# Chapter 2
# Independent Batch Scheduling: ETC Matrix Model and Grid Simulator

**Abstract.** This chapter addresses the problem of *Independent Batch Scheduling* in Computational Grids (CGs). The Expected Time to Compute (ETC) matrix model is defined and employed for the specification of the main scheduling objectives, namely makespan and flowtime, in terms of completion times of the grid computational nodes. This chapter ends with a outline of the main concept of the grid simulator dedicated to the batch scheduling. This simulator is used in the experimental analysis presented in the rest of this book.

## 2.1 Introduction

*Independent Batch Scheduling* is a fundamental model of scheduling in grid systems. In this model the tasks are grouped into batches and can be executed independently in a hierarchically structured static or dynamic grid environments. Due to the massive capacity of parallel computation in CGs, this kind of scheduling is very useful in illustrating large amount of realistic scenarios. Real life examples of batch scheduling include: (a) processing of large log data files of online systems (e.g. banking systems, virtual campuses, and health systems), (b) processing of large data sets from scientific experimental simulations (e.g. High Energy Physics and Parameter Sweep Applications), and (c) data mining in bio-informatics applications.

According to the notation introduced in Sec. 1.4.2 an instance of the independent batch grid scheduling problem can be defined as follows:

$$Rm\left[\{b, indep, (stat, dyn), hier\}\right](objectives)) \tag{2.1}$$

where:

- $Rm$ – Graham's notation references that tasks are mapped into (parallel) resources of various speed[1]
- $b$ – designates that the task processing mode is 'batch mode'

[1] In independent grid scheduling it is usually assumed that each task may be assigned just to one machine.

- $indep$ – denotes 'independency' as the task interrelation
- $(sta, dyn)$ – indicates that we will consider both static and dynamics grid scheduling modes
- $hier$ – references that the scheduling objectives are optimized in hierarchical mode
- $objectives$ – denotes the set of the considered scheduling objective functions.

## 2.2 Expected Time to Compute (ETC) Matrix Model

In this section, the following notation for tasks and machines in independent grid scheduling is introduced from this point forward will be used throughout the book:

- $n$ – the number of tasks in a batch;
- $m$ – the number of machines available in the system for the execution of a given batch of tasks;
- $N = \{1,\ldots,n\}$ – the set of tasks' labels;
- $M = \{1,\ldots,m\}$ – the set of machines' labels.

Tasks and machines are characterized by the following parameters:

(a) **Task** $j$:

- $wl_j$ – workload parameter expressed in Millions of Instructions (MI)
- $WL = [wl_1,\ldots,wl_n]$ is a *workload vector* for all tasks in the batch;

(b) **Machine** $i$:

- $cc_i$ – computing capacity parameter expressed in Millions of Instructions Per Second (MIPS) , this parameter is a coordinate of a *computing capacity vector*, which is denoted by $CC = [cc_1,\ldots,cc_m]$ ;
- $ready_i$ – ready time of $i$, which expresses the time needed for the reloading of the machine $i$ after finishing the last assigned task, a *ready times vector* for all machines is denoted by
  $ready\_times = [ready_1,\ldots,ready_m]$ .

Tasks in this model may be considered as monolithic applications or meta-task with no dependencies among the components. The workloads of tasks can be estimated based on specifications provided by the users, or on historical data, or can be obtained from system predictions [64]. The term 'machine' is related to a single or multiprocessor computing unit or even to a local small-area network.

For each pair $(j,i)$ of task-machine labels, the coordinates of $WL$ and $CC$ vectors are used for an approximation of the completion time of the task $j$ on machine $i$. This completion time is denoted by $ETC[j][i]$ and can be calculated in the following way:

$$ETC[j][i] = \frac{wl_j}{cc_i}. \tag{2.2}$$

All $ETC[j][i]$ parameters are defined as the elements of an *ETC* matrix , $ETC = [ETC[j][i]]_{n\times m}$, which is the main structure in ETC model. The elements in the rows of the ETC matrix define the estimated completion times of a given task on different machines, and elements in the column of the matrix are interpreted as approximate times of the completion of different tasks on a given machine.

The ETC matrix model can be characterized by three main parameters:

- heterogeneity of resource;
- heterogeneity of task;
- consistency.

*Heterogeneity of machine* is defined as a variation of the values in rows of the ETC matrix. It is interpreted as a degree of variation of the machine execution times for a given task. *Heterogeneity of task* is defined as a variation of the values in matrix columns. It is interpreted as a degree of variation of the task execution times for a given machine. The averaged values of all tasks and machine heterogeneity parameters define the heterogeneities of tasks and resources in the whole system.

Another feature of the ETC matrix is its consistency . An ETC matrix is *consistent* if for each pair of the machines $i$ and $\hat{i}$ the following condition is satisfied: if the completion time of some task $j$ is shorter on machine $i$ than on machine $\hat{i}$ then all tasks can be executed (and finalized) faster on $i$ than on $\hat{i}$. The inconsistency of the matrix *ETC* means that there no consistency relation among machines. Semi-consistent $ETC$ matrices are inconsistent matrices having a consistent sub-matrix.

There are numerous methods of generating the ETC matrices, which reflect the machine and task heterogeneity. In the range-based method [100] the heterogeneity of a set of completion times is quantified by the range of the values of those times. Two range parameters are defined as task and machine heterogeneities, and an uniform distribution is used for generating the ETC matrix elements.

In a *Coefficient-of-Variation (CVB)* [6] method the ETC matrix is generated by gamma distributions [93]. The key parameters for this method are defined as follows:

- the estimated execution time of all tasks on an 'average' machine in the system, $exec_{ave}$,
- the variance in the execution times of task, $tvar_{tasks}$,
- the variance in the heterogeneity of grid resources, $mvar_{mach}$.

The parameters $exec_j$ and $tvar_{tasks}$ are used for estimating the execution times $ETC[j][i]$ of the tasks on the machine $i$ with the 'average' speed in the systems. The times $ETC[j][i]$ are generated by using the gamma distribution with the shape and scale parameters denoted by $\alpha_t$ and $\beta_t$ respectively. That is:

$$ETC[j][i] = Gamma(\alpha_t, \beta_t), \tag{2.3}$$

where:

$$\alpha_t = \frac{1}{tvar^2_{tasks}} \tag{2.4}$$

$$\beta_t = \frac{exec_{ave}}{\alpha_t} \tag{2.5}$$

A vector of $ETC[j][i]$ parameters ($j \in N$) defines one column (indexed by $i$) of the ETC matrix. Each element of this column is then used for generating one row of the ETC matrix, that is:

$$ETC[j][\hat{i}] = Gamma(\alpha_m, \beta_m[j]), \tag{2.6}$$

where:

$$l\alpha_m = \frac{1}{mvar^2_{mach}} \tag{2.7}$$

$$\beta_t = \frac{ETC[j][i]}{\alpha_m} \tag{2.8}$$

and $\hat{i} \in M$, $\hat{i} \neq i$.

Finally, the Eq.( 2.2) may be used for calculating the ETC matrix. The coordinates of $WL$ and $CC$ vectors are generated by using the Gaussian distributions the parameters of which express the heterogeneities of tasks and resources in the system.

### 2.2.1 Schedule Representation

Schedules in grid computing can be represented by the vectors of machines' or tasks' labels. Two different encoding methods of schedules in grids can be defined in the following way.

**Definition 2.1.** *Let us denote by $\mathscr{S}$ the set of all permutations* ***with repetition*** *of the length n over the set of machine labels M. An element $S \in \mathscr{S}$ is termed a* schedule *and it is encoded by the following vector:*

$$S = [i_1, \ldots, i_n]^T, \tag{2.9}$$

*where $i_j \in M$ denotes the number of the machine on which the task labeled by $j$ is executed.*

This encoding method is called *direct representation* of the schedule.

The $\mathscr{S}$ set can be also defined as the Cartesian product of $n$ copies of the $M$ sets. That is to say:

$$Schedules = \underbrace{M \times \ldots \times M}_{n}. \tag{2.10}$$

The cardinality of $\mathscr{S}$ is $m^n$.

*Remark 2.1.* In some approaches the $\mathscr{S}$ set may be considered as a discrete subset of an $n$-dimensional metric space $\mathbb{R}^n$ with the conventional *Euclidean Metrics* restricted to $\mathscr{S}$. The distance of any two schedules $S^1, S^2 \in Schedules$ is calculated by using the following formula:

$$Dist_e(S^1,S^2) = \sqrt{\sum_{j=1}^{n} (S^1[j] - S^2[j])^2} \tag{2.11}$$

The metrics $Dist_e(S^1,S^2)$ can be further normalized and used for the definition of the *similarity* relation for the schedules (see Chapter 3, Sec. 3.2 and 3.3).

The direct representation of the schedules can be easily transformed into a *permutation-based representation* , which is defined as a vector $u$ of labels of tasks assigned to the machines. For each machine the labels of the tasks assigned to this machine are sorted in ascending order by the completion times of the tasks. Formally, this kind of schedule encoding method can be defined in the following way:

**Definition 2.2.** *Let us denote by $\mathscr{S}_{(1)}$ the set of all permutations* ***without repetitions*** *of the length n over the set of task labels N. A permutation $Sch \in \mathscr{S}_{(1)}$ is called a* permutation-based representation *of a schedule in CG and can be defined by the following vector:*

$$Sch = [Sch_1, \ldots, Sch_n]^T, \tag{2.12}$$

*where $Sch_i \in N$, $i = 1, \ldots, n$. The cardinality of $\mathscr{S}_{(1)}$ is $n!$.*

In this representation some additional information about the numbers of tasks assigned to each machine is required. The total total number of tasks assigned to a machine $i$ is denoted by $\tilde{Sch_i}$ and is interpreted as the $i$-th coordinate of an assignment vector $\widetilde{Sch} = [\widetilde{Sch}_1, \ldots, \widetilde{Sch}_m]^T$, which defines in fact the loads of grid machines.

*Example 2.1.* The following vector $S = [1,2,1,4,3,1,2,4,3,3]^T$ is an example of the schedule for 4 machines and 10 tasks encoded by the direct representation method. The same schedule in the permutation-based representation is as follows:

$(Sch = [1,3,6,2,7,5,9,10,4,8]^T; \widetilde{Sch} = [3,2,3,2]^T)$.

## 2.2.2 Scheduling Criteria

ETC matrix model is very useful for the formal definition of all main scheduling objective functions (see Chapter 1, Sec. 1.4.2 ). The makespan and flowtime may be additionally expressed in terms of the completion times of the machines . A *completion time $C_i$* of the machine $i$ is defined as the sum of the ready time parameters for this machine and a cumulative execution time of all tasks actually assigned to this machine. The completion time of the machine $i$ is denoted by $completion[i]$ and it is calculated in the following way:

$$completion[i] = ready_i + \sum_{j \in Task(i)} ETC[j][i], \tag{2.13}$$

where $Task(i)$ is the set of tasks assigned to the machine $i$.

The $completion[i]$ parameters are the coordinates of the following completion vector:

$$completion = [completion[1], \ldots, completion[m]]^T \tag{2.14}$$

Vector $C$ is used for calculating the makespan $C_{max}$[2] in the following way:

$$C_{max} = \max_{i \in M} completion[i]. \tag{2.15}$$

In terms of ETC matrix model, a flowtime for a machine $i$ can be calculated as a workflow of the sequence of tasks on a given machine $i$, that is to say:

$$F[i] = ready_i + \sum_{j \in Sorted[i]} ETC[j][i] \tag{2.16}$$

where $Sorted[i]$ denotes a set tasks assigned to the machine $i$ sorted in ascending order by the corresponding $ETC$ values.

The cumulative flowtime in the whole system is defined as the sum of $F[i]$ parameters, that is:

$$F = \sum_{i \in M} F[i] \tag{2.17}$$

A comprehensive list of the scheduling criteria defined in terms of completion times and by using the ETC matrix model can be found in [157].

## 2.3 Main Concept of the Grid Simulator: *Sim-G-Batch*

Simulation seems to be the most effective method for a comprehensive analysis of the scheduling algorithms in large-scale distributed dynamic systems, such as grid or cloud environments. It simplifies the study of schedulers performances and avoids the overhead of coordination of the resources, which usually happens in the real-life grid or cloud scenarios. Simulation is also effective in working with difficult and highly parametrized problems. In such cases a considerable number of independent runs is needed to ensure significant statistical results. Using the simulators for the evaluation of grid schedulers is feasible, mainly because of the high complexity of the grid environment.

Using the simulators for the evaluation of the grid schedulers is feasible, mainly because of high complexity of the grid environment. Many simulation packages, useful in the design and analysis of scheduling algorithms in grid systems, have been recently proposed in literature. MicroGrid [135] , ChicSim [121] and GridSim [27] are currently the major projects in grid simulation.

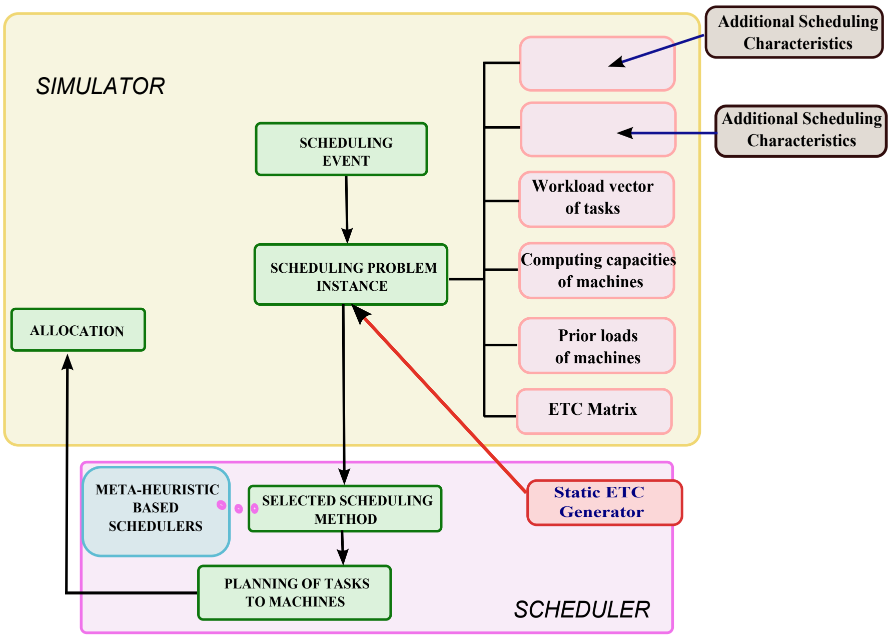
[2] The notation for the scheduling objectives is the same as in Sec. 1.4.2.

This section presents the main concept of a *Sim-G-Batch* grid simulator for independent batch scheduling,as an extension and modification of the *HyperSim-G* framework [163].

## 2.3.1 *Basic Concept of* Sim-G-Batch

*Sim-G-Batch* is based on the discrete event-based model, which facilitates the evaluation of different scheduling heuristics under a variety of scheduling criteria across several grid scenarios. These scenarios are defined by the configuration of security conditions for scheduling and the access to the grid resources, grid size, energy utilization parameters, and system dynamics. The simulator allows the flexible activation or deactivation of all of the scheduling criteria and modules, as well as works with a mixture of meta-heuristic schedulers. The simulation results and traces are graphically represented and may be saved as files of different formats such as spreadsheets or pdf files. The simulator structure allows for an easy association with the external or internal embedded database systems particularly for storing historical executions.

The main concept of the *Sim-G-Batch* simulator is presented in Fig. 2.1.



Additional Scheduling Characteristics
SIMULATOR
Additional Scheduling Characteristics
SCHEDULING EVENT
Workload vector of tasks
SCHEDULING PROBLEM INSTANCE
Computing capacities of machines
ALLOCATION
Prior loads of machines
ETC Matrix
META-HEURISTIC BASED SCHEDULERS
SELECTED SCHEDULING METHOD
Static ETC Generator
PLANNING OF TASKS TO MACHINES
SCHEDULER

**Fig. 2.1** General flowchart of the *Sim-G-Batch* simulator linked to independent batch scheduling in CGs

The *Sim-G-Batch* simulator generates an instance of the scheduling problem by using the following input data:

- the workload vector of tasks ,
- the computing capacity vector of machines,
- the vector of prior loads of machines, and
- the $ETC$ matrix of estimated execution times of tasks on machines.

It is important to address that fact that input data may be extended if additional scheduling criteria are or need to be considered (see Chapter 5 and Chapter 8). The static benchmarks for the small grids are generated by using the external *Static ETC Generator* module.

The users can specify their own scheduling scenario by changing the number of tasks and machines. The capacity of the resources and the workload of tasks are randomly generated by using the Gaussian distribution [101]. It is also assumed that all tasks submitted to the system must be scheduled and all machines in the system can be used.

The structure of the *Sim-G-Batch* application is based on the 2-module *HyperSim-G* architecture and it is composed of *Simulator* and *Scheduler* modules. The main simulation flow can be defined as follows. When a scheduling event is triggered, the *Simulator* creates an instance of the scheduling problem, based on the current task batch and the pool of available machines. The *Simulator* computes an instance of the scheduling and passes it on to the *Scheduler*, which activates a scheduling method specified by the user of the simulator software. The *Scheduler* generates the optimal schedules according to the specified scheduling criteria and sends the schedules back to the *Simulator*. The *Simulator* makes the allocation of the grid resources and re-schedules any tasks assigned to machines which are unavailable in the system.

The *Sim-G-Batch* software was written in C++ for Linux Ubuntu *10.10*. The access to selected modules and resolution methods is available through the Web service, which is the result of work on the WebGridUPC project – a common project with Technical University of Catalonia in Barcelona (UPC Spain) [151][3].

### 2.3.2 Key Parameters

The simulator is highly parameterized in order to illustrate the typical realistic grid scenarios. The main parameters of *Sim-G-Batch* can be interpreted as follows:

- *Init. hosts*: Number of hosts initialized in the grid environment.
- *Max. hosts*: Maximum number of resources in the grid system.
- *Min. hosts*: Minimum number of resources in the grid system.
- *MIPS*: Computing capacity of resource.
- *Add host*: The frequency of activation of the new resources in the system.
- *Delete host*: The frequency of deactivation of the idle or failed resources in the system.

[3] The codes of the benchmars and meta-heuristics are available upon request to Fatos Xhafa (www.lsi.upc.edu/fatos) or Joanna Kołodziej (www.joannakolodziej.org)

- *Total tasks*: Total number of tasks in the batch.
- *Init. tasks*: Initial number of tasks in the system .
- *Workload*: Workload of task.
- *Interarrival*: Frequency of submission of new tasks to the system .
- *Reschedule*: Re-scheduling policy.
- *Number runs*: Number of independent runs of the simulator with the same configuration of the parameters.
- *Scheduling strategy* : The type of the scheduler, maximal execution time of the scheduler (in seconds), and the optimization mode of the scheduling objective functions.

The initial number of machines in the system is defined by the parameter *Init. number of hosts*. The parameters *Max.hosts* and *Min.hosts* specify the range of changes in the number of active hosts during the simulation process[4]. The frequency of appearing and disappearing resources is defined by *Add host* and *Delete host*, according to the constant distributions for the static case, and normal distributions in the dynamic case. The initial number of tasks is denoted by *Init. tasks*parameter, which is constant in the static case. New tasks in the dynamic scheduling can be submitted to the system with the frequency defined by *Interarrival* parameter until the *Total tasks* value is reached. The *Scheduler strategy* parameter defines the type of scheduler, the termination condition for the scheduler and the optimization mode for the scheduling objectives. The setting $GA\_Scheduler(25,s)$ means that the simulator runs the GA-based scheduler for 25 seconds in simultaneous optimization mode[5].

## 2.3.3 Heuristic Schedulers Integrated with the Simulator

The *Sim-G-Batch* simulator allows and facilitates an integration of a mixture of heuristic scheduling algorithms. The simulator architecture enables to design the schedulers as the external dynamic-link libraries (dll files) and store them separated from the simulator main body. The schedulers are plugged in the simulator by using and *Adapter* pattern as it is presented in Fig. 2.2.

The heuristic scheduling methods are usually classified into three main groups, namely (1) calculus-based (greedy algorithms and ad-hoc methods); (2) stochastic (guided and non-guided methods); and (3) enumerative methods (dynamic programming and branch-and-bound algorithm). The heuristic schedulers integrated with *Sim-G-Batch* simulator are divided into three classes, namely *ad hoc*, *local search-based*, *population-based* heuristics. A simple taxonomy of those schedulers is presented in Fig. 2.3.

**Ad-Hoc Methods**

These methods are usually used for single-objective optimization. They are characterized by the low computational cost. Those methods are also very useful in

[4] In the case of dynamic scheduling, they are different from the initial number of hosts.

[5] The parameter $h$ is used for hierarchical optimization mode , e.g. $GA\_Scheduler(25,h)$.

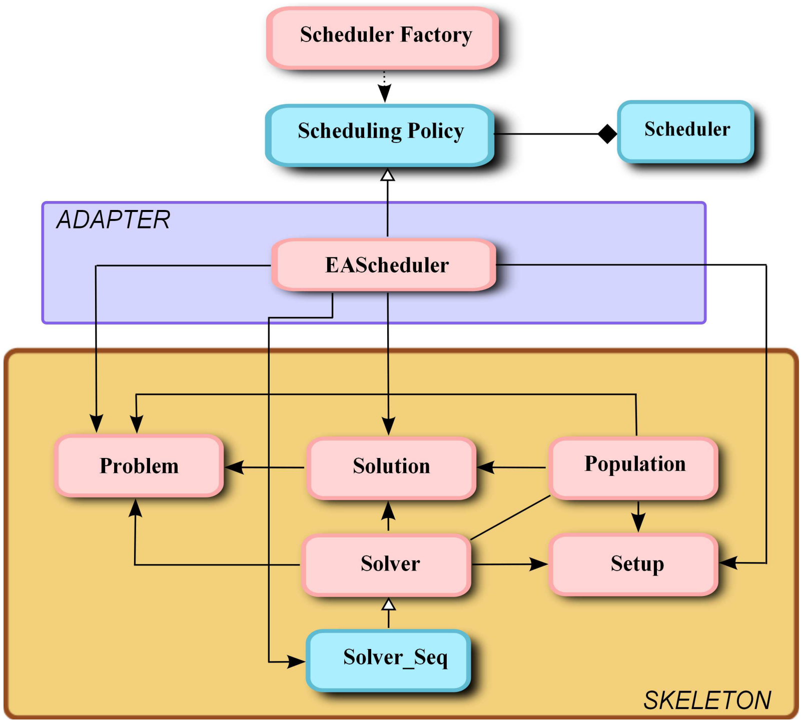


**Fig. 2.2** The simulator adapter pattern used for different evolutionary based grid schedulers

generating the initial solutions for population-based schedulers. The Ad-hoc heuristics could be categorized as immediate mode heuristics and batch mode heuristics.

The *Immediate Mode Heuristics* group includes the following schedulers:

- *Opportunistic Load Balancing (OLB)*, sends a task to the first idle machine without taking into account the machines execution time;
- *Minimum Completion Time (MCT)*, assigns tasks to machines yielding the earliest completion times;
- *Minimum Execution Time (MET)*, assigns tasks to the machine having the smallest execution time for this task.

The *Batch Mode Heuristics* group contains the following methods:

- *Min-Min*: In this method for each task the machine yielding the earliest completion time is computed, then the task with the shortest completion time is selected and mapped to the corresponding machine.
- *Max-Min*: This method differs from the Min-Min as far as the final selection of the task with the latest completion time.
- *Sufferage*: The main idea of this method is to assign a given machine a task, which would “suffer” more if it were assigned to any other machine.

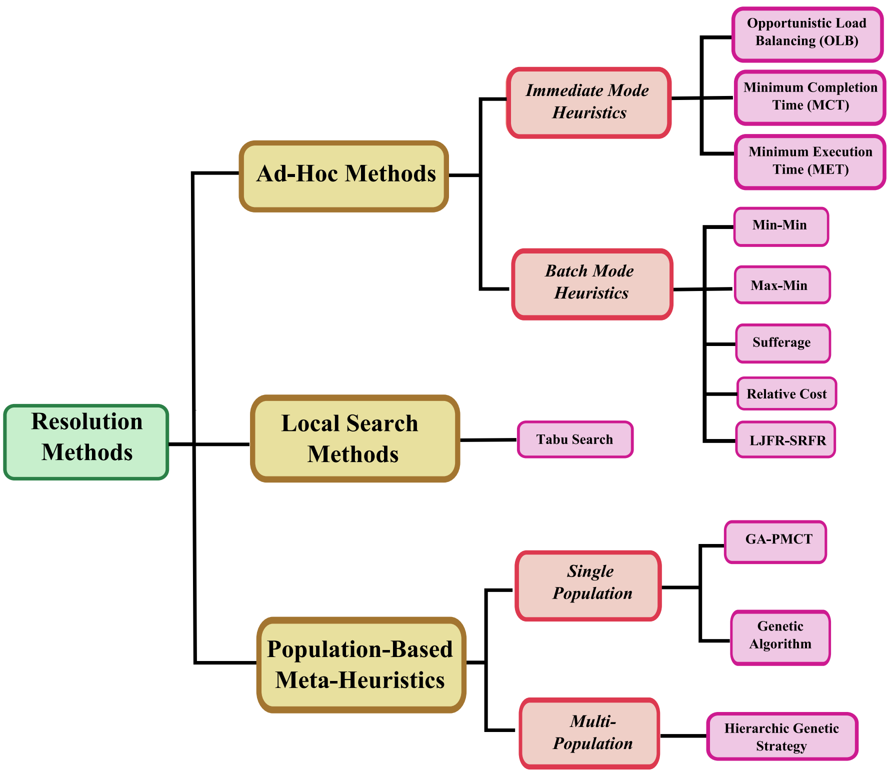


**Fig. 2.3** Taxonomy of heuristic schedulers integrated with *Sim-G-Batch*

- *Relative Cost*: This methods allocates the grid resources according to the load balancing of machines and the execution times of tasks on machines.
- *Longest Job to Fastest Resource - Shortest Job to Fastest Resource (LJFR-SRFR)*: This method tries to simultaneously minimize both makespan and flowtime values: LJFR minimizes makespan and SJFR minimizes flowtime.

### Local Search Methods

Methods from this category explore the optimization domain by constructing a sequence ('path') of partial solutions in optimization space. The most effective local-based grid scheduler is Tabu Search (TS) [162]. TS can be easily hybridized with more sophisticated schedulers (like GAs) to improve their efficiency.

### Population-Based Heuristics

These methods use a population of individuals for the exploration of the solution space. The individuals in the populations represent the partial solutions of the considered problem. The major group in this class is a mixture of the single population

Genetic Algorithms (GAs), proposed by many authors as the effective grid schedulers [160, 161, 120]. Recently, a multi-population hierarchical GA-based scheduler has been defined in [90], [86]. In this method a set of dependent genetic processes is executed simultaneously. Each process creates a branch within the whole strategies tree structure, by using the GA-based scheduler with different settings. The search accuracy in a given branch (expressed as the branch degree parameter) depends on the mutation probability set for the scheduler activated in this branch (the higher mutation problem–the lower accuracy). A generic model of the hierarchical genetic scheduler and its several implementations are presented in Chapters 3-8 in this book.