Pruning Search Spaces of RATA Model for the Job-Shop Scheduling

Farid Arfi[1], Jean-Michel Ilié[2], and Djamel-Eddine Saidouni[1]

[1] Computer Science Department, University of Mentouri,
MISC Laboratory, Constantine, 25000, Algeria
[2] Computer Science Department, Paris 6 University,
LIP6 Laboratory, Paris, 75005, France
arfi_f@hotmail.com, jean-michel.ilie@lip6.fr
saidouni@misc-umc.org

Abstract. In this paper, we propose a pruning method in order to reduce the search space for the job-shop scheduling problem with makespan minimization. In RATA model each trace corresponds to a feasible schedule, so we apply this method to the reachability algorithm of RATA model that explores the space of all possible schedules. We conducted an experimental study over a set of benchmarks. The results show that the proposed method is able to reduce both the space and the time in searching for optimal schedules.

Keywords: Scheduling, reachability analysis, timed model, job-shop.

1 Introduction

The job-shop scheduling problem is a paradigm of optimization and constraint satisfaction problems for distributed systems referenced in many researches over the last years. Traditionally, the optimization criterion is the so-called makespan minimization, which requires to count the time spent to perform the actions of the job-shop. Job-shop problems require the expressions of concurrent and parallel behaviors [1][2][3].

In this paper, we propose to capture job-shop scheduling problems from a very intuitive and compact description model, called Resource Allocation Timed Automata (RATA). This model inherits from the DATA model which introduces true concurrency semantics to deal with concurrent events [4]. Extensions are provided to explicitly represent the resource requirements needed for scheduling analysis. In this model, the parallelism is implicitly expressed from the starting events of actions (i.e. once started, the actions are assumed to behave in parallel until their terminations). This avoids splitting the description of running actions in start and end events, as this is proposed in the Timed Automata models (TA) dedicated to scheduling problems, e.g. [5]. As another interest, the RATA reachability graphs are generally much smaller than the TA ones, e.g. [6]. However, both suffer from the well-known combinatorial explosion problem.

A standard way to attack this explosion problem consists in restraining the execution of actions by focusing on the immediate runs, to study the scheduling

K. Singh, A.K. Awasthi, and R. Mishra (Eds.): QSHINE 2013, LNICST 115, pp. 387–401, 2013.

problems. However in practice, other reduction techniques must be exploited. The main contribution of this paper consists in proposing a set of search space reduction techniques adapted to the RATA models and that can be combined to gain more efficiency. In the last decade, several advances were investigated, from different representations including the TA models. In particular, partial order techniques such as stubborn sets can take advantage of the independency of some executions of actions, in order to reduce the reachability graph to consider [7]. In [8], another partial order technique, namely the sleep sets, is combined with the so-called laziness reduction technique. The aim is avoiding the generation of some lazy runs, featured by configurations from which a bad exploitation of the machines can be detected. In this paper, we propose an improved version of both the stubborn set and laziness reduction techniques, and we propose to combine them since sleep sets are known to only reduce transitions but preserve useless states [9].

Moreover, in another proposition [3], the laziness techniques were replaced by the so-called domination test. This test aims at defining relations between the configurations to be explored within the search space. It is used to suppress the lazy runs and other bad configurations but also is used to replace equivalent sets of configurations by representatives. Observe that the domination test performs comparisons over the set of computed configurations, therefore the laziness reduction technique maintains its interest, since only locally applied from each considered configuration. In this paper, an improvement is proposed to better establish the dominance property between the configurations.

We also use a last reduction technique based on an estimation of the remaining time to be spent in order to achieve the runs from some configuration. Actually, several variants and improvements exist in the literature, e.g. [10].

The remaining of the paper is organized as follows. In section 2, the RATA model is presented and a job-shop use case is described using this model. Section 3 brings out our reachability algorithm dedicated to makespan minimizations. Search space reduction techniques are proposed in Section 4. In Section 5, experimental studies are presented to highlight the efficiency of our approach. This includes some comparisons against an extension of TA [2]. Section 6 presents our conclusion and perspectives.

2 RATA Model

The RATA model re-uses DATA concepts, in particular the non-atomicity of actions is captured by the fact that each transition only corresponds to a start of an action. From state to state, one or several independent actions can be launched, therefore, each state could be associated with a set of launched actions. In the model, each of these launched actions are represented by means of a distinct clock, dynamically created and initialized to 0 at the transition which starts the action. A set of temporal constraints is also associated with each state, expressing the conditions of ends concerning the launched actions in the state. As an example, consider the DATA of Figure *1.a*, modeling a system *S* where two actions, *a* and *b*, able to run concurrently. A distinct clock is assigned to these actions, *x* and *y* respectively. Starting from the initial state s_0, there are two possible transitions: $s_0 \xrightarrow{a,x} s_1$ and $s_0 \xrightarrow{b,y} s_2$. A label *(a,x)* attached

to a transition indicates that the action a has just been launched, and that the clock x will give the time spent since the launching of a. Similarly from the reached states, the following two transitions $s_1 \xrightarrow{b,y} s_3$ and $s_2 \xrightarrow{a,x} s_3$ are possible.

In the initial state s_0, the set of temporal constraints is empty because none of the actions is running in this state. In s_1, $\{x \geq 10\}$ specifies that the action a finishes its execution as soon as x reaches 10. Similarly, s_2 is labeled by $\{y \geq 12\}$. In s_3, the actions a and b can continue their runs in parallel, and each one can finish only if its proper clock reaches a value equal to its duration, so the associated set of temporal constraints is $\{x \geq 10, y \geq 12\}$.

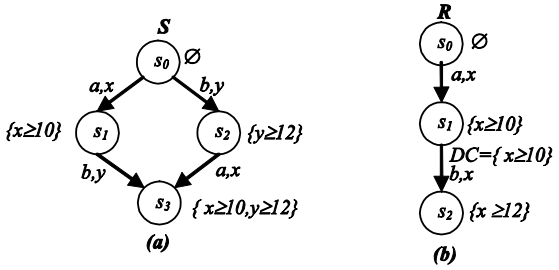

Fig. 1. Behaviors of two systems in terms of DATA

The precedence relation between actions implies to annotate each transition with some additional guard, namely Duration Condition *(DC)*. This guard on a transition expresses that the new launched action is possible, provided the preceding launched ones have been terminated. In the DATA model, this is formally expressed as a subset of the set of temporal constraints attached to the source state of the transition. Consider for instance the system R wherein the action a must be followed by the action b. The behavior of R is shown in Figure *1.b*. Since at most one action can run at a certain point, the same clock x can be assigned to both actions a and b. From the initial state, the unique transition expresses that a can be launched without any duration constraint, hence $DC=\varnothing$ for this transition (not represented in the figure since empty). From the state s_1 where the temporal constraints is $\{x \geq 10\}$ for a, the action b can obviously be run only if the action a finishes its execution. This condition is expressed by the set $DC=\{x \geq 10\}$ attached to the transition which launches the action b. So, b can start at any time within the enabling open interval $x \in [10, +\infty[$.

2.1 Intuition of RATA Model

The RATA model is an extension of the DATA one, assuming an execution platform of (M) machines. An action is executed on a predetermined machine, inducing a duration for its execution. Since a machine cannot be allocated to several actions at the same time, a mutual exclusion mechanism must hold constraining the execution of actions.

Let us consider the system S again, but assume that the execution platform is either P_1 or P_2. The first one contains two machines m_1 and m_2, used for executing the actions a and b respectively, whereas the second contains a single machine m used for

executing any action. The corresponding behaviors for S are represented by the RATA of Figures 2.*a* and 2.*b*, respectively.

It appears that DATA and RATA have the same structure, however the duration of the launched actions are now expressed by using the function τ such that $\tau(a,m)$ yields the duration of any action a executed on a machine m. In addition to *DC*, each transition will be labeled by another guard, namely Availability Condition *(AC)*, expressing the mutual exclusion constraints on shared machines. As for *DC*, The condition *AC* for a transition is a subset of the temporal constraints of the source state, however concern the ones related to the machine of the transition. *AC* is not displayed when empty.

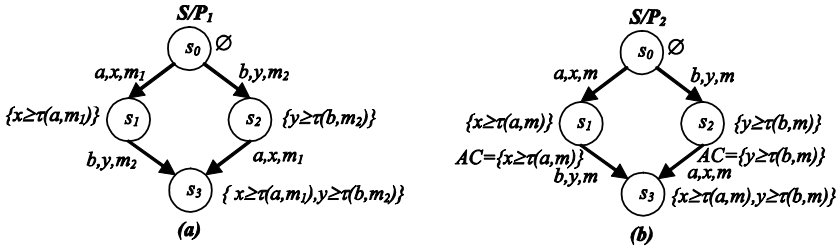

Fig. 2. System S executed on platforms P_1 and P_2

In Figure 2.*a*, all the transitions have an empty *AC* since there is no shared machine on P_1. In Figure 2.*b*, the transition starting from the state s_1 is labeled by $AC=\{x\geq\tau(a,m)\}$. Indeed, the temporal constraint $x\geq\tau(a,m)$ in the source state relates to the shared machine m of the platform P_2. So in the state s_1, the action a is possibly in execution on m and the launching of the action b is enabled only if the temporal constraint $x\geq\tau(a,m)$ holds (i.e. the machine m has finished the execution of a and can start b). Observe that similar reason implies the label of the transition starting from s_2. Observe that the state s_3 of Figure 2.*b* represents different situations of execution where at most one action can be running in s_3, with regard to the set of temporal constraints associated with s_3.

Further, *DC* and *AC* sets are removed from the figures since they can be easily deduced from the clocks and machines used in the labels of transitions, together with the information of the temporal constraints of the source and target states.

2.2 Formalization

Definition 1: Let $H=\{x,y...\}$ be a set of clocks whose values are defined in a time domain R^+ and M a set of machines. The set $\Phi(H)$ of temporal constraints γ over H is defined by the syntax $\gamma::=x\geq t$, where t is a duration value. Durations are expressed by the duration function $\tau:A\times M\rightarrow N$ s.t. $\tau(a,m)$ represents the duration of action a of A (A the set of actions), running on a machine m of M (M the set of machines). Given F a set of constraints, its subset F_x and F^m respectively represent the constraint to the clock x and the different constraints related to the machine m.

A valuation v (of the clocks) of H is a mapping which assigns each clock of H to a value in R^+. The set of all valuations for H is denoted $\Xi(H)$. A valuation $v\in\Xi(H)$

satisfies a temporal constraint $\gamma=(x\geq t)$ with $x\in H$, which is denoted $v|=x\geq t$, iff $v(x)\geq t$. Further, this satisfaction is linearly extended to deal with sets of temporal constraints. W.r.t. $x\in H$, $[x\rightarrow 0]v$ denotes the valuation of H which assigns the value 0 to the clock x and accords with v concerning the clocks of $H\backslash\{x\}$.

Definition 2: A RATA model *RM* is a tuple (S, s_0, H, M, L, T) where:

- S is a finite set of states, $s_0\in S$ is the initial state,
- H and M are respectively the finite set of clocks and the finite set of machines,
- $L: S \rightarrow 2^{\Phi(H)}$ is a mapping that associated with each state s, a set of temporal constraints $F=L(s)$, representing the set of actions possibly in execution in s,
- $T \subseteq S \times A \times H \times M \times S$ is the set of transitions. A transition (s,a,x,m,s') also denoted $s \xrightarrow{a,x,m} s'$ represents a change from the state s to the state s', involving to start the action a on the machine m and define a clock x initialized to 0 to be associated with the action a.

Definition 3: W.r.t. a transition $(s,a,x,m,s')\in$ T, the sets *DC* and *AC* of constraints are defined by: $DC=L(s)\backslash(L(s')\backslash L(s')_x)$ and $AC= L(s)^m$.

The first equation is deduced from $L(s')= (L(s)\backslash DC) \cup \{x\geq\tau(a,m)\}$, where *DC* can be regarded as the precedence constraints defined over the actions of a job. For the first action, *DC* is empty, otherwise it is reduced to a singleton which relates to the preceding action of a within the same job. The cardinality of *AC* can be larger than one, since there may be in s several actions which share the same machine.

The launching of a transition (s, a, x, m, s') from a given valuation v associated with s, is constrained by the following two conditions:

- $v \models DC$. The specification of a system directly corresponds to the properties of precedence over the action executions.
- $v \models AC$. Thus, any action executed in s on a machine m must be completed to allow the firing of a transition which refers to the same machine.

Definition 4: The semantics of a RATA $RM =(S, s_0, H, M, L, T)$ is defined by associating with *RM*, an infinite transition system *SA* on the alphabet $A\cup R^+$. A state of *SA*, also called a configuration, is a pair $<s,v>$ where s is a state of *RM* and v a clock valuation for H. A configuration $<s_0,v_0>$ is initial iff s_0 is initial in *RM* and $\forall x\in H$, $v_0(x)=0$. The two following rules express that two types of transitions can link the *SA* configurations, corresponding to an elapsing of time (*RA*) and an execution of an action of *A* (*RD*), respectively :

$$\frac{d\in \mathbb{R}^+}{\langle s,v\rangle \xrightarrow{d} \langle s,v+d\rangle}(RA) \qquad \frac{(s,a,x,m,s')\in T \quad v \models AC\cup DC}{\langle s,v\rangle \xrightarrow{a} \langle s',[x\mapsto 0]v\rangle}(RD)$$

According to the model semantics, the label a in the RD rule implies the start of an action a and not the whole execution of a. This rule can be applied only in case both sets *DC* and *AC* are satisfied. Otherwise, the time step rule *RA* is applied.

By applying the above rules from the initial configurations, we are able to compute the set of reachable configurations. Further, a run is a path of reachable

configurations, by application of the two former rules. A possible run denoted $(s_0,v_0) \xrightarrow{d} (s_0,v_1) \xrightarrow{a} (s_1,v_2)$, where d represents the time spent in (s_0,v_0) and a the action to be started from (s_0,v_1), first induces that $v_1=v_0+d$, moreover there are a machine m and a clock x such that (s_0,a,x,m,s_1) is a transition of the RATA and $v_2=[x\rightarrow 0]v1$.

2.3 Modeling the Job-Shop with RATA Model

In this paper, the job-shop model is obtained compositionally. First, the sequential semantics for each job is parsed, yielding a RATA model for each job. Then, a standard parallel composition is used to obtain the RATA model of the whole system. The reader can find more details in [11] about the modeling approach and the formal definition of job-shop problem. We restrict our presentation to an example of this problem and its resulting RATA model.

Consider a job-shop system R sharing a set of machines $M=\{m_1,m_2\}$, knowing that each machine performs at most one action at a time, without capacity of preemption. Further, we consider the two following jobs: $j_1=a\prec b$ and $j_2=c$, where $\prec$ represents the precedence relation concerning the execution of actions. The machine allocated to the actions are~:$\mu(a)=\mu(c)=m_1$, $\mu(b)=m_2$, and the duration of the action execution over the shared machines follows~: $\tau(a,m_1)=4$, $\tau(b,m_2)=5$ and $\tau(c,m_1)=3$. The behavior of these jobs is concisely represented by the RATAs of Figures $3.(j_1)$ and $3.(j_2)$ and the resulting composition by Figure $3.(j_1|||j_2)$.

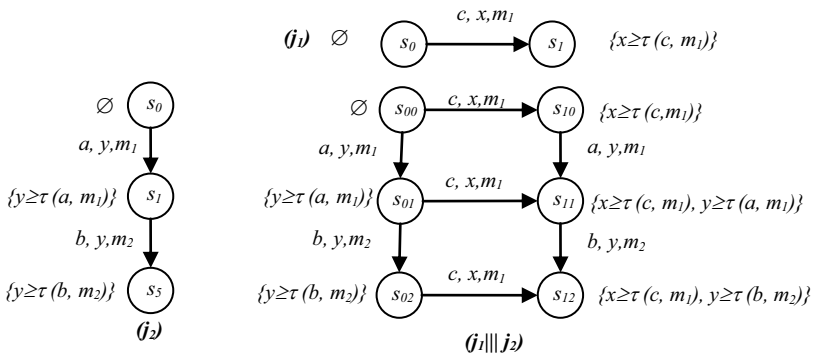

Fig. 3. RATAs of the jobs j_1, j_2 and $j_1||| j_2$.

It is worth noting that the labels of states and transitions in the model $(j_1||| j_2)$ still allow an evaluation of the constraints AC and DC. For instance, the transition from s_{10}, which can violate the mutual exclusion w.r.t. the machine m_1 is prohibited until the action c is considered as terminated in this state (The AC condition specifies that the constraint relative to m_1 in the source state must hold). Observe finally that there are only 6 nodes and 7 edges in the $(j_1||| j_2)$ model, whereas the same specification according to the standard TA approaches involves 14 nodes and 18 arcs [2].

3 Reachability Analysis Algorithm

Starting from the initial configuration of a RATA model, a run is complete if it leads to a final configuration, the potentially running actions of which are considered as terminated. From every *complete run*, say C_R, a schedule can be straightforwardly derived, associating with each action a, the starting time $st(a)$ of the transition labeled by a in C_R. Actually, the length of the schedule coincides with the metric length of C_R. In order to compute the length of a run and the start times of the actions, the considered RATA is augmented by an additional clock to measure the elapsing time spent from the beginning of a run, therefore this clock is never reset to zero. Further, its valuation is denoted t_A. A configuration (s,v) of the RATA model is reachable within the time t_A iff (s,v,t_A) is reachable in the augmented RATA. Our objective is to perform a *makespan minimization* of job-shop problem that is to determine the minimal time schedule where all the actions are completed. The basic algorithm for this problem is presented below.

Algorithm I (A minimal-time Reachability algorithm)
$W \leftarrow \{(s,v0,0)\}$; $P \leftarrow \varnothing$; $Best \leftarrow \infty$
while $(W \neq \varnothing)$ *do*
  $(s,v,t_A) \leftarrow selectRemove(W)$
  if $((s,v,t_A) \notin_d P)$ *then*
    $P \leftarrow P \cup \{(s,v,t_A)\}$
    if $(E(s,v,t_A) < Best)$ *then*
      $S \leftarrow \{(s,v',t_{A'}) \mid (s,v,t_A) \rightarrow (s',v',t_{A'}) \wedge \rightarrow \notin reduce(s)\}$
      if $S = \varnothing$ *then*
        $Best \leftarrow E(s,v,t_A)$
      else
        $W \leftarrow W \cup_d S$
      end if
    end if
  end if
end while
Return Best

The reachability analysis is realized on-the-fly during the building of the RATA model. This avoids an overall construction in case the optimal solution is rapidly discovered. The algorithm operates on the configurations of the reachability graph, the information of which are mainly the location, the clock valuations and an accumulated global time. The list of configurations that must be explored is represented by an ordered set W, called the waiting list. The waiting list W initially contains the initial configuration $(s,v_0,0)$, which in our case corresponds to the state where no job are started, therefore clocks are initialized to zero. The set P is the passed list that means the list of already explored configurations, which is normally empty at the start of the search algorithm. The global variable *Best* holds the time of the best path found so far. (s,v,t_A) represents the configuration that is currently explored in the algorithm. From some configuration (s,v,t_A), S is the sub-set of immediate successors that be reached by a single transition

and which does not belong to the reduced transitions of this configuration. In case the set S is empty, the configuration (s,v,t_A) is a final state, thus can be used to update the *Best* value. The function selectRemove(W) selects and removes from W, a configuration which is minimal according to the ordering $\leq_{ord}$ defined by the tree search strategy. In order to avoid visiting and exploring the same configuration repeatedly, the operations $\not\in_d$ and $\cup_d$ are performed with a respect to some specific property, called dominance property. The function reduce (S) is used to remove those configurations that cannot contribute to a better path or to the optimal run. The test $E(s,v,t_A)<Best$ compares the estimation time value of the current configuration to the best time value found. This avoids the explorations of non-optimal runs if the comparison fails. In a final configuration (s,v,t_A), $E(s,v,t_A)$ yields the exact value of the global (reachability) time for the configuration, here t_A. In the next section, we discuss some decision criteria concerning the function reduce, the dominance property and the global time estimation.

4 Search-Space Reduction Techniques

We propose various search-space reduction techniques that can be used to efficiently prune useless configurations in the reachability graph. They are divided in 2 great classes, the ones that can be realized locally to the considered configuration, then before the generation of its successors, and the second ones which require comparisons between the generated successors and the existing configurations.

4.1 Reduction before the Generation of Configurations

Immediate Runs. Because the time spent on states is left unrestricted by the rule RA in Definition 4, it appears that each qualitative path in a RATA features an infinite number of runs. This can be corrected by only focusing on the restricted notion of immediate runs.

Definition 5: (*Immediate Run*) an immediate run is a run within which each transition is taken as soon as the firing conditions, AC and DC, are satisfied. A non-immediate run is defined as a run containing the fragment: $(s,v) \xrightarrow{t} (s,v+t) \xrightarrow{a} (s',v')$, where the transition taken at (s,v+t) is already enabled at (s,v+t') with t'<t.

Every immediate run represents an immediate schedule. The two schedules S_1 and S_2 in Figure 4 respectively represent a non-immediate schedule and an immediate schedule, with regards to the example problem of the system R explained in Section *2.3*. In the non-immediate schedule S_1 , there are two unnecessary zones of waiting, one for the machine m_2 during the time period [4, 5] and one for m_1 during [10, 11]. In contrast, S_2 brings out a schedule where the waiting times are minimal, thus making the schedule immediate. Therefore in order to find an optimal schedule for a given path, the exploration can be restrained to the immediate runs. This restriction transforms the RATA semantics into a *discrete directed acyclic graph* of configurations, like in Figure 5.

Corollary (*Job-Shop Scheduling and RATA model*): The optimal job-shop scheduling problem can be reduced to the finding of the shortest immediate run within a RATA.

Let us consider the job-shop system in Figure *3.*(j_1||| j_2). Starting from the initial configuration, the immediate runs are directly obtained from the paths of the RATA, by evaluating the satisfaction of the sets *DC* and *AC* in each reached configuration, in order to start the next actions as soon as possible (immediate execution). These evaluations require replacing each occurrence of the function τ by its corresponding value, in order to compare with the values taken by the clocks.

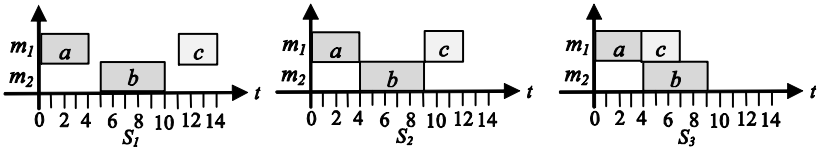

Fig. 4. Non-immediate, immediate but lazy and non-lazy schedules

Figure 5 shows the derivation tree obtained by the immediate runs of the system of Figure 3.(j_1||| j_2). The length of the optimal schedule is 9, which corresponds to the two left immediate runs represented in this figure. Moreover, each configuration is of the form $(s,v(x),v(y),t_A)$, where s represents a reachable state; $v(x)$ and $v(y)$ are respectively the valuations of the clocks x and y used in the configuration; t_A is the value of the additional clock.

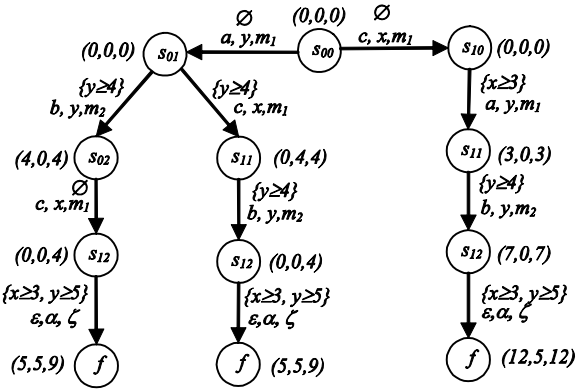

Fig. 5. The immediate runs of the RATA of figure 3.(j_1||| j_2)

The evolution of the elapsing time in a configuration is not represented explicitly but is specified indirectly by the set of constraints attached the transitions issuing the state. Actually, this time depends on the transition to consider. W.r.t. some transition, it can progress from the global time value specified in the state, to a time value featuring that all the constraints attached to the transition are satisfied. For instance, with regard to the transition (s_{01}, s_{02}) and its constraints $\{y \geq 4\}$ the elapsing time in s_{01}

progresses from $t_A = 0$, until the value 4 is reached. Since transitions are immediate, the global time value when reaching s_{02} is equal to 4 again.

Observe also that the last transition corresponds to a hidden action, clock and machine ($\varepsilon \notin A$, $\alpha \notin H$ and $\zeta \notin M$), with an enabled condition used to terminate the execution of all of the actions not yet finished. So, the value of t_A in the final configuration of a run represents the total duration of the run, hence the length (time) of the schedule. The function reduce of Algorithm I mainly performs the generation of the finite number of immediate runs.

Stubborn Set Reduction. A valuable kind of space reduction is based on the checking of "stubborn sets" of transitions that are subset of the transitions which in some state does not influence the other transitions. Such a set can be fired whereas the others can be considered latter, in the next configuration. With regard to the job-shop scheduling problem, the stubborn set technique consists from some configuration to be explored, in selecting the transitions which not only have a minimal launching time but also correspond to the last use of a machine among the jobs. The selected transition can be immediately added to the trace, i.e. advanced w.r.t to the others. When several transitions are candidate to be selected, one is chosen arbitrarily [7].

For instance, consider the second configuration in Figure 5 (i.e. state s_{01}). Both transitions have the same (minimal) launching time $t=4$ and both concern the last use of a machine (the used machines m_1 and m_2 remains unclaimed by any other action during the corresponding action processing time). As a consequence, the function reduce in Algorithm I selects one of them randomly in order to be launched.

We propose here an improvement consisting in weakening the former last use constraints~: we privilege transition, the machine of which remains unclaimed during the processing time of the considered action. As a consequence, the machine can be claimed after this time. Observe that this requires for each job to estimate the earliest starting time of the remaining actions which refers to the same machine. To privilege the transition, all the estimated values must be great than (or equal to) the ending time of the considered action.

Laziness Reduction. Although immediate runs are performed, there could remain lazy schedules. Laziness indicates suboptimal use of the resources, here the machines. Such run can produce suboptimal schedules.

A lazy run of a RATA model *RM* contains a sequence of states and transitions like $(s,v) \ldots \xrightarrow{t} \ldots (s',v') \xrightarrow{a} (s'',v'')$ wherein the transition a is enabled in (s,v), but is taken after a certain delay in (s',v').

In Figure 4, S_2 illustrates an immediate but lazy schedule, s.t. the machine m_1 is free at time 4 then could be used to perform the action *c*. Starting *c* after 3 time units more, introduces a "time hole" which is large enough to be filled with the action *c*. This should make the schedule suboptimal. Exploration of lazy runs can be prevented by a laziness reduction technique. In this special case, the lazy schedule S_2 of Figure 4 is already not expressed in the immediate runs of our model (see Figure 5). This is due to the used semantics, which combines parallel executions and immediate runs. An explicit interleaving of start and end events would make visible such lazy runs, as in the timed automata models [2], thus requires much effort in space and time to remove them.

In the algorithm I, the elimination of the lazy schedules is carried out by the function reduce in case the reduction given by stubborn set technique fails. From the

considered configuration, it applies on every pairs of possible transitions; hence the possible successor configurations that lead to suboptimal solutions are not computed and inserted in the set *S*.

Definition 6. (*Lazyness Reduction*) Assume that two transitions labeled by the actions α and β are enabled in some configuration. Consider that τ(α,m) be the duration of the action α when running on the machine m and let enabl(α) represent the time where the action α is enabled (launched). The transition labeled by β is removed from the successors list S if the following sufficient condition holds~:

$$enabl(\alpha) + \tau(\alpha,m) \leq enabl(\beta) \quad (L)$$

Consider the previous example again, but change the execution machine of the action *c* to m_3 .The lazy schedule *S* of this system represented in Figure 6 is obtained by the following immediate runs, also depicted in the graph (*G*) of the same figure~:

$$(s00,0,0,0) \xrightarrow{a} (s01,0,0,0) \xrightarrow{b} (s02,4,0,4)\ldots$$

At s_{01}, the condition of the laziness reduction holds: $enabl(c)+ \tau(c,m) \leq enabl(b) \Rightarrow 0+3 \leq 4$. The transition labeled by the action *b* can be pruned from s_{01}, so the only considered transition from this state is the one labeled by the action c launched at time 0 (non-lazy schedule).

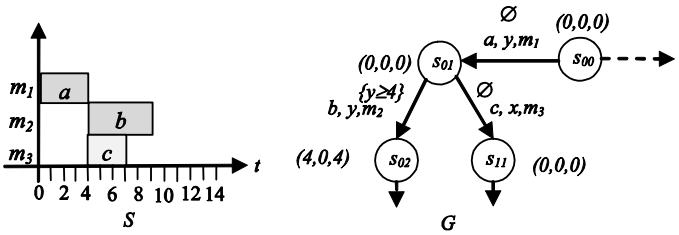

Fig. 6. Lazy schedule S and a sub-graph G of immediate runs

In case the execution time of *c* overpasses the starting of the action *b*, the former laziness technique fails. However, an improvement of Definition 6 is possible.

Laziness Reduction Improvement: Considering the problem specified in definition 6, and assumes *t* is the time interval from the moment where β is started to the moment where an action uses the execution machine dedicated to α. Then, the transition labeled by β is removed from S if~:

$$enabl(\alpha) \leq enabl(\beta) \ \wedge \ enabl(\alpha)+\tau(\alpha,m) \leq enabl(\beta)+t \quad (L').$$

The first condition in *L'* is a direct consequence of the laziness characterization. The second condition suggests that it is possible to remove the action β if the execution machine of α remains unused from the configuration where β was considered. Therefore the execution of β and the other executions enabled in the future of β are preserved in their time after the starting of α.

As for the stubborn set technique, the non-use of a machine during some time period is estimated over the remaining executions of the jobs. If the machine is unclaimed for the remaining action of the jobs, then the value ∞ is assigned to *t*.

4.2 Reduction after the Generation of Configurations

As the reduction technique made before the generation of configurations are based on heuristics, it could remain some reduction to perform that we can detect after the generation of configurations. We propose another set of reduction techniques to be applied on configurations once generated.

Domination Test. This test is used to avoid exploring identical configurations or configurations that are obviously worse than already computed ones. The domination test is based on the following definition:

Definition 7 (D1): Let (s,v,t_A) and $(s,v',t_{A'})$ be any two reachable configurations. We say that (s,v,t_A) dominates $(s,v',t_{A'})$ if $t_A \leq t_A' \wedge v \geq v'$.

The fact that $(v \geq v')$ implies that whatever the enabled transition, it will be launched in (s,v,t_A) before $(s,v',t_{A'})$ or at the same time. Moreover with $(t_A \leq t_{A'})$, we deduce that for every complete run reaching some final configuration $(s,v',t_{A'})$, there is a run reaching (s,v,t_A) which leads to a better solution (i.e. with a lower execution time).

Whenever a new configuration is visited in the graph, we check whether it is dominated by an already computed one, and in this case it is discarded. Moreover, a dominated configuration in the waiting list is replaced by the dominated one. Observe that in the algorithm I, the operations $\cup_d$ and $\in_d$ respectively denote the union and membership relations between configurations, with respect of dominance property.

According to Figure 5, a dominance reduction is possible over the two configurations whose expression is $(s_{12},0,0,4)$ because they have the same global time and the same clocks valuation. Therefore, consider only one, instead of both.

We now propose a finer dominance relation based on a weaker relation between the clocks used in the compared configurations, e.g. (s,v,t_A) and $(s,v',t_{A'})$. For each clock x, we consider its duration denoted $\tau(a,m)$.

Definition 8 (D2) : (s,v,t_A) dominates (s,v',t_A') if :

$t_A \leq t_A' \wedge \forall x \in H, (v(x) + (t_A' - t_A) \geq v'(x)) \vee (v(x) + (t_A' - t_A) \geq \tau(a, m))$.

So, we admit that v could be less than v' for some clocks x in two cases.

- Either the clock difference $v'(x)-v(x)$ is compensated by the value $t = t_{A'} - t_A$ which means that if the same sequence of transitions is fired from (s,v,t_A) and $(s',v',t_{A'})$, reaching (s_r,v_r,t_{Ar}) and $(s_{r'},v_{r'},t_{Ar'})$ respectively, then the above dominance rules globally still hold for the reached configurations. In case where the reached states are final, we have $t_{Ar'} \geq t_{Ar}$.
- In the 2nd term of the or clause, the action a associated with x is terminated within the duration $t_{A'}-t_A$. In this case, the valuation $v(x)$ must be excluded out of the dominance test since it cannot influence the future firing of transitions.

For sake of concision in this paper, the proof is not reported.

In Figure 5, we can now make a dominance reduction between the configurations $(s_{12},0,0,4)$ and $(s_{12},7,0,7)$. Here, each configuration is of the form $(s,v(x),v(y),t_A)$, and the actions associated with the clocks x and y are c and b, with respective durations 3 and 5. Using previous conditions, we can deduce that the first configuration dominates the second.

Reduction Based on the Remaining Time. In the algorithm I, w.r.t. some configuration (s,v,t_A), the estimation $E(s,v,t_A)$ of the global execution time is given by the t_A value complemented by an heuristics on the remaining time to achieve the run. This estimation is compared to the Best known global time value, in order to delete bad configurations and also to search from the most promising configurations first. Clearly, this contributes to reduce the search space.

The main used heuristic in the literature simply computes a remaining time value, under the assumption that there is no conflict between the concerned machines. From some configuration, it is the maximum value between the remaining execution times of the machines, knowing that for a machine, this time corresponds to the sum of the durations of the actions that remain to be executed on the machine.

Recently, an improved heuristic was proposed based on a simple modification of the Jackson's preemptive schedule, which often yields a better estimation, that means greater remaining time values, closer from the exact values. For more details, See [10].

5 Experimental Results

To implement the makespan minimization, we have developed a C++ software tool based on the RATA model. Our tool embeds all the search space reduction techniques discussed in this paper: immediate runs, stubborn set, lazy reduction, domination test and heuristics of the remaining execution time. The searches in the graph use a combination of depth-first and best-first search strategies. The search strategies implemented in the function *selectRemove* of the algorithm I, decide which configuration will be chosen next, from the waiting list. The first criterion is to privilege the configuration which the minimum estimation value of the global execution time. The second criterion is the maximum depth of each configuration, evaluated in the number of actions that have been already executed. Thus, configurations close from a complete run can be privileged. In case of configurations having the same global time estimation and the same depth, the first one is chosen. The computational equipment for the experiments was a Pentium machine with 3 GHz and a Windows7 OS.

Three series of job-shop instances are investigated to demonstrate the performance of our algorithm. The first series (A) consists in randomly generating small instances of jobs having three operations for each job. The number of jobs varies from 2 to 6 in order to investigate the scalability of the proposed reductions. Table 1 shows a comparison between two techniques of reduction considered separately, namely the domination and the laziness techniques. This comparison is given on the derivation tree of immediate runs without considering the other reduction techniques. In Table 1, the left part indexed 1 concerns the classical use of the reduction techniques, whereas the right part indexed 2 corresponds to our proposed improvements. The columns *#laz1* and *#laz2* bring out the performances in terms of number of explored configurations. *T_laz1* and *T_laz2* highlight the processing time values. Similar notations are used for the domination reduction (dom). The number of generated configurations is limited to one million. Comparing the left and part, the improved versions appear to be better and better as the size of the job-shop problem augments.

In the second series (B), a comparison of the size of the RATA against the one of the model obtained by applying the approach of [2], proposed for TA, is given in

Table 1. Performances of the laziness/domination reductions techniques

#j	*RATA*							
	#laz1	*T_laz1*	*#dom1*	*T_dom1*	*#laz2*	*T_laz2*	*#dom2*	*T_dom2*
2	37	0	30	0	22	0	25	0
3	1520	0	305	0	156	*0*	122	*0*
4	62584	1.7	6133	0.4	2198	0.1	1043	0.1
5	/	/	233373	200	191025	*6.4*	10345	*1.2*
6	/	/	/	/	/	/	295084	*27*

Table 2. The column *#ds* informs on the number of discrete states for each model, where *#bf* brings out the performance in terms of number of explored configurations. We restrain our tool reductions to the domination test and best-first strategy, that are used in [2]. As the number of jobs grows, we observe a drastic size reduction by using the RATA approach. The gain rapidly reaches orders of magnitude.

Table 2. Comparison of the RATA model against timed automata

#j	*Timed automata*		*RATA*	
	#ds	*#bf*	*#ds*	*#bf*
2	77	38	25	22
3	629	384	125	105
4	4929	1561	625	306
5	37225	2810	3125	714
6	272125	32423	15625	2520

Table 3. The results for some instances of LA problems

instance	*#alg*	*#time*	*Opt*
LA01	176	0.1	666
LA03	3025	2.1	597
LA05	400	0.0	593
LA06	32460	11.2	926
LA08	17461	4.3	863
LA10	2851	0.4	958
LA11	13327	3.7	1222
LA13	3744	1.5	1150
LA15	/	/	/

In the last series (C), we consider three sets of small and medium benchmarks taken from the well-known OR-library : (1) three instances of size 10*5 (10 jobs and 5 machines), LA01, LA03 and LA05; (2) three instances of size 15*5 (15 jobs and 5 machines), LA06, LA08 and LA10 ; (3) three medium instances of size 20*5 (20 jobs and 5 machines), LA11, LA13 and LA15. All the proposed reduction techniques are used. The results of the experiments are shown in Table 3. The column *#alg* highlights the number of explored configurations and Opt shows the optimal time of the considered problem. As we can see, our algorithm is able to find the optimal for these instances except the last one in a reasonable time (within 5 minutes).

6 Conclusions and Perspectives

Exploiting the RATA model in order to solve optimal job-shop scheduling problems is a novel application of models based maximality-semantics, in addition to

verification purpose [12,13]. Our experiments demonstrate that the search space developed from a RATA model could be drastically much smaller than the ones derived from standard timed automata. The proposed approach covers many reduction techniques that can be used to reduce the search space. In particular, it easily focuses on immediate transitions, moreover, the stubborn set, laziness and domination test techniques are improved and shown to be combined. To a better performance, a usual remaining time based reduction technique is introduced.

Our perspective consists in dealing with larger size problems. We refer to the ideas of [14,1], which argue that one should minimize the length of the schedules without necessarily targeting the optimal solution.

References

1. Subanatarajan, S., Thomas, T., Sebastian, P., Sebastian, E.: Multi-product Batch Scheduling with Intermediate Due Dates Using Priced Timed Automata Models. J. Computers and Chemical Engineering 33, 1661–1676 (2009)
2. Abdeddaim, Y., Asarin, E., Maler, O.: Scheduling with timed automata. J. Theoretical Computer Science 354(2), 272–300 (2006)
3. Abdeddaïm, Y., Maler, O.: Preemptive job-shop scheduling using stopwatch automata. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 113–126. Springer, Heidelberg (2002)
4. Belala, N., Saïdouni, D.E.: Non-Atomicity in Timed Models. In: ACIT 2005, Al-Isra Private University, Jordan (2005)
5. Alur, R., Dill, D.: A Theory of Timed Automata. J. TCS 126, 183–235 (1994)
6. Mokhdad, A., Ilié, J.M., Saidouni, D.E.: Addressing State Space Explosion Problem in Performance Evaluation Using Maximality-based Labeled Stochastic Transition Systems. In: 2nd International Conference on Computer and Software Modeling (ICCSM 2012), India, (2012)
7. Abdeddaim, Y., Niebert, P.: On the use of partial order methods in scheduling. In: Ninth International Conference on Project Management and Scheduling (PMS 2004) (2004)
8. Sebastian, P., Olaf, S., Sebastian, E.: Efficient synthesis of production schedules by optimization of timed automata. J. Control Engineering Practice 14(10), 1183–1197 (2006)
9. Godefroid, P., Wolper, P.: Using partial orders for the efficient verification of deadlock freedom and safety properties. In: Larsen, K.G., Skou, A. (eds.) CAV 1991. LNCS, vol. 575, pp. 332–342. Springer, Heidelberg (1992)
10. Sierra, M.R., Varela, R.: Pruning by dominance in best-first search for the job shop Scheduling problem with total flow time. J. Intelligent Manufacturing 21(1), 111–119 (2010)
11. Arfi, F., Ilié, J.M., Saïdouni, D.E.: Scheduling with RATA model. J. International Journal of Computer Science and Telecommunications (IJCST) 3(10), 14–20 (2012) ISSN:2047-3338
12. Saïdouni, D.E., Benamira, A., Belala, N., Arfi, F.: FOCOVE: Formal Concurrency Verification Environment for Complex Systems. In: Intelligent Systems and Automation (CISA 2008), Annaba, Algeria, vol. 1019 (1), pp. 375–380 (2008)
13. Saïdouni, D.E., Ghenaï, A. : Intégration des Refus Temporaires dans les Graphes de Refus. In : NOTERE 2006, Hermes, Toulouse, France, (2006)
14. Yang, S., Wang, D., Chai, T., Kendall, G.: An improved constraint satisfaction adaptive neural network for job-shop scheduling. J. Journal of Scheduling 13(1), 17–38 (2010)