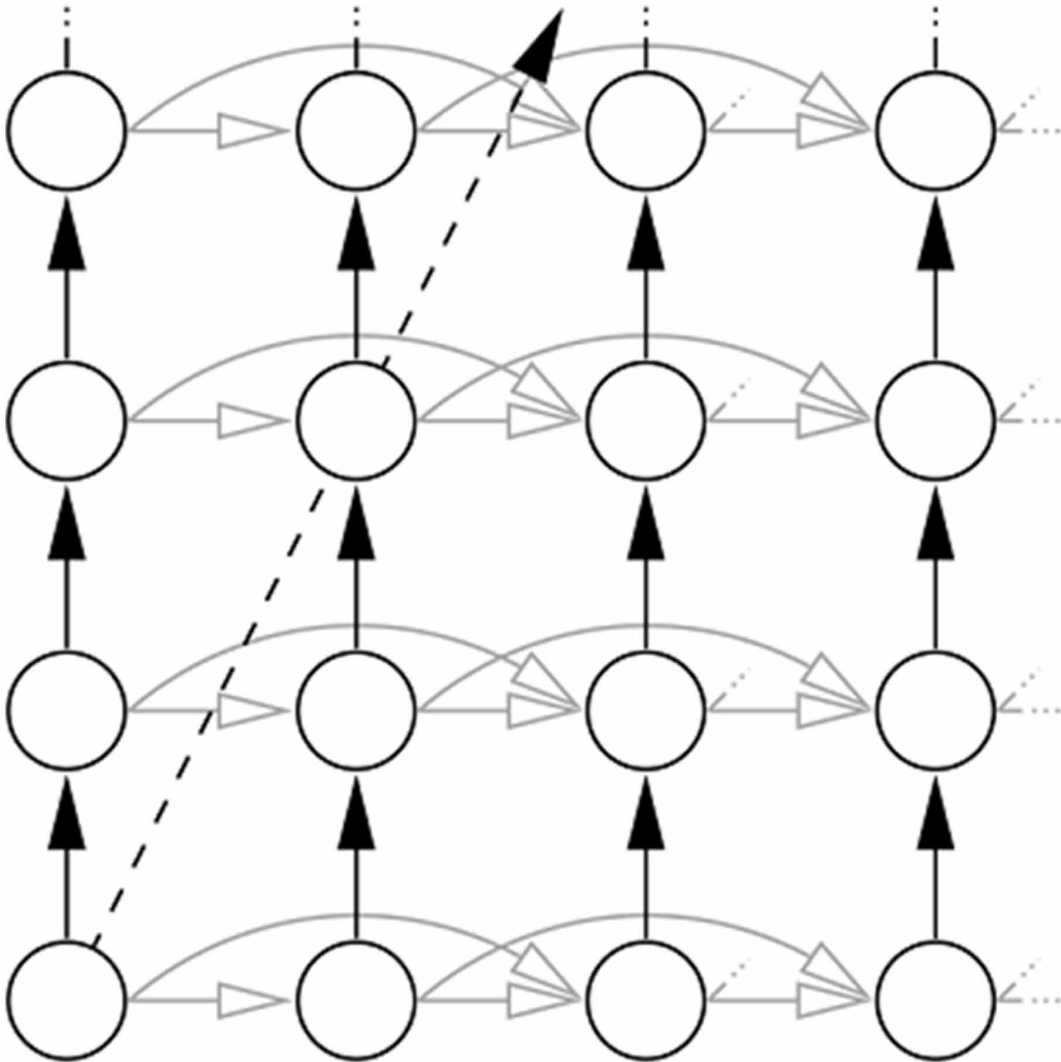
$\tau(x)$ denotes the total elapsed time in state x .

Each action I in this MDP corresponds to the decision to run task T_i . Transitions are determined according to task duration distributions, so that:

$$P(y | x, i) = \begin{cases} P(t | i)y = x + t\Delta_i \\ 0 & \text{otherwise} \end{cases}$$

Where Δ_i is the zero vector except that component i is equal to one. The cost of a state is its L_1 -distance from target utilization within the hyperplane of states with equal elapsed time $\tau(x)$

Figure 11. Illustrates the utilization state model for a problem with two tasks and a target utilization $u = (1, 2) = 3$ (that is, task T1 should receive 1/3 of the processor, and task T2 should receive the rest).



$$C(x) = \sum_{i=1}^n |x_i - \tau(x)u_i|$$

The target utilization defines a *target utilization ray* $\{\lambda u: \lambda \geq 0\}$. When the components of u are rational, this ray regularly passes through many utilization states. In Figure 11, for example, the utilization ray passes through integer multiples of $(1, 2)$. Every state on this ray has zero cost, and states with the same displacement from the target utilization ray have equal cost.

T1 (grey, open arrowheads) stochastically transitions to the right, while T2 (black, closed arrowheads) deterministically transitions upward. The dashed ray indicates the utilization target. This task scheduling MDP has an infinite state space and unbounded costs. However, an optimal policy can be estimated accurately using a finite state approximation. RL is used to integrate the model and

the policy estimation. In order to know how much experience is necessary before learned policies can be trusted. The authors address this issue by deriving a PAC (Probably Approximately Correct) bound on the sample complexity of obtaining a near-optimal policy.

In the context of a PhD thesis, (Tidwell, 2011) defined a Markov Decision Process (MDP) model that enables to derive value-optimal schedulers, and also provides a formal framework for comparing the performance of different scheduling policies. He equally showed how the problem structure allows to bound the number of states in the MDP by wrapping states into a finite number of exemplar states.

Q-learning

Q-learning is a reinforcement learning algorithm which does not require a model of the environment. It treats problems with stochastic transitions and rewards, without requiring adaptations. “Q” designates the function that returns the reward used to provide the reinforcement and can be understood as the “quality” of an action taken in a given state.

In Q-learning, the system consists of a finite state space S and a set of actions A . Selecting an action $a \in A$ at state $s \in S$ conducts the system to a new state with a reward or penalty.

A policy π is a mapping $\pi: S \rightarrow A$. For each state–action pair (s, a) , it maintains a value function $Q(s, a)$ that represents the penalty or reward. Based on the function value, the agent decides which action should be taken in current state to achieve the maximum rewards. The Q-value for each state–action pair is updated iteratively in the Q-table each time an action is issued and a penalty is received based on the following expression:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t (s_t, a_t) + \delta \left[p_t(s_t, a_t) + \min_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

Where $p_t(s_t, a_t)$ is the penalty received in state s_t with action a_t taken. α_t is the learning rate that determines what degree the newly acquired information will override the old information.

δ is the discount rate that determines the importance of future rewards and plays an important role when the environment is sequential. $Q(s_{t+1}, a_{t+1})$ is determined based on the action, which costs minimum Q-value. The next time when state s_t is visited again, the action with the minimum Q-value is chosen (Q-learning, n.d).

(ul Islam & Lin, 2015) proposed to use Q-learning algorithm to reduce the energy consumption in RT scheduling of periodic and preemptive tasks using a set of DVFS techniques on mono-core processor. The impetus behind applying Q-learning is because existing DVFS techniques work with different strategies and perform well under different conditions. However, a single DVFS technique is not always optimal under different workloads, dynamic slacks, and power settings. Furthermore, the variation in device configuration also affects the suitability of a given DVFS algorithm. Thus, the aim of the Q-learning algorithm is to take the advantage of different strategies used by each of the existing DVFS techniques by training the scheduler what and when selecting the more suitable DVFS policy in order to minimize the energy consumption while meeting tasks deadlines. In their experiments, the authors selected three hard real-time DVFS techniques that are CC, LA, and DRA. Results show that the proposed approach under some assumptions can save more energy than any single policy executing individually.

4.6. Artificial Neural Networks

Artificial Neural Networks (ANN) have proved their efficiency in classification and optimization problems with constraints. To solve optimization problems using ANN, one must first select an appropriate neural network model, and then map the cost functions and the constraints of a the

target problem into an energy function which represents a state function of the selected model. The mapping must be accomplished in such a way that while the energy function has reached a stable state (i.e. local minimum or as a global minimum), the corresponding cost functions and violations of the constraints are minimized.

However, for ES where resources are limited, the application of ANN is still questionable.

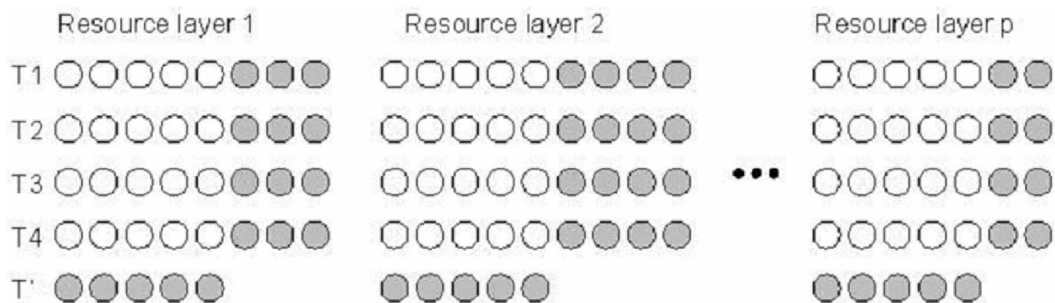
Because of the particularity of ES, any ANN-based solution should attentively select the appropriate number of neurons, layers and connections between them, and conceive the learning algorithm as simple as possible (Feng & al., 2005). Traditionally, the online scheduling problem can be modeled through ANN as follows:

- In the case of mono-processor architecture, Neurons n_{ij} are organized in a matrix form, with the size $N_T \times N_C$, where line i is the task T_i and the column j corresponds to schedule time unit j . The number of time units N_C is the least common multiple of all the task periods and N_T is the number of tasks.
- A neuron n_{ij} is considered active when the task T_i is being executed, during the corresponding time unit j .
- One line of neurons is added to model the possible inactivity of the processor during the schedule times. These neurons are called slack neurons.
- In the case of a homogeneous Multiprocessor architecture, several matrices arranged in layers are required to model the different execution resources.
- New slack neurons are then necessary to manage the exclusive execution of each task on resources. So for each couple (task T_i , resource j), C_{ij} new slack neurons must be added.

Slack neurons are added to ensure the network convergence when applying a k-out-of-N rule on each vertical line of neurons. k-out-of-N rule allows the construction of N neurons for which the evolution leads to a stable state with exactly k active neurons among N.

An example of network with p resource layers is shown in Figure 12. Grey circles represent slack neurons (Chillet et al., 2007).

Figure 12. Classical structure used to model the ANN. Grey circles represent slack neurons



However, this technique has two major drawbacks. The first is the important number of necessary slack neurons needed to model the problem. The second problem is the presence of several local minima when several rules are applied to the same set of neurons. These local minima are particular points of the energy function, which represent invalid solutions. To ensure convergence towards valid solutions, these minima must be detected and the network needs to be re-initialized. In order

to reduce the number of required neurons and avoid re-initializations of the network, many authors have proposed some modifications.

(Chillet et al., 2007) presented a model of ANN used for RT online scheduling on heterogeneous on chip multiprocessors. The proposed model is an adaptation of the Hopfield model and the main objective concerns the minimization of the neurons number to facilitate its hardware implementation. To do so, the authors proposed new constructing rules to design smaller neural network so the number of slack neurons is considerably reduced and independent of the period. Moreover, no re-initialization is necessary because there is no local minimum in the energy function. The authors proposed two modifications of the neural network structure. Firstly, a specific neuron is placed for each task and each type of execution resource, this neuron is called inhibitor neuron. The main idea consists in creating a mutual exclusion between the possible task instantiations on execution resources. Secondly, in order to remove the slack neurons which model the possible inactivity of the processor (idle cycles). The authors proposed the application of a k_1 -out-of- N_1 classical rule on the horizontal sets of neurons and an at-most- k_2 -among- N_2 rule on the vertical sets of neurons.

(Rehaïem et al., 2016) presented an ANN model which is a Hopfield model for energy-aware online RT non-preemptive scheduling with hard constraints based on the combination of DVS and Neural Feedback Scheduling (NFS) with the priority-energy earliest-deadline-first (PEDF) algorithm. A flexible BP (Back Propagation) learning algorithm is adopted in which only the symbol of gradient to the error function is needed rather than the increase amplitude of gradient.

The basic idea of the NFS is to allocate available resources dynamically among multiple real-time tasks based on feedback information about actual resource usage. The addressed optimization problem is to minimize the total energy consumed by the set of n tasks by optimally determining their start times, their voltages and corresponding execution speeds when scheduling them. PEDF is an extension of the well-known earliest deadline first (EDF) algorithm. It maintains a list of all released tasks and when tasks are released, the task with the nearest deadline is selected to be executed. A check is performed to see if the task deadline can be met by executing it at the lower voltage (speed). If the deadline can be met, PEDF assigns the lower voltage and the task begins execution.

4.7. Fuzzy Logic

Fuzzy logic is a superset of classical Boolean logic and extends it to deal with new issues such as the partial truth and uncertainty. The basic elements of fuzzy logic are linguistic variables, fuzzy sets and fuzzy rules. A fuzzy set is a set of pairs of elements generalizing the concept of a traditional set, allowing its components to have a partial membership defined as a membership function. The latter can be expressed as a curve that defines how each point in the input space is mapped to a membership value or a degree of truth between zero and one. The most common form of a membership function is triangular, trapezoidal and bell curves (Fuzzy logic, n.d).

Using fuzzy logic to solve any problem follows generally three steps that are fuzzification, computing fuzzy output functions by executing all applicable inference rules in the rule base and defuzzification. Fuzzification is the transformation of the digital inputs or the crisp set into a set of membership values in the interval $[0, 1]$ to corresponding fuzzy sets. The fuzzy inference rules describe the relationships between linguistic, inaccurate and qualitative expressions of system input and output and finally, defuzzification, which is the process that convert a fuzzy set to a crisp set.

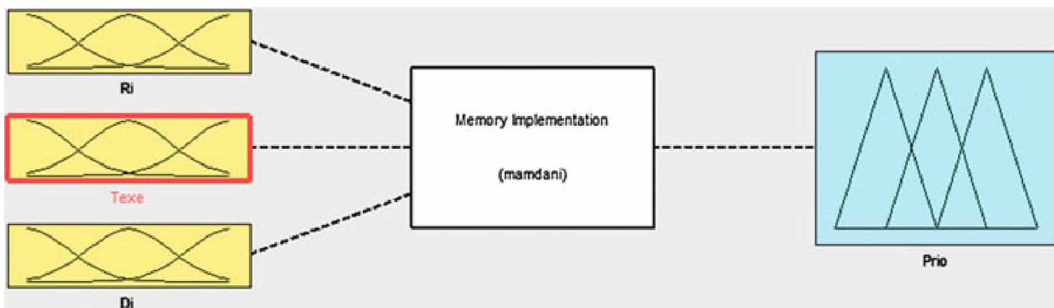
Literature on fuzzy logic application to solve the RT scheduling problem is relatively new. (Saini, 2005; Vijayakumar & Aparna, 2010; Blej & Azizi, 2016).

For RT systems with hard constraints, the objective was primary to guarantee timing constraints (deadline) respect however, for soft real time systems the objective is to minimize the mean response time of the system. The idea of applying fuzzy logic in RT scheduling problem seems very important. This is justified by the fact that ES are dealing with inaccurate and uncertain data. The latter include for example, the arrival times of tasks, the actual execution times of tasks which are generally very far from their execution times at the worst case, the best clock frequency of the processor which

leads to the minimal energy consumption, the right time to migrate a task to another processor, etc. Intrinsic uncertainty in real-time systems and in particular dynamic systems increases the difficulties of conventional scheduling algorithms to optimize performance and/or energy consumption. By integrating fuzzy logic into the real-time scheduling problem, the scheduler's decisions regarding choice of the best processor clock rate, priorities, and task migration dates can be improved considerably.

(Mehalaine & Boutekkouk, 2016) presented an energy aware fuzzy RT scheduling model for periodic independent tasks targeting mono-processor embedded architecture. The proposed algorithm functions on two steps. The first step uses fuzzy system to generate fuzzy priorities and the second step uses the outputs of the first one to schedule tasks with minimum energy consumption basing on the EDF* algorithm. Energy consumption is reduced by processor use with minimum speed without tasks deadlines missing.

Figure 13. Linguistics variables and corresponding output variable



In the proposed model, the input stage comprises three variables that are T_{exei} : the actual execution time of the task, which is expressed in processor cycles; R_i : the arrival date of the task and D_i : the relative deadline of the task as shown in Figure 13. The three input parameters decide the highest priority of the task from the tasks queue.

The arrival date of the task T_i : $V=R_i$, $X = [0, 1]$, $T_v = \{\text{already arrived, near, far}\}$.

The actual execution time of task T_i : $V = T_{exei}$, $X = [0, 1]$, $T_v = \{\text{large, medium, small}\}$ (relative to worst execution time). The relative deadline of the task T_i :

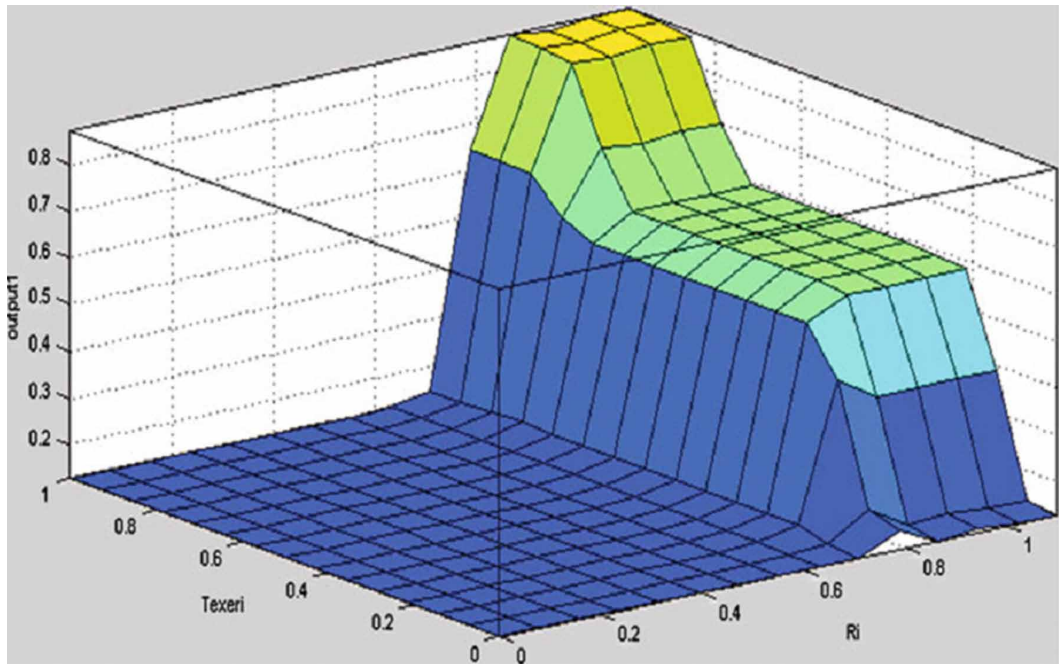
$V=D_i$, $X = [0,1]$, $T_v = \{\text{very close, close, medium, far}\}$ (relative to the moment t)

The output (fuzzy priorities): $V = Prio$, $X = [0, 1]$, $T_v = \{\text{high, medium, low}\}$

Twenty (20) fuzzy inference rules were defined. As examples of these rules are:

- R1: if ($D_i = \text{very close}$) then the fuzzy scheduling priority is high;
- R2: if ($R_i = \text{already arrived}$) and ($T_{exei} = \text{large}$) and ($D_i = \text{close}$) then the fuzzy scheduling priority is high;
- R3: if ($R_i = \text{close}$) and ($T_{exei} = \text{large}$) and ($D_i = \text{close}$) then the fuzzy scheduling priority is high;
- R4: if ($R_i = \text{close}$) and ($T_{exei} = \text{medium}$) and ($D_i = \text{close}$) then the fuzzy scheduling priority is average;

Figure 14. The set of decisions for the proposed fuzzy model



4.8. Multi Agent Systems

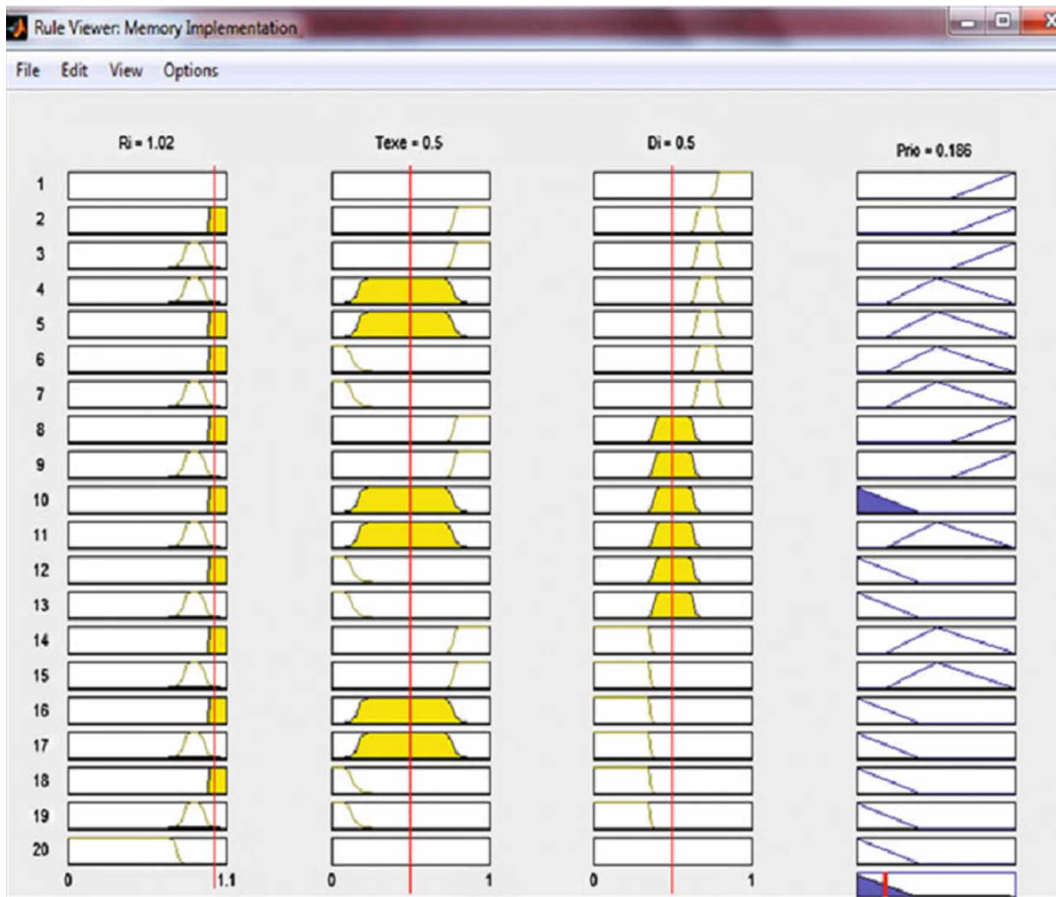
Multi Agents Systems (MAS) are becoming mainstream for complex large scale and distributed systems modeling and simulation. One major benefit of using MAS is their ability to model complex interactions, and many aspects such as reactivity, proactivity, autonomy and decision making explicitly. Furthermore, MAS offer a pool of design methodologies and associated tools and platforms helping designers to efficiently develop MAS-based systems. MAS have been extended to cover a large class of systems including RT and cyber physical systems.

(Jin et al., 2009) developed a software RT scheduler agent. Tasks should firstly register in the agent. Then, after the agent agrees for a reasonable execution sequence, it synchronizes all tasks and schedules them. The schedule module is implemented with a soft Rate Monotonic Scheduling (RMA) algorithm. The latter can lead client task to miss its deadline when the new request cannot be satisfied with current schedule group. However, it at least can avoid the case that new task contest with scarce resources, thus guarantee not to deteriorate the whole system performance. When multi-access requests happened in designate intervals, the agent can figure out whether or not the current tasks could be merged. If they can meet their time constraints, then it computes their priorities and schedules them sequentially. Else, the tasks should be scheduled in a new group.

(Chniter et al., 2018) proposed a multi-agent adaptive architecture to handle dynamic reconfigurations and ensure the correct execution of the concurrent real-time distributed tasks under energy constraints. Tasks are assumed periodic, independent and non-preemptive.

The proposed architecture is supposed to optimally allocate tasks to processors while determining for each task the execution speed, the start time, the finish time and the effective execution duration on the target processor. A token ring topology is used to minimize the exchanges of messages among all entities. The hardware platform is composed of identical processors where each one has a fixed number of available scaling factors. The proposed architecture integrates a centralized scheduler agent (ScA) and a Reconfiguration Agent (RAp).

Figure 15. Defuzzification



The ScA is considered as the common decision making element for the scheduler. In this agent, the computation is centralized to avoid any error. ScA is responsible for the management of the consumed energy in the system, ensuring the calculation of the solution, sending statistics on the current state, updating a non-feasible solution, and communicating the solution to each RAP.

Once a request is received from RAP, the scheduler triggers proactively the coordination module and the solver, which is based on constraint programming and simulated annealing research techniques. The role of the RAP consists in locally applying the addition removal-update of real-time tasks to adapt the related device and the whole system to its environment.

However, these functional reconfiguration scenarios may not respond to the time and power requirements and can push the system to an infeasible state. In this case, the RAP coordinates and requests a help from the scheduler agent, which proposes the required solutions.

(Cavaresi et al., 2018) studied the deadline-missing rate occurring in general-purpose setups, evaluated on an agent-based simulator developed on OMNET++, named MAXIM-GPRT.

The obtained results strengthen the motivations for adopting and adapting real-time scheduling mechanisms as the local scheduler within agents.

4.9. Evolutionary Algorithms

Evolutionary Algorithms (EA) usually begin with a population of organisms (initial solutions) and then allow them to mutate and recombine, selecting only the fittest to survive each generation. The well-known evolutionary algorithms are genetic algorithms (GA), genetic programming, evolution strategies (ES), evolution programming and differential evolution (DE). Genetic algorithms (GA) are evolutionary algorithms which generate near optimal solution of a problem by a guided random search method where elements (i.e. individuals) in a given set of solutions (i.e. population) are randomly combined and modified until some termination condition is achieved. The population evolves iteratively in order to improve a given fitness function of its individuals. GA reflect the “survival of the fittest” competition mechanism. In fact, many studies have showed the efficiency of GA-based scheduling compared to other ones. (Monnier et al., 1998 ; Zomaya et al., 1999 ; Boutekkouk & Bounabi, 2014 ; Samal et al., 2014 ; Pradhan et al., 2015 ; Zhiyu & Li, 2016 ; Kaur & Singh, 2019).

However, the efficiency of GA for RT scenarios has been questionable since the search time for good solutions using GA is relatively long and this can lead to tasks deadlines missing. In addition, it is well known that GA are very sensitive to many input parameters and if these parameters are not well chosen and controlled, GA can itself diverge from optimal or near optimal solutions. In their former applications, GA were used to minimize the total execution time (makespan) in the multiprocessor scheduling problem under strict hypothesis. In the real time context, several works have tried to apply GA to primarily minimize the response time mean and the number of tasks missing their deadlines. Other works have applied GA to optimize multiple objectives (multi-objective optimization) including time, energy consumption and reliability. Existing works differ in solution or chromosome coding (i.e. binary versus real coding), the population size, the method of selection, the crossover (one point, two points, etc...), the mutation, the reparation mechanism, the halt criterion, the adopted multi-objective optimization technique (weighted sum versus based-Pareto, etc.).

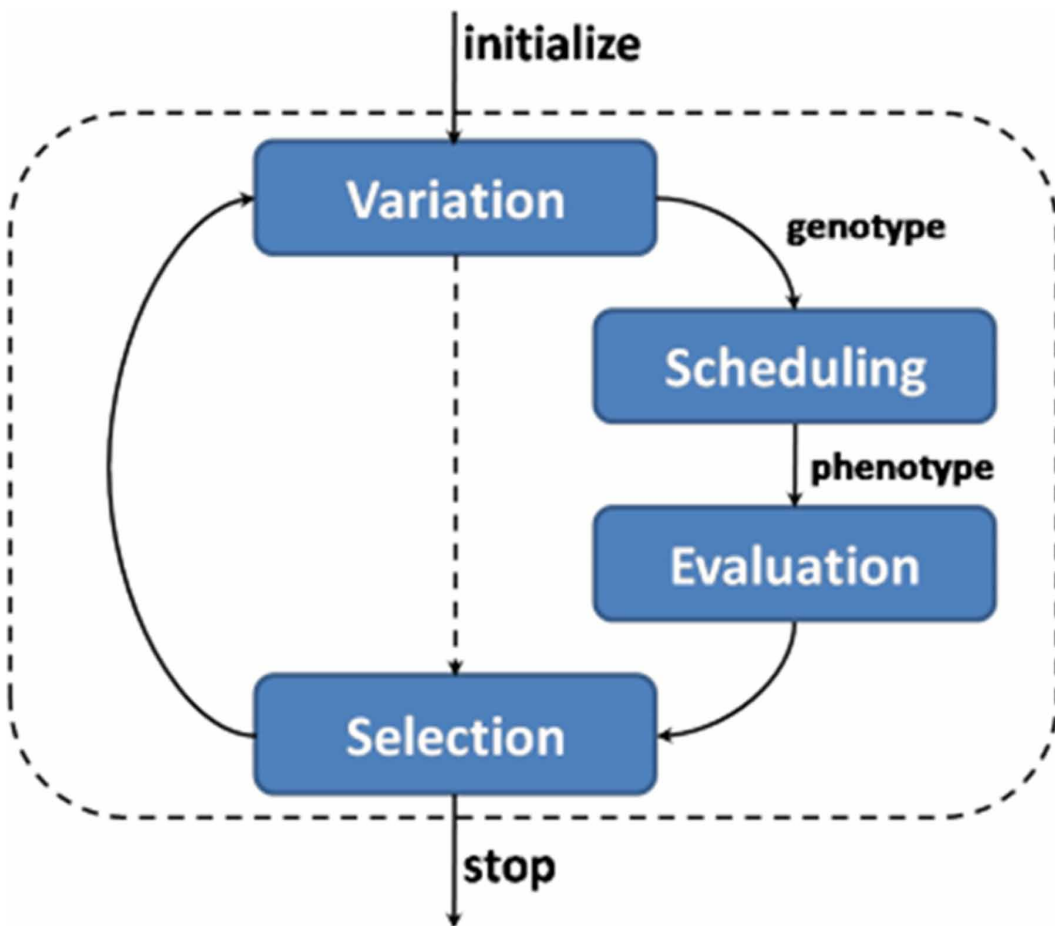
(Dahal et al., 2008) proposed to use GA incorporating traditional scheduling heuristics to generate a feasible schedule. The scheduling algorithm considered, aims in meeting deadlines and achieving high utilization of processors. The architecture is assumed consisting of identical processors connected through a shared medium and the tasks are aperiodic, independent and non-preemptive. Each task is characterized by its arrival time, ready time, worst-case computation time and its relative deadline. The algorithm considers a set of tasks from the sorted list to generate an initial population. In the initial population, each chromosome is generated by assigning each task in the task set to a randomly selected processor and the pair (task, processor) is inserted in a randomly selected unoccupied locus of the chromosome. GA operators are then applied to the population of chromosomes until a maximum number of iterations have been completed. When applying GA operators to the population, reproduction is applied first followed by crossover, partial-gene mutation, sublist-based mutation and then order-based mutation. In each iteration, the algorithm sorts the tasks in the chromosomes based on their deadlines, evaluates and sorts the chromosomes based on their fitness value. The tasks that are found infeasible (exceed their deadline) are removed from the chromosomes so that they are not reconsidered for scheduling.

(Huang et al., 2011) tackled the problem of analysis and optimization of fault-tolerant task scheduling for multiprocessor embedded systems. A Binary Tree Analysis (BTA) is proposed to compute the system-level reliability in the presence of software/hardware redundancy. The BTA is then integrated into a multi-objective evolutionary algorithm (MOEA) via a two-step encoding to perform reliability-aware design optimization. The optimization results contain the mapping of tasks to processing elements, the exact task and message schedule and the fault-tolerance policy assignment. In addition, the authors proposed a virtual mapping technique to take both permanent and transient faults into account. The EA is performed in two main steps: production of new solutions by varying existing solutions and selection of good solutions based on their fitness (figure 16). Using MOEA, the schedule is encoded as a chromosome. Nevertheless, a direct encoding of a schedule requires a very large chromosome, which affects the optimization efficiency. Therefore, instead of encoding the

entire schedule, the algorithm place only partial information, namely the mapping and fault-tolerance policy, into the chromosome. A scheduler is embedded to transform the chromosome to an optimized schedule and the resulting schedule is then utilized for fitness evaluation (i.e. reliability analysis).

(Boutekkouk & Bounabi, 2014) presented a multi-objective genetic algorithm to optimize the performance of a time-driven real-time distributed embedded system with mixed constraints and a shared bus based on the DVS technique. The three objectives to minimize are the energy consumption, the average response time and the number of tasks missing their deadlines.

Figure 16. Work flow of EA-based Optimization



4.10. Swarm Intelligence

Swarm Intelligence (SI) can be defined as the collective behavior of decentralized and self-organized swarms. Bird flocks, fish schools and the colony of social insects such as ants and bees are some well-known examples of SI. The more attractive feature of SI is the fact that each individual agent has a simple cognitive capacity but the cooperation of all agents lead to some complex emerging behavior as self-organization. The individual agents of these swarms behave without supervision and each of these agents has a stochastic behavior. The intelligence of the swarm lies in the networks of interactions among these simple agents and between agents and the environment. In the context of

RT scheduling for ES, numerous works have been investigated the idea of adopting SI for scheduling performances optimization especially ant colony, PSO and bees colony optimization meta-heuristics.

4.10.1 Ant Colony Optimization

Ant Colony Optimization (ACO) is relatively a recent technique, based on a probabilistic decision process, originally introduced to solve the Traveling Salesman Problem. ACO is inspired from the cooperative behavior of real ants for nourishment foraging by which ants start from their nest going in random directions, depositing pheromones. As time goes by, the shortest path to the food will be concentrated by pheromones, encouraging the other ants to follow this path instead of longer tracks. The ACO heuristic seems appropriate for problems in which the solution can be found via subsequent decisions. In fact, the amount of pheromone models the probability, for each decision point or admissible choice. All the decisions are initialized with this probability. Then, the process is iteratively re-started to construct different solutions (each ant constructs a solution). Typically, the probability is generated according to the following formula (Ant colony optimization algorithms, n.d):

$$p_{x,y} = \frac{[\tau_{x,y}]^\alpha * [n_{x,y}]^\beta}{\sum l \in \Omega[\tau_{x,l}]^\alpha * [n_{x,l}]^\beta}$$

Where x is the current point (state) in the decision process, and y is the candidate destination, $\eta_{x,y}$ models the desirability of state transition from x to y (typically equals to the inverse of the distance between x and y). $\tau_{x,y}$ is the amount of pheromone deposited for transition from x to y .

α is a parameter to control the influence of $\tau_{x,y}$ while β is a parameter to control the influence of $\eta_{x,y}$. $\tau_{x,l}$ and $\eta_{x,l}$ represent attractiveness and the track level for the other possible state transitions. Ω_x contains all the choices in the current point. At the end of each iteration, the results are ranked and the pheromones updated through different policies. In general, the pheromones are updated as follows:

$$\tau_{x,y} = (1 - \rho) * \tau_{x,y} + \epsilon$$

Where ρ is the pheromone evaporation coefficient. ϵ is a term usually proportional to the quality of the solution to maintain consistency among different iterations. In this way, only the best choices are reinforced and the others are penalized through evaporation.

Regarding the application of ACO to optimize RT scheduling, many authors have believed that ACO seem very appropriate to optimize scheduling of tasks in soft real-time systems since these algorithms provide inherent parallelism and robustness. In addition, ACO are adaptive and can easily be tuned to any domain-specific problem.

(Ferrandi et al., 2010) proposed an ACO heuristic that, given a heterogeneous multiprocessors architecture and an application modeled as an acyclic tasks graph, executes both scheduling and mapping to optimize the overall execution time of the application (makespan). The heuristic is compared with several other heuristics like simulated annealing, taboo search and genetic algorithms, on the performance to reach the optimum value and on the potential to explore the design space. The authors stated that the approach obtains better results than other heuristics by at least 16% in average, despite an overhead in execution time.

(Umrani et al., 2013) presented an ACO based approach for generating a feasible RT schedule that ensure load balancing across the processors and deadlines respect for all tasks. The processors are assumed heterogeneous. The algorithm pseudo-code is shown in figure 17.

The obtained results of simulation showed that the proposed ACO algorithm performs better than the FCFS (First Come First Served) algorithm with respect to the wait time.

Figure 17 . Pseudo-code for ACO algorithm for Multiprocessor RT scheduling problem

```
Repeat  
Do while (solution is not converged) and (reasonable amount of time not elapsed)  
Update the pheromone based on the quality of each feasible schedule generated in  
the previous iteration  
Generate the  $\tau$  matrix for the current iteration  
For each ant  $K$   
For each task  $i$   
Select the processor stochastically using the  $\tau$  matrix  
If the schedule obtained by an ant is feasible (individual processor utilization and  
deadline compliance of tasks are satisfied) then compute its quality.  
Else set the quality for zero (infeasible schedule).  
Calculate the standard deviation using the quality of all the schedules.  
End while  
Until (the standard deviation is less than a threshold value or the number of  
iterations exceed a particular predefined value)
```

4.10.2 Bee Colony Optimization

Bee Colony Optimization (BCO) (Bolaji et al., 2013) is a combinatorial optimization metaheuristic inspired by the behavior of honeybees in the nature. In a honeybee colony, forager bees examine the environment for flower paths and if they find a good source of food, they share it with other bees. As soon as, the forager bees come back to the hive, they share the discovered information about food sources by a special movement named waggle dance. The latter brings meaningful information like direction, distance, quantity and quality of the food source and are shared with respect to other bees. The general Artificial BCO algorithm is presented as follows (Karaboga et al., 2012):

Initialization Phase

Repeat

Employed Bees Phase

Onlooker Bees Phase

Scout Bees Phase

Memorize the best solution achieved so far

Until (Cycle=Maximum Cycle Number or a Maximum CPU time)

In the initialization phase, the population of food sources is initialized by artificial scout bees and control parameters are set. Typically, ABC consists of three control parameters that are the population size (i.e. the number of food sources), the maximum cycle number (the maximum number of generations) and *limit* which is used to determine the number of allowable generations for which each non improved food source is to be abandoned. In the employed bees phase, each employee bee is assigned to its food source and in turn, seeks for new food sources having more nectar within the neighborhood of the food source in its memory. After producing the new food source, its fitness is evaluated and a greedy selection is applied between it and its parent. After that, employed bees share their food source information with onlooker bees waiting in the hive by dancing on the dancing area. In the onlooker bees phase, artificial onlooker bees probabilistically choose their food sources depending on the information provided by the employed bees. For this purpose, a fitness based selection technique can be used, such as the roulette wheel selection method. After a food source for an onlooker bee is probabilistically chosen, a neighborhood source is determined, and its fitness value is evaluated. As in

the employed bees phase, a greedy selection is applied between two sources. In the scout bees phase, employed bees whose solutions cannot be upgraded through the *limit* parameter become scouts and their solutions are abandoned. Then, the scouts start to search for new solutions, randomly. Hence, those poor sources (i.e. initially poor or have been made poor by exploitation) are abandoned and negative feedback behavior arises to balance the positive feedback. These three steps are repeated until a termination criterion is satisfied (i.e. a maximum cycle number or a maximum CPU time).

According to literature, only few works have been interested in applying BCO to optimize RT scheduling. In fact, the BCO was used as a local search method in a global search meta-heuristic or combined with another optimization heuristic.

(Kazemi et al., 2016) proposed to hybrid between BCO and simulated annealing to optimize RT scheduling of non-preemptive tasks with soft constraints on heterogeneous multiprocessor platform. The proposed algorithm tries to minimize five parameters that are the total tardiness of tasks, the total number of utilized processors, the total completion time, the total waiting time of tasks, and the total waiting time of processors. The impetus behind the combination between ABC and simulated annealing is to avoid trapping in local minima points and improve the convergence speed of both BCO and SA. The proposed multi-objective optimization uses the weighted sum method. Simulation results demonstrated the efficiency of the proposed methodology as compared with the existing scheduling algorithms.

4.10.3 PSO

PSO algorithm is inspired by a social behavior of a group of migrant birds. It imitates the communication of the real birds when they are flying together. Each bird moves towards a certain direction; when in communication, it determine the best position. Therefore, each bird depends on the current position

Figure 18. Pseudo-code for PSO algorithm

```

Initialize the population randomly.
DO
{
For each particle.
{
Calculate fitness value
If the fitness value is better than the best fitness value ( $P_{best}$ ) in history then set current
value as the new  $P_{best}$ .
}
Choose the particle with the best fitness value of all particles as the  $G_{best}$ .
For each particle.
{
Calculate new velocity:
 $V_{new} = W \cdot V_{old} + C1 \cdot R1 \cdot (P_{best} - X) + C2 \cdot R2 \cdot (G_{best} - X)$ 
(Where  $W$  is inertia constant,  $R1$  and  $R2$  are random values.  $C1$  and  $C2$  are constant
values and  $X$  is the particle position)
Update particle position :
 $X_{new} = X_{old} + V_{new}$ 
}
}
Until termination criterion is met.
    
```

at a particular speed towards the best birds. Then, each bird forms its new position and repeats the process until the bird reaches the desired destination (Particle swarm optimization, n.d).

In the PSO algorithm, each solution is a group of birds and each bird is said to be a particle. All particles have a fitness value which is determined by the function to be optimized and each particle has a speed which determines its flight direction and distance and then the particle searches the optimal solution in the solutions space with the current optimal particle.

The PSO algorithm involves the interaction and intelligence in the swarm to learn from their own experience (local search) and from the surrounding particles experience (global search).

The typical procedure of PSO is as follows:

(Zhang et al., 2014) proposed a PSO-based algorithm to solve energy-aware RT scheduling problem on heterogeneous multiprocessors using the DVFS technique. RT tasks are assumed periodic, independent and non-preemptive. Experimental results showed that the PSO-based energy-aware metaheuristic uses 40%–50% less energy than the GA-based and SFLA (Shuffled Frog-Leaping Algorithm)-based algorithms and spends 10% less time than the SFLA-based algorithm in finding the solutions. Besides, it can also find 19% more feasible solutions than the SFLA-based algorithm.

(Awadallah, 2016) addressed the optimization problem of energy-aware RT scheduling on heterogeneous multiprocessor platforms using the DVFS technique. Tasks are supposed periodic, dependent and preemptive. A hybrid approach of PSO variant and Min-Min algorithm is proposed to minimize the overall energy consumption, while respecting tasks deadlines. The hybrid approach proposed in this paper modifies the initialization step in the PSO procedure by assigning priorities for each task and then incorporating a Min-Min solution in the randomly generated population. This approach gives the PSO algorithm a push to start from a good solution and then goes on trying to optimize the solution resulting in the Min-Min solution. Authors stated that the proposed algorithm significantly outperforms related approaches in terms of the number of executed iterations and energy saving.

DISCUSSION

Applying AI to resolve the problem of RT scheduling for ES has gained more attention from researchers. The impetus behind this is because the RT scheduling problem for modern ES is a hard multi-objective problem under a variety of constraints and conventional methods have proved to be relatively unsuccessful to find high quality solutions in a reasonable search time.

It is noted that modern ES are working in a dynamic environment with only a partial or imprecise knowledge. In addition, modern ES are becoming very complex in terms of functionality, data and communication. For these reasons, researchers over the past decade have investigated the applicability of AI methods to the RT scheduling problem for ES.

Some important remarks can be appointed out:

1. Most of existing works target only one class of RT scheduling algorithms (i.e. periodic tasks, hard timing constraints, non-preemptive, etc.)
2. Most of existing works target only one or two objectives (i.e. minimizing the response time and/or energy consumption).
3. Most of existing works make very restricted, sometimes unrealistic assumptions. For instance, a task is subject to, at most, one fault occurrence, the processor can adjust its speed in a continuous range; the number of cores is always greater than the number of tasks, etc.
4. Most of energy-aware RT scheduling works reduce dynamic energy but either neglect static energy or use very abstract and simple static energy models.
5. Most of existing works targeting reliability enhancement consider only transient faults for processors and underestimate some other important faults such as communication and memory faults.

6. Lack of a unified framework for validation and approving of the proposed approaches and algorithms on standard benchmarks.

Undoubtedly, each surveyed AI method has a set of advantages and a set of drawbacks.

The big challenge remains in choosing the most appropriate AI method (s). In practice, authors favor the combination of two or more methods in order to improve the quality of results.

CP seems a powerful mathematical tool sine it can model constraints explicitly and enables the integration of sophisticated search techniques in order to improve the search time. However, CP cannot specify explicitly all things related to RT scheduling for modern ES and weaker for continuous variables (for example in the case of continuous voltages values for a processor). Furthermore, most CP tools can fail to find good solutions when constraints contain a big number of variables and lack of user-feedback and interactivity.

The application of Game theory to resolve the RT scheduling problem for ES is relatively a recent tendency and needs more studies especially when the number of players increases but the most important question is related to considering GT as an optimization technique: GT provides a general rule of logic for wining, but not for the optimization strategy.

The primary use of CA is to simulate complex systems and to conclude some emerging behaviors for non-linear systems. Furthermore, with CA it seems very hard to define in some local rules all the necessary constraints for RT, energy and reliability aspects.

In the context of RT scheduling, MAS can bring a big benefit in particular to model complex interactions in large scale distributed embedded systems including cooperation, negotiation, etc. MAS offer also a good opportunity to model high-level aspects as autonomy, self-organization, planning, etc. however, and despite the academic maturity of MAS, their adoption in the ES industry is still unpopular.

AIS are still a nonstandard method and its application to RT scheduling has not well established yet. Machine learning and notably reinforcement learning and ANN are so important.

However, the major issue with ANN is the fact there is no perfect theory as a guide in the aspect of network structure development and design, so network parameters are adjusted just only on the basis of former researchers designing experience and experiment analysis.

Furthermore, we think that the big issue with machine learning is the huge amount of required resources in terms of memory in order to implement it in embedded devices.

EA and swarm intelligence meta-heuristics are attracting more attention from researchers to solve the multiprocessor scheduling optimization problem, but their validation in the context of RT scheduling for ES is still debatable. Table 1 recapitulates the key strengths and pitfalls of the above presented methods.

In order to alleviate some of the drawbacks of the above-mentioned AI methods with regard to the RT scheduling problem for ES, researchers have resorted to some well-known solutions inspired from other disciplines. Among these solutions, we can mention:

1. Enhance the search time for good solutions by parallelization of GA and other meta-heuristics and apply some relatively new styles of computing such as quantum computing.

Quantum computing (Quantum computing, n.d) is a newly emerging interdisciplinary science of information science and quantum science. In quantum computing, the smallest unit of information storage is the quantum bit (qubit). A qubit can be in the state 1, in the state 0 or in a superposition of both. The state of a qubit can be represented as: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where $|0\rangle$ and $|1\rangle$ represent the values of classical bits 0 and 1 respectively, α and β are complex numbers satisfying $|\alpha|^2 + |\beta|^2 = 1$. $|\alpha|^2$ is the probability where a qubit is in state 0 and $|\beta|^2$ represents the probability where a qubit is in state 1. A quantum register of m qubits can represent 2^m values simultaneously. However, when the 'measure' is taken, the superposition is destroyed and only one of the values becomes available for use. QIGAs

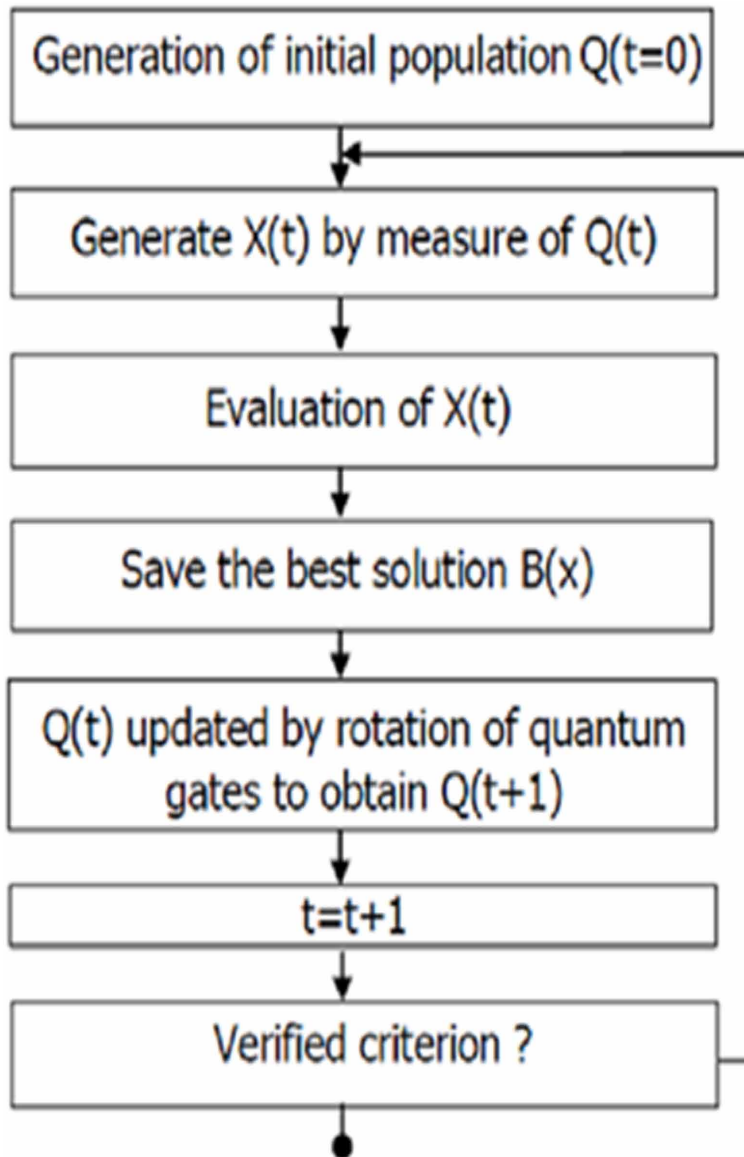
Table 1. Key strengths and pitfalls of AI-based methods used in RT scheduling problem for embedded systems

Method	Key strengths	Key pitfalls
Constraint Programming	-constraints are modeled explicitly. -enables the integration of sophisticated search techniques.	-weak for continuous variables. -can fail to find good solutions when constraints contain a big number of variables. -lack of user-feedback and interactivity.
Game theory	-provides a systematic quantitative approach for deciding the best strategy that will result in maximum gain or minimum loss in competitive situations.	-questionable when the number of players increases. -provides a general rule of logic for winning, but not for the optimization strategy.
Cellular Automata	-simulate complex systems to conclude some emerging behaviors	-it seems very hard to define in some local rules all the necessary constraints for RT, energy and reliability aspects.
Artificial immune systems	-suitable for adaptive RT scheduling modeling.	-a nonstandard method and its application to RT scheduling has not well established yet. -may lead to poor performance.
Artificial Neural networks	-more suitable for complex and intelligent embedded systems. -ability to work with incomplete knowledge.	-there is no perfect theory as a guide in the aspect of network structure development and design, so network parameters are adjusted just only on the basis of former researchers designing experience and experiment analysis. -huge amount of required resources in terms of memory in order to implement it in embedded devices.
Reinforcement learning	-dynamic learning capacity.	-huge amount of required resources in terms of memory in order to implement it in embedded devices.
Multi-Agents Systems	-model complex interactions in large scale distributed embedded systems -offer a good opportunity to model high-level aspects as autonomy, self-organization, planning, etc. -Availability of simulation platforms.	-their adoption in the ES industry is still unpopular. -lack of agent oriented programming languages especially for embedded systems.
Fuzzy logic	-suitable to model uncertainty.	-can lead to a considerable overhead in performance. -requires much experience to select the most appropriate membership function and inference rules.
Evolutionary Algorithms	-Well suitable for nonlinear and multi-modal embedded systems.	-poor performance if the parameters are not well controlled. -can diverge from optimums.
Swarm intelligence	-Well suitable to model adaptive embedded systems. -Simplicity and ease of implementation.	-requires many parameters. -can lead to premature convergence.

(Quantum Inspired Genetic Algorithms) are a combination of GA and quantum computing. They are mainly based on qubits and states superposition of quantum mechanics. A quantum chromosome is simply a string of m qubits that forms a quantum register. Two main operations characterizing QIGA: interference and qubit rotation gates strategy. Interference allows modifying the amplitudes of individuals in order to improve performance. It mainly consists of moving the state of each qubit in the sense of the value of the best solution. This is useful for intensifying the search around the best solution. The rotation of individual's amplitudes is performed by quantum gates. Quantum gates

can also be designed in accordance with the present problem. The population $Q(t)$ is updated with a quantum gates rotation of qubits constituting the individuals.

Figure 19. Structure of quantum inspired genetic algorithm



For instance, the work of (Boutekkouk, 2019b) was interested in RT scheduling optimization with periodic/apperiodic dependent tasks and hard/soft constraints targeting multicores architecture using traditional GA and QIGA. Two strategies called SQIGA (Static Quantum Inspired Genetic Algorithm) based on static preemptive scheduling and DQIGA (Dynamic Quantum Inspired Genetic Algorithm) based on dynamic preemptive scheduling were developed to minimize tasks response

times mean and the number of tasks missing their deadlines under the idea of balancing between processors usage ratios.

Note that the genetic diversity in QIGA is mainly caused by qubit representation so it is not necessary to use genetic operators. Thus, the big evident advantage of QIGA is the reduction in population size and the search time to find optimal solution with comparison to conventional GA (Konar et al., 2018).

2. Deep learning

Deep learning is a subset of machine learning that has networks capable of learning unsupervised from data that is unstructured or unlabeled (Deep Learning Definition, n.d). Deep learning mimics the multilayered human cognition system. Deep-learning architectures comprise of three major layers: the input layer, hidden layers, and the output layer. The number of hidden layers defines the depth of the architecture. The input layer takes the input from the environment or the user and consequently the result of the input layer becomes the input of the first hidden layer. Each hidden layer adds an abstraction to the features and these features then become the input to the next higher layer and this process continues. The output of the final hidden layer serves as the input to the last layer which is an output layer. This output layer provides the result based upon the calculations in the lower layers. Deep learning is generally used where the dataset is very large. (Zhang et al., 2019) proposed an energy-efficient scheduling algorithm (QL-HDS) for periodic tasks based on the deep Q-learning model. Specially, a deep Q-learning model is designed to learn the Q-values of three DVFS technology techniques for different system states by combining a stacked auto-encoder (SAE) which is a typical deep learning model and a Q-learning model. Furthermore, a training strategy is devised to learn the parameters of the deep Q-learning model based on the experience replay scheme. Finally, the performance of the proposed scheme is evaluated by comparison with QL-HDS on different simulation task sets. Results demonstrated that the proposed algorithm could save average 4.2% energy than QL-HDS.

3. Using hybrid methods

In order to get best results, the authors have resorted to combine between a diversity of methods including for instance ANN with GA, ANN with fuzzy logic; GA combined with a swarm intelligence based meta-heuristic or other local search technique, etc.

For instance, (Kashani & Jamei, 2011) used BCO as a local search method in a memetic algorithm to optimize the makespan of the traditional tasks scheduling problem in distributed systems. (Elhossini et al., 2013) combines between a GA, PSO and ANN to generate and evaluate the performance and power consumption of a static scheduler for DSP-based ES. (Mahmood et al., 2017) formulated the RT scheduling for DVS-enabled multiprocessor ES as a combinatorial optimization problem using genetic algorithm hybridized with the stochastic evolution algorithm to allocate and schedule real-time tasks with precedence constraints.

The simulation results show that the proposed algorithm outperforms other algorithms such as GA, PSO, ACO and cuckoo search in terms of solution quality.

In a real-time context, AI activities is known to be a bottleneck with regard to performance so in order to solve this dilemma, we can for instance parallelize them or to implement them as high performance hardware components with multicores (e.g. DSPs) or reconfigurable computing capabilities (FPGA). For instance, Fraunhofer IMS has developed AifES (Artificial Intelligence for Embedded Systems) which is a platform-independent and constantly growing machine learning library in the programming language C, which implies a fully configurable feedforward artificial

neural network (ANN) (Artificial-Intelligence-for-Embedded-Systems, n.d). The implementation of rule-based AI systems in RT embedded systems is also possible (Bouzayen et al., 2017).

5. CONCLUSION AND FUTURE WORK

In this paper, the most relevant taxonomies and AI methods used in the RT scheduling problem for ES were overviewed. RT scheduling is considered as a hard multi-objective optimization problem under a variety of constraints where the necessity to advise new methods that can deal with this problem efficiently. These new methods are inspired by AI domain and include mainly Constraint Programming, Game theory, Machine Learning, fuzzy logic, Artificial Immune Systems, Cellular Automata, Evolutionary Algorithms, Multi Agent Systems and Swarm Intelligence. For each method, the underlying principle and some of the most pertinent works were presented while explaining the idea of each work. However, the validation of such methods is still under research and more experimentations are still required. The survey is ended by a discussion putting the light on some interesting current and future directions including EA and swarm intelligence parallelization, leveraging quantum computing to reduce the search time, use of machine learning and especially the deep and reinforcement learning to deal with big, incomplete and complex data interconnections and the combination between AI methods to get better results. Noting that the implementation of AI notably the EA and deep learning into RT embedded devices is becoming possible with the emerging of new embedded hardware technologies such as the multicore, high performance DSP, the FPGA and the quantum architectures. AI-based optimization is still an open and fresh topic requiring more investigation and an effective collaboration between academy and industry. AI may additionally pose unprecedented challenges (i.e. AI security) due to its ever-growing complexity and the Internet connection, which make AI more vulnerable to threats. It is speculated that future trends will converge towards boosting existing AI methods by adding securing and safety mechanisms while leveraging the progress in hardware technology and high performance computing.

As middle-term perspectives, it is planned to study the different mechanisms used to secure AI methods and estimate their overhead in terms of time and energy consumption in the context of RT scheduling.

REFERENCES

- Abdeyazdan, M., Parsa, S., & Rahmani, A. M. (2013). Task graph pre-scheduling, using Nash equilibrium in game theory. *The Journal of Supercomputing*, 64(1), 177–203. doi:10.1007/s11227-012-0845-z
- Agrawal, P., & Rao, S. (2012). Energy-Aware Scheduling of Distributed Systems Using Cellular automata. *6th Annual IEEE International Systems Conference (IEEE SysCon 2012)*.
- Ahmad, I., & Ranka, S. (2008). Using Game Theory for Scheduling Tasks on Multi-Core Processors for Simultaneous Optimization of Performance and Energy. *IEEE International Symposium on Parallel and Distributed Processing*.
- Ahmeda, M., Fisher, N., Wang, S., & Hettiarachchi, P. (2011). Minimizing peak temperature in embedded real-time systems via thermal-aware periodic resources. *Sustainable Computing: Informatics and Systems*, 1(3), 226–240. doi:10.1016/j.suscom.2011.05.006
- Ant colony optimization algorithms. (n.d.). In *Wikipedia*. https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms
- Artificial Immune Systems. (n.d.). In *Wikipedia*. https://en.wikipedia.org/wiki/Artificial_immune_system
- Artificial Intelligence for Embedded Systems. (n.d.). https://www.ims.fraunhofer.de/en/Business_Units_and_Core_Competencies/Electronic-Assistance-Systems/Technologies/Artificial-Intelligence-for-Embedded-Systems-AIfES.html
- Awadallah, M. H. A. (2016). Hybrid Scheduling Scheme for Real Time Systems. *International Journal of Computers and Technology*, 15(6), 6838–6849. doi:10.24297/ijct.v15i6.1584
- Bambagini, M. (2014). *Energy Saving in Real-Time Embedded Systems* (PhD thesis). Scuola Superiore, Sant'Anna, di Studi Universitari, e di perfezionamento.
- Bambagini, M., Marinouni, M., Aydin, H., & Buttazzo, G. (2016). Energy-Aware Scheduling for Real-Time Systems: A Survey. *ACM Transactions on Embedded Computing Systems*, 15(1), 7:1-7:34.
- Barkahoum, K., & Hamoudi, K. (2019). An Efficient Fault-Tolerant Scheduling Approach with Energy Minimization for Hard Real-Time Embedded Systems. *Cybernetics and Information Technologies*, 19(4), 45–60. doi:10.2478/cait-2019-0035
- Blej, M., & Azizi, M. (2016). Task parameters managing and system accuracy in fuzzy real time scheduling. *International Journal of Engineering and Scientific Research*, 5(7), 61–67.
- Bolaji, A. L., Khader, A. T., Al-Betar, M. A., & Awadallah, M. A. (2013). Artificial bee colony algorithm, its variants and applications: A survey. *Journal of Theoretical and Applied Information Technology*, 47(2), 434–459.
- Boutekkouk, F. (2015, April). A cellular automaton based approach for real time embedded systems scheduling problem resolution. *CSOC2015. 4th Computer Science On-line Conference*. doi:10.1007/978-3-319-18476-0_2
- Boutekkouk, F. (2019a). Embedded systems codesign under artificial intelligence perspective: a review. *International Journal of Ad Hoc and Ubiquitous Computing*, 32(4), 257-269.
- Boutekkouk, F. (2019b). Real time scheduling optimization. *Journal of Information Technology Research*, 12(4), 132–152.
- Boutekkouk, F., & Bounabi, C. (2014). Real Time distributed embedded systems Performance optimization using MultiObjective genetic algorithms. *International Conference on Artificial Intelligence and Information Technology CAIT'2014*.
- Bouzayen, W., Gharsellaoui, H., & Khalgui, M. (2017). New Solutions for AI-Based Adaptive System Under Real-Time and Low-Memory Constraints. *PDPTA'17: The 23rd Int'l Conf on Parallel and Distributed Processing Techniques and Applications*.
- Calvaresi, D., Albanese, G., Marinoni, M., Dubosson, F., Sernani, P., Dragoni, A. F., & Schumacher, M. (2018). Timing Reliability for Local Schedulers in Multi-Agent Systems. *1st International Workshop in Real-Time compliant Multi-Agent Systems @AAMAS 2018*.

- Cellular Automaton. (n.d.). In *Wikipedia*. https://en.wikipedia.org/wiki/Cellular_automaton
- Chandarli, Y. (2014). *Real-time Scheduling for Energy-Harvesting Embedded Systems* (Thèse doctorat). Université Paris, Est.
- Chillet, D., Pillement, S., & Sentieys, O. (2007). A Neural Network Model for Real-Time Scheduling on Heterogeneous SoC Architectures. *Proceedings of International Joint Conference on Neural Networks*. doi:10.1109/IJCNN.2007.4370938
- Chniter, H., Li, Y., Khalgui, M., Koubaa, A., Li, Z., & Jarray, F. (2018). Multi-agent Adaptive Architecture for Flexible Distributed Real-time Systems. *IEEE Access*, 6.
- Classification of embedded systems. (n.d.). <https://www.watelectronics.com/classification-of-embedded-systems/>
- Dahal, K., Hossain, A., Varghese, B., Abraham, A., Xhafa, F., & Daradoumis, A. (2008). Scheduling in Multiprocessor System Using Genetic Algorithms. *27th Computer Information Systems and Industrial Management Applications CISIM '08*.
- Deep Learning Definition. (n.d.). In *Investopedia*. <https://www.investopedia.com/terms/d/deep-learning>
- Ekelin, C. (2004). *An Optimization Framework for Scheduling of Embedded Real-Time Systems* (PhD thesis). Chalmers University of Technology.
- Elhossini, A., Areibi, S., & Dony, R. (2013). Architecture Exploration Based on GA-PSO Optimization, ANN Modeling, and Static Scheduling. *VLSI Design, 2013*, 624369. doi:10.1155/2013/624369
- Elmenreich, W. (2003). Intelligent Methods for Embedded Systems. In *WISES 2003, the First Workshop on Intelligent Solutions in Embedded Systems*. Vienna University of Technology.
- Fan, M., Han, Q., & Yang, X. (2017). Energy minimization for on-line real-time scheduling with reliability awareness. *Journal of Systems and Software*, 127, 168–176. doi:10.1016/j.jss.2017.02.004
- Feng, X., Tang, L., & Leung, H. (2005). A Real Time Scheduler Using Generic Neural Network for Scheduling with Deadlines. *International Conference on Neural Networks and Brain*.
- Ferrandi, F., Lanzi, P. L., Pilato, C., Sciuto, D., & Tumeo, A. (2010). Ant Colony Heuristic for Mapping and Scheduling Tasks and Communications on Heterogeneous Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(6).
- Fuzzy logic. (n.d.). In *Wikipedia*. https://en.wikipedia.org/wiki/Fuzzy_logic
- Game Theory. (n.d.). In *Wikipedia*. https://en.wikipedia.org/wiki/Game_theory
- Ghafarian, T., Deldari, H., & Akbarzadeh, M. (2009). *Multiprocessor Scheduling with Evolving Cellular Automata Based on Ant Colony Optimization*. The 14th International CSI Computer Conference (CSICC'09), Tehran, Iran. doi:10.1109/CSICC.2009.5349618
- Glaubius, R., Tidwell, T., Gill, C., & Smart, W. D. (2010). Real-Time Scheduling via Reinforcement Learning. *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence UAI2010*.
- Han, Q. (2015). *Energy-aware Fault-tolerant Scheduling for Hard Real-time Systems* (PhD thesis). Florida International University, FIU Digital Commons.
- Hladik, P-E., Cambazard, H., Deplanche, A-M., & Jussien, N. (2005). Dynamic Constraint Programming for Solving Hard Real-Time Allocation Problems. *Journal of Network*, 4.
- Housseyni, W., Mosbahi, O., Khalgui, M., & Chetto, M. (2016). *Real-Time Scheduling of Reconfigurable Distributed Embedded Systems with Energy Harvesting Prediction*. IEEE/ACM 20th International Symposium on Distributed Simulation and Real Time Applications, London, UK. doi:10.1109/DS-RT.2016.30
- Huang, J., Olaf Blech, J., & Raabe, A. (2011). *Analysis and Optimization of Fault-Tolerant Task Scheduling on Multiprocessor Embedded Systems*. Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Taipei. doi:10.1145/2039370.2039409

- Jin, H., Chen, L.-Y., Chen, N.-W., & Lei, Y. (2009). *Software Agent Design with Real Time Scheduling for Embedded Systems*. International Conferences on Embedded Software and Systems ICCESS.2009, Hangzhou, Zhejiang, China. doi:10.1109/ICCESS.2009.55
- Kandasamy, N., Hayes, J. P., & Murray, B. T. (2000). Task Scheduling Algorithms for Fault Tolerance in Real-Time Embedded Systems. In *Dependable Network Computing*. Springer. doi:10.1007/978-1-4615-4549-1_18
- Karaboga, D., Gorkemli, B., & Karaboga, C. O. N. (2012). A comprehensive survey: Artificial bee colony (ABC) algorithm and applications. *Artificial Intelligence Review*. Advance online publication. doi:10.1007/s10462-012-9328-0
- Kashani, M. H., & Jamei, M. (2011). Utilizing Bee Colony to Solve Task Scheduling Problem in Distributed Systems. *2011 Third International Conference on Computational Intelligence, Communication Systems and Networks*. doi:10.1109/CICSyN.2011.69
- Kaur, R., & Singh, G. (2019). Design and Analysis of Multi - Heuristic Based Solution for Task Graph Scheduling Problem. *International Journal of Engineering and Advanced Technology*, 8(6), 2673–2681. doi:10.35940/ijeat.F8680.088619
- Kazemi, H., Zahedi, Z. M., & Shokouhifar, M. (2016). Swarm intelligence scheduling of soft real-time tasks in heterogeneous multiprocessor systems. *Electrical & Computer Engineering: An International Journal*, 5(1), 1–13.
- Konar, D., Sharma, K., Sarogi, V., & Bhattacharyya, S. (2018). A Multi-Objective Quantum-Inspired Genetic Algorithm (Mo-QIGA) for Real-Time Tasks Scheduling in Multiprocessor Environment. *ICICT, 2018. Procedia Computer Science*, 131, 591–599. doi:10.1016/j.procs.2018.04.301
- Kulkarni, J. (2015). *The Design of Scheduling Algorithms Using Game Theoretic Ideas* (PhD thesis). Department of Computer Science in the Graduate School of Duke University.
- Laalaoui, Y., & Bouguila, N. (2014). Pre-run-time scheduling in real-time systems: Current researches and Artificial Intelligence perspectives. *Expert Systems with Applications*, 41(5), 2196–2210. doi:10.1016/j.eswa.2013.09.018
- Lay, N. C. (2009). *Enhancing real-time embedded systems development using artificial immune systems* (PhD thesis). The University of York Computer Science.
- Lee, I., Leung, J. Y. T., & Son, S. H. (2007). *Handbook of Real-Time and Embedded Systems*. Chapman and Hall/CRC. doi:10.1201/9781420011746
- MacCarthy, B. L., & Jou, P. (1995). *A case-based expert system for scheduling problems with sequence dependent setup times*. *Transactions on Information and Communications Technologies*, 8.
- Mahmood, A., Khan, S. A., Albaloooshi, F., & Awwad, N. (2017). Energy-Aware Real-Time Task Scheduling in Multiprocessor Systems Using a Hybrid Genetic Algorithm. *Electronics*, 6(40).
- Markov decision process. (n.d.). In *Wikipedia*. https://en.wikipedia.org/wiki/Markov_decision_process
- Mehalaine, R., & Boutekkouk, F. (2016). Fuzzy Energy aware Real Time Scheduling targeting monoprocessor embedded architectures. *CSOC 2016: 5th Computer Science On-line Conference*.
- Monnier, Y., Beauvais, J.-P., & Deplanche, A.-M. (1998). A Genetic Algorithm for Scheduling Tasks in a Real-Time Distributed System. *24th EUROMICRO Conference*, Vasteras, Sweden. doi:10.1109/EURMIC.1998.708092
- Moser, C., Brunelli, D., Thiele, L., & Benini, L. (2006). Real-Time Scheduling with Regenerative Energy. *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06)*. doi:10.1109/ECRTS.2006.23
- Mottaghia, M. H., & Zarandi, H. R. (2014). DFTS: A dynamic fault-tolerant scheduling for real-time tasks in multicore processors. *Microprocessors and Microsystems*, 38(1), 88–97. doi:10.1016/j.micpro.2013.11.013
- Multi-Objective Optimization. (n.d.). In *Wikipedia*. https://en.wikipedia.org/wiki/Multi-objective_optimization
- Munavalli, J. R., Vasudeva Rao, S., Srinivasan, A., & van Merode, G. G. (2020). An intelligent real-time scheduler for out-patient clinics: A multi-agent system model. *Health Informatics Journal*, 26(4), 1–24. doi:10.1177/1460458220905380 PMID:32081068

- Musliner, D. J., Hendler, J. A., & Agrawala, A. K. (1994). *The Challenges of Real-Time AI*. U. Maryland Technical Report CS-TR-3290, UMIACS-TR-94-69.
- Niu, L., & Quan, G. (2006). Energy Minimization for Real-Time Systems With (m; k)-Guarantee. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(7), 717–729.
- Particle swarm optimization. (n.d.). In *Wikipedia*. https://en.wikipedia.org/wiki/Particle_swarm_optimization
- Pradhan, S. R., Sharma, S., Konar, D., & Sharma, K. (2015). A Comparative Study on Dynamic Scheduling of Real-Time Tasks in Multiprocessor System using Genetic Algorithms. *International Journal of Computer Applications*, 120(20).
- Q-learning. (n.d.). In *Wikipedia*. <https://en.wikipedia.org/wiki/Q-learning>
- Quantum computing. (n.d.). In *Wikipedia*. https://en.wikipedia.org/wiki/Quantum_computing
- Rabideau, G., Chien, S., & Ferguson, E. (2015). *Using Automated Scheduling for Mission Analysis and a Case Study Using the Europa Clipper Mission Concept*. In iwps2015, 9th International Workshop on Planning and Scheduling for Space (IWPS), Buenos Aires, Argentina.
- Rehaim, G., Gharsellaoui, H., & Ben Ahmed, S. (2016). *A Neural Networks Based Approach for the Real-Time Scheduling of Reconfigurable Embedded Systems with Minimization of Power Consumption*. ICIS 2016, Okayama, Japan.
- Reinforcement learning. (n.d.). In *Wikipedia*. https://en.wikipedia.org/wiki/Reinforcement_learning
- Saini, G. (2005). *Application of Fuzzy Logic to Real-Time Scheduling*. 14th IEEE-NPSS Real Time Conference, Stockholm, Sweden. doi:10.1109/RTC.2005.1547449
- Samal, A. K., Mall, R., & Tripathy, C. (2014). Fault tolerant scheduling of hard real-time tasks on multiprocessor system using a hybrid genetic algorithm. *Swarm and Evolutionary Computation*, 14, 92–105. doi:10.1016/j.swevo.2013.10.002
- Shrishi, M., Shabir, A., Bong Wan K., Dong Hwan, P. & DoHyeun K. (2019). Hybrid Inference Based Scheduling Mechanism for Efficient Real Time Task and Resource Management in Smart Cars for Safe Driving. *Electronics Journal*, 8(344), 1-23.
- Seredynski, F. (1998). Scheduling Tasks of a parallel program in two-Processor Systems with the use of Cellular Automata. *Journal Future Generation Computer Systems*, 14(5-6), 351–364. doi:10.1016/S0167-739X(98)00039-9
- Swiecicka, A., Seredynski, F., & Zomaya, A. Y. (2006). Multiprocessor Scheduling and Rescheduling with Use of Cellular Automata and Artificial Immune System Support. *IEEE Transactions on Parallel and Distributed Systems*, 17(3), 253–262. doi:10.1109/TPDS.2006.38
- Szymanek, R., Gruian, F., & Kuchcinski, K. (2000). *Digital systems design using constraint logic programming*. *Practical Application of Constraint Logic Programming (PACLP) Conference*.
- Tidwell, T. (2011). *Utility-Aware Scheduling of Stochastic Real-Time Systems* (PhD thesis). Washington University in St. Louis, School of Engineering and Applied Science.
- ul Islam, F.M.M. & Lin, M. (2015). Hybrid DVFS Scheduling for Real-Time Systems Based on Reinforcement Learning. *IEEE Systems Journal*, 11(2), 931-940.
- Umarani Srikanth, G., Uma Maheswari, V., Shanthi, A. P., & Siromoney, A. (2013). Scheduling of Real Time Tasks Using Ant Colony Optimisation. *International Journal of Soft Computing*, 8(1), 50–55.
- Vijayakumar, P., & Aparna, P. (2010). Fuzzy EDF Algorithm for Soft Real Time Systems *International Journal of Computer Communication and Information System (IJCCIS)*, 2(1).
- Wu, G., Xu, Z., Xia, Q., & Ren, J. (2012). An Energy-Aware Multi-Core Scheduler based on Generalized Tit-For-Tat Cooperative Game. *Journal of Computers*, 7(1), 106–115. doi:10.4304/jcp.7.1.106-115
- Zhang, Q., Lin, M., Yang, L. T., Chen, Z., & Li, P. (2019). Energy-Efficient Scheduling for Real-Time Systems Based on Deep Q-Learning Model. *IEEE Transactions on Sustainable Computing*, 4(1), 132–141. doi:10.1109/TSUSC.2017.2743704

Zhang, W., Xie, H., Cao, B., & Cheng, A. M. K. (2014). Energy-Aware Real-Time Task Scheduling for Heterogeneous Multiprocessors with Particle Swarm Optimization Algorithm. *Mathematical Problems in Engineering, 2014*, 2014. doi:10.1155/2014/287475

Zhiyu, H., & Li, L. (2016). A Study on Multi-core Task Scheduling Algorithm based on Artificial Intelligence. *International Journal of Grid and Distributed Computing, 9*(12), 307–320. doi:10.14257/ijgdc.2016.9.12.27

Zhou, J., Yan, J., Wei, T., Chen, M., & Hu, X. S. (2017). *Energy-Adaptive Scheduling of Imprecise Computation Tasks for QoS Optimization in Real-Time MPSoC Systems. Design, Automation & Test in Europe Conference & Exhibition.*

Zomaya, A. Y., Ward, C., & Macey, B. (1999). Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues. *IEEE Transactions on Parallel and Distributed Systems, 10*(8), 795–812. doi:10.1109/71.790598

Fateh Boutekkouk is a senior lecturer in informatics, at the University of Oum El Bouaghi, Algeria. His research interests include design of embedded and intelligent systems, networks on chip, and formal verification. He obtained his doctorate degree from the University of Constantine in 2010.