CHALMERS

Chalmers Publication Library

Scheduling model for systems with complex alternative behaviour

This document has been downloaded from Chalmers Publication Library (CPL). It is the author´s version of a work that was accepted for publication in:

2012 IEEE Conference on Automation Science and Engineering, CASE 2012 (ISSN: 2161-8070)

Citation for the published paper:
Wigström, O. ; Lennartson, B. (2012) "Scheduling model for systems with complex alternative behaviour". 2012 IEEE Conference on Automation Science and Engineering, CASE 2012 pp. 587-593.

http://dx.doi.org/10.1109/CoASE.2012.6386474

Downloaded from: http://publications.lib.chalmers.se/publication/169038

Notice: Changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published source. Please note that access to the published version might require a subscription.

Chalmers Publication Library (CPL) offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all types of publications: articles, dissertations, licentiate theses, masters theses, conference papers, reports etc. Since 2006 it is the official tool for Chalmers official publication statistics. To ensure that Chalmers research results are disseminated as widely as possible, an Open Access Policy has been adopted.
The CPL service is administrated and maintained by Chalmers Library.

(article starts on next page)

Scheduling model for systems with complex alternative behaviour

Oskar Wigstrom and Bengt Lennartson

***Abstract*— In this paper we propose a flexible model for scheduling problems, which allows the modeling of systems with complex alternative behaviour. This model could for example facilitate the step from process planning model to optimization model. We show how automatic constraint generation can be performed for both Constraint Programming and Mixed Integer Linear Programming (MILP) models. Also, for the MILP case, a new formulation for mutual exclusion of resources is proposed. This new formulation works well for proving optimality in systems with multiple capacity resources. Some benchmarks for such job shop scheduling problems as well as systems with a large number of alternatives are also presented.**

I. Introduction

A number of variations of the Job Shop Scheduling Problem (JSSP) have been presented over the years. In its most basic form, the JSSP entails finding the optimal schedule for a plant with m machines and n jobs. Each job j consists of n_j operations, which are to be processed in a specific order in specific machines; these jobs can have both release and due dates. Most often a criterion such as the makespan, weighted average or minimum tardiness is subject to minimization [1].

As computational power and the efficiency of algorithms have increased over the years, so has the posibility for more detailed models. One extension of the JSSP is the Flexible Job Shop Scheduling Problem (FJSSP), where operations may be processed in a set of machines. The FJSSP has been further extended to include amongst others limited capacity buffers [2], positive/negative time-lags [3] and time varying resource availability [4].

The formal model formulation for many of these extensions requires a unique sets of variables for each extension, even though some properties are in essence only linear equalities or inequalities. But most importantly, the alternatives in a FJSSP consist of picking a machine among a set to process an operations [5][6][7]. Because of this, it is not possible to express multiple layers of alternatives with the FJSSP model. Systems where one operation has a number of alternatives, and one or more of these alternatives include yet another layer of options, cannot be modeled by the FJSSP framework.

These nested alternatives in a scheduling problem can in a natural way be described by a tree-like structure. While keeping the model as general as possible, we define a node-tree structure with complementary sets of relations and resources. We show how this new model is well suited for generation of Constraint Programming (CP) as well as Mixed Integer Linear Programming (MILP) constraints. The CP model is good for quickly finding optimal or close to optimal solutions of problems with linear cost functions. For proof of optimality however, its performance degrades quickly in some cases. The MILP constraints can be used either for proof of optimality in linear problems or for problems with a nonlinear cost function, as in [8].

So, to summarize, the contribution of this paper is a new way of modeling nested alternatives, which is shown to be efficient, especially for scheduling of multiple resource systems. Also, corresponding CP and MILP optimization models which can be used for automatic constraint generation are presented, considered and compared. The MILP formulation in this paper is similar to [9][10], but applied to the tree structure formed by the nodes, which allows a more flexible modeling of alternatives. We present two alternatives for modeling resources with multiple capacity. The first models several identical machines with capacity one and treats the allocation as an alternative. The other, which to our knowledge is novel, abstracts the resources' identities.

The paper is structured as follows. Section II contains the scheduling model and definitions for its components. Section III covers the corresponding Constraint Programming model while the Mixed Interger Linear Programming model can be found in Section IV. Some brief comments on computational complexity as well as simulation results are presented in Section V, and finally in Section VI conclusions are drawn along with a brief discussion.

II. Preliminaries

For a set $\mathcal{S}$, $|S|$ denotes cardinality, the number of elements in $\mathcal{S}$. Let $A.y$ refer to element y in a tuple $A = \langle x, y \rangle$. Note that in some cases where certain properties can be inferred, some arguments are removed for notational simplicity. For example the expression $A.x = 2\ \forall A \in \mathcal{A}$ is equivalent to $x = 2\ \forall A \in \mathcal{A}$, as it can easily be inferred in the latter that x is a property of A, and let b_n specify element n in a vector b. Let 2^S be all subsets of S. Also, $\mathbb{R}^n$, $\mathbb{R}^n_+$, $\mathbb{N}^n$, $\mathbb{N}^n_+$ are the n-dimensional spaces of real numbers, positive real numbers, natural numbers and positive natural numbers. Furthermore, if N is a node in an ordered directed tree, the operator $\mathbb{P}[N]$ is the set containing all upwards located nodes as well as N. The operator $\mathbb{S}[N]$ indicates the set of all downwards located nodes, in this case N itself is not included in the set. Finally, a note on mappings: if we define for example $\alpha : A \to B$, $a \in A$ and $b \in B$. Then the function $\alpha(a) = b$ maps a to the actuall element b, not its value.

This work was carried out at the Wingquist Laboratory VINN Excellence Center within the Area of Advance – Production at Chalmers, supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) and the Swedish Research Council. The support is gratefully acknowledged.

O. Wigström, B. Lennartson, Automation Research Group, Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden, oskar.wigstrom@chalmers.se

A. Problem Formulation

As previously mentioned, in FJSSPs, the type of alternative behaviour available is expressed by a set of possible resources which can be used to process an operation. If the alternative represents more than just a choice of resources, or contains even more altarnatives embedded within its child nodes, a more flexible model is necessary. Therefore, we consider a tree-structure of nodes, where each node can either represent an operation or a collection of nodes in which all or only a subset must execute. We define a Scheduling Problem (SP) as the tuple

$$SP = \langle \mathcal{N}, \mathcal{T}, \mathcal{R}, \mathcal{A}, \mathcal{C} \rangle,$$

where $\mathcal{N}$ is a set of top level nodes, $\mathcal{T}$ is a set of temporal orderings that defines relations between start and final times for different nodes, $\mathcal{R}$ is a set of resources, $\mathcal{A}$ is a set of resource allocations and $\mathcal{C}$ is a set of linear constraints. Each node $N \in \mathcal{N}$ forms an ordered directed tree. That is, each node contains zero or more child nodes. A node that has a child is called a parent node, and a node may have at most one parent. The allocations in $\mathcal{A}$ associate nodes with resources.

An assignment to the SP entails setting start and end times to each top level executing node $N \in \mathcal{N}$ as well as the child nodes $\mathbb{S}[N]$. We will define this set of all nodes contained within the tree-structure as $\hat{\mathcal{N}}$, more formally:

$$\hat{\mathcal{N}} = \bigcup_{N \in \mathcal{N}} N \cup \mathbb{S}[N]$$

The assignment to the SP is valid if all temporal orderings in $\mathcal{T}$ as well as resources in $\mathcal{R}$ are satisfied. A temporal ordering is a temporal relation between two nodes, for example node N_1 should have a starting time larger than the finishing time of N_2. Finding a valid assigment equates to solving the satisfiability problem. It could also be the case that some function is to be optimized. Then the solution that among the set of valid assignments that yields an optimal value is the solution to the optimization problem.

Definition 1 (Node)**.** A node is an object that can for example represent a classical JSSP operation or act as a parent for a set of child nodes. The following five-tuple describes a node N

$$N = \langle s, f, d, \mathcal{N}, \nu \rangle,$$

where s and f, both in $\mathbb{R}_+$, are start/end-times, $s \leq f$, $d \in \mathbb{R}_+^2$ is the minimum and maximum duration of the node, $d_1 \leq e - f \leq d_2$, $\mathcal{N}$ is a set possibly containing child nodes, ν expresses the number of nodes in $\mathcal{N}$ that may execute, $0 \leq \nu \leq |\mathcal{N}|$. A node is said to be executing in the time interval $[s...f]$, a node that should not execute due to alternatives, does not require an assignment to neither s nor f.

Remark 1. (i) A fixed duration is a special case where $d_1 = f - s = d_2$. (ii) For an open ended execution time, $d_2 = \infty$. (iii) If $\mathcal{N} = \emptyset$ it can be regarded as an operation.

Definition 2 (Allocation)**.** An allocation A connects a node N to a resource R. Also, it describes during which temporal part, τ, of N the allocation occurs as well as the number of units, $m \in \mathbb{N}$, of R that are allocated. An allocation is given by the tuple

$$A = \langle N, R, \tau, m \rangle$$

The same tuple is also used for deallocation, in which case, m is negative.

Assumption 1. The parameter τ is open for definition. In Section III and IV we define the choice $\tau \in \{pre, post\}$. The values pre and $post$ specify the allocation time instants to be at the start or the end of a node. This implies that an allocation could for example occur at the end of an operation, if $\tau = post$. An extension of this domain is however easy to implement but is omitted for a clearer formulation. An example of an extension is if one would like an allocation could occur a specific number of time units after a node has finished executing.

Definition 3 (Temporal ordering)**.** A temporal ordering relates the start/end times of two nodes. It is defined by

$$T = \langle N_1, N_2, A_t, b_t, k \rangle,$$

where N_1 and N_2 are two nodes subject to the linear constraint formed by $A_t \in \mathbb{R}^{1\times 4}$ and $b_t \in \mathbb{R}$. Also, $k \in \{eq, leq\}$ determines whether the constraint is an equality or inequality . A temporal relation is satisfied if the linear constraint on x by A_t and b_l holds, where x includes the start and end times in N_1 and N_2.

Definition 4 (Allocation pair)**.** This construction will relate two allocations that act on the same resource but with negated magnitude. That is, if a resource is booked, this booking can also be related to an unbooking. The following tuple represents an allocation pair

$$P = \langle A_1, A_2, m \rangle,$$

where $A_1.R = A_2.R$, $A_1.m > 0$ and $m = A_1.m = -A_2.m$. This means that A_1 is an allocation and A_2 a deallocation, both acting on the same resource. The reason for this construct is the MILP model's implementation of mutual exclusion is based the assumption that a resource is booked during time intervals

Assumption 2. It is assumed that it is possible to form pairs for all $A \in SP.\mathcal{A}$. It is possible to formulate CP constraints without this construct, but for the MILP model it is required.

Definition 5 (Resource)**.** A resource R has two parameters,

$$R = \langle c, \mathcal{P} \rangle,$$

where $c \in \mathbb{N}_+$ is the capacity of the resource and $\mathcal{P}$ is a set of allocation pairs acting on the resource. The sum of allocations acting upon R may at no time exceed its capacity c. For a formal description, see the definition of the $cumulative$ constraint in Section III.

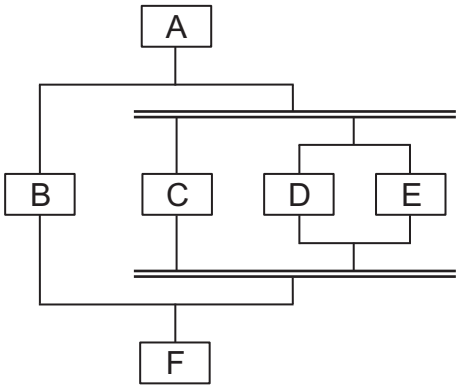

Fig. 1. An example graphical scheduling model

Remark 2. (i) Variable resource availability can be modeled using a job containing operations with predetermined start/end times that allocate the unavailable resource. (ii) negative/positive time-lags as well as release/due dates can be added to $\mathcal{C}$.

B. Modeling example

A simple example, illustrated in Figure 1 will demonstrate how the node structure can be used for modeling alternatives. The graphical representation used is from Sequence Planner Language [11]. A single arc branching into two arcs represents an alternative, only one of the branches should execute. A single arc leading to a double horizontal line represents parallel execution, all arcs leading out from the double line should execute. In this example, there are three posible cases: $A \rightarrow B \rightarrow F$, $A \rightarrow (C + D) \rightarrow F$ and $A \rightarrow (C + E) \rightarrow F$.

Using our modeling formulation, we would add 3 nodes: N_1, N_2 and $N_{D|E}$. First, let $N_{D|E}$ represent the alternative between D and E. That is, set D and E as child nodes of $N_{D|E}$ and as only one may execute, $N_{D|E}.\nu = 1$. Next, the parallell execution is now modeled by adding $N_{D|E}$ and C as child nodes in N_2. As N_2 does not model an alternative, $N_2.\nu = 2$. Last, the alternative between B and the other nodes is implemented by setting N_2 and B as child nodes of N_1, where $N_1.\nu = 1$. The temporal ordering of all the nodes, A before N_1, etc is implemented in $\mathcal{L}$.

III. Constraint Programming Model

Compared to the classical way of solving JSSPs by branch and bound techniques based on mixed integer programming [12], a number of competitive methods now exist. Modern approaches are often based on advanced heuristics and local search algorithms, e.g [13]. CP methods by themselves are relatively competitive while providing a more flexible approach [14]. In combination with heuristics they outperform even the best local search algorithms [15], see also for example [16] [17].

A. Constraints

For the cumulative scheduling constraint, we use the definition from [18]. The constraint is defined as

$$cumulative(s, d, m, c),$$

where s is a list of variables representing starting times, d and m are lists of duration times and booked resource capacity. Finally, c is the resource consumption limit, $c \in \mathbb{N}_+$. The constraint is true given the following condition:

$$\sum_{j|s_j \leq t \leq s_j + d_j - 1} m_j \leq c, \quad \text{for all times } t$$

Since alternative behaviour is included in the model, some nodes may not execute. Because of this, some sort of implementation of optional variables must be present in the CP solver. In [19] a number of methods are mentioned for dealing with optional variables. In commercial optimization software such as IBM ILOG CPLEX Optimization Studio [20] this type of functionality is included. For our model, we define three constrains which we expect have analogues or are implementable in common CP modeling languages. First we define the optional constraint, which applied to a node allows its start and end time to take no value and still fulfil any constraints it is a member of. For example

$$Optional(N),$$

expresses that N is an optional node. Also, we define the implication constraint,

$$N_1 \Rightarrow N_2$$

which expresses that if one node N_1 executes, so must N_2. Or in other words, if s and f in N_1 are present, so should the start and end time for N_2 also be incorporated. Finally, the constraint

$$Alternative(N, \mathcal{N}, k),$$

implies that if N executes, then so should only $k \in \mathbb{N}_+$ of the nodes in the set of nodes $\mathcal{N}$, $0 < k <= |\mathcal{N}|$.

B. Model

The model corresponding to the scheduling problem SP in Section II is now given by the following equations. Note that in some CP languages, s and f can be represented as an interval variable. Reformulating the problem with interval variables should prove rather simple. Observe that in most CP solvers, real valued variables are discretized.

Alternatives: First off, any parent nodes $N_p \in \hat{\mathcal{N}}$, for which not all child nodes $N_c \in N_p.\mathcal{N}$ may execute will have to be flagged as alternatives.

$$Alternative(N, N.\mathcal{N}, N.\nu) \quad \forall\{N : N \in \hat{\mathcal{N}}, N.\nu < |N.\mathcal{N}|\} \quad (1a)$$

This is however not enough, all child nodes as well as any nodes in lower branches $\mathbb{S}[N_p.\mathcal{N}]$ must be set to optional.

$$Optional(N_c) \quad \forall\ \{N_c : N_c \in \mathbb{S}[N_p.\mathcal{N}] \cup N_p.\mathcal{N}\}, \{N_p : N_p \in \hat{\mathcal{N}}, N_p.\nu < |N_p.\mathcal{N}|\} \quad (1b)$$

Also, since the lower branches have been set to optional, we must make sure that any nodes in $N_p.\mathcal{N}$ or even

lower branches which do not act as alternatives, imply the execution of their child nodes.

$$N \Rightarrow N_c \quad \forall \{N_c \in N.\mathcal{N}\}, \{N : N \in \hat{\mathcal{N}}, \\ N \text{ is optional}, 0 < N.\nu = |N.\mathcal{N}|\} \quad (1c)$$

Or in other words, suppose a parent node contains child nodes which all should execute. Also, all these nodes are optional because of an alternative higher up in the tree. Then it must be made sure that if the parent node executes, so do the child nodes.

Execution: The start and end time restrictions on each node are formed by simple linear constraints, which could just as well have been included in $\mathcal{C}$.

$$s + d_1 \leq f \qquad \forall \{N : N \in \hat{\mathcal{N}}\} \qquad (2a)$$
$$f - s \ \leq d_2 \qquad \forall \{N : N \in \hat{\mathcal{N}}\} \qquad (2b)$$

ensures that duration constraints of each node are upheld. If the duration is fixed, this can be replaced with an equality constraint.

Temporal relations: The set of temporal orderings are also in principle a set of linear constraints.

$$A_t y \leq b_t \qquad \forall \{T : T \in \mathcal{T}, k = leq\} \qquad (3a)$$
$$A_t y = b_t \qquad \forall \{T : T \in \mathcal{T}, k = eq\} \qquad (3b)$$

$$y = [N_1.s, N_1.f, N_2.s, N_2.f]^T$$

Resources: Let us define a function $\alpha(A)$ which maps an allocation A to the specific time instance indicated by its N and τ. With our previously defined domain of τ:

$$\alpha(A) = \begin{cases} N.s & \text{if } \tau = pre \\ N.f & \text{if } \tau = post \end{cases}$$

With this, we can model the resource capacity by:

$$cumulative(s, d, n, c), \qquad \forall \{R : R \in \mathcal{R}\} \qquad (4)$$

Here, s, d and m are lists of variables for which, for each $R \in \mathcal{R}$, $P_j : J \to \mathcal{P}$ and $j \in \mathbb{N}$. In other words, there is an index set J for each resource, which maps to the pairs of that resource.

$$s_j = \alpha(P_j.A_1),$$
$$d_j = \alpha(P_j.A_2) - \alpha(P_j.A_1)$$
$$m_j = P_j.m$$
$$\forall \{j : j \in J\}, \{R : R \in \mathcal{R}\}$$

In other words, s, d and m are simply ordered sets with start times, durations and allocation magnitudes for each allocation pair for a resource R.

Linear constraints: Finally, the set of linear constraints in $\mathcal{C}$ should be included. Provided the optional, implication and alternative constraints work as specified, the implementation of these should be trivial.

IV. Mixed Integer Linear Model

The first integer programming models for JSSPs fall into three different categories [21], with the major difference being the modeling of time. The start and end times of operations can be treated as integers [22] or time can be given implicitly by the order of operations [23]. As a third option, the planning period could be discretized and an operation starting in a specific time instance is modeled by a boolean variable [24]. Since CP is very adept at solving scheduling problems with linear constraints, we focus on the modeling formalism in [22], where the integer decision variables, representing start and end times, can just as well be real valued variables. This makes it possible to use the constraints for scheduling problems with nonlinear cost functions as well, cf.[8]. Recent formulations for the FJSSP based on the formalism in [22] can be found in for example [9][10].

The MILP formulation in this paper is similar to these, but applied to the tree structure formed by the nodes, which allows a more flexible modeling of alternatives. Also, the model allows allocations with magnitude larger than one and allocations that can span more than only one operation. Two alternatives for modeling resources with multiple capacity is presented. The first is based on modeling several identical machines with capacity one and treating the allocation as an alternative, while the other abstracts the resources' identities.

A. Model

Alternatives: Define a set of boolean variables B, and a bijective function $\beta : N \to B$. This mapping links each node, which may not be executed because of an alternative, to a boolean representing if the node is to execute. A boolean with value 1 implies execution.

$$\sum_{b \in \beta(N_p.\mathcal{N})} b = N_p.\nu \\ \forall \{N_p : N_p \in \hat{\mathcal{N}}, N_p.i < |N_p.\mathcal{N}|\} \quad (5)$$

These constraints will give the correct behaviour for all parent nodes N_p, where only a subset of its child nodes $N_c \in N_p.\mathcal{N}$ should execute. These booleans will be present in other constraints to nullify constraints where non executing nodes are present.

Execution: The duration constraints are now expressed:

$$s + d_1 \leq f + \sum_{b \in \beta(\mathbb{P}[N])} M(1-b), \quad \forall \{N : N \in \hat{\mathcal{N}}\} \quad (6a)$$
$$f - s \leq d_2 + \sum_{b \in \beta(\mathbb{P}[N])} M(1-b), \quad \forall \{N : N \in \hat{\mathcal{N}}\} \quad (6b)$$

where M is a constant which is sufficiently large to force the constraint to be satisfied (nullified) if $b = 0$. Note that, in the case of a fixed duration operation, the two above constraints can may very well be written differently, similar to the equality case in the temporal relation constraints. Recall the definition of $\mathbb{P}[N]$, the set of all upwards located nodes including N itself. This means that equations (6a,b) will only hold if N and all its upwards located nodes should

execute.

Temporal relations: The temporal orderings, which form temporal relations between pairs of nodes (N_1,N_2), are implemented by

$$A_t y \leq b_t + \sum_{b \in \beta(Q_1)} M(1-b), \quad \forall \{T : T \in \mathcal{T}, k = leq\} \quad (7a)$$

$$\begin{cases} A_t y & = b_t + s, \\ s & \leq \sum_{b \in \beta(Q_1)} M(1-b), \\ \sum_{b \in \beta(Q_1)} M(b-1) & \leq s, \end{cases} \quad \forall \{T : T \in \mathcal{T}, k = eq\} \quad (7bcd)$$

$$Q_1 = \mathbb{P}[N_1] \cup \mathbb{P}[N_2]$$
$$y = [N_1.s, N_1.f, N_2.s, N_2.f]^T$$

In this case, the parent nodes for both N_1 and N_2 as well as the nodes themselfs must be executing for the constraints to be true. This is defined by the set of nodes Q_1. For inequality constraints, (7a) functions in the same way as (6ab). In the equality case however, (7b) expresses the equality constraint itself, while the variable s is either constrained to 0 by (7cd) or set free.

Resources v1: In the first resource model, multi capacity resources are split up into multiple single capacity resources. First, we define a set of boolean vectors C, each element is denoted c. Also, $i \in \mathbb{N}$ is an index representing each single capacity resource, the i:th element in c is denoted c_i. Also define a bijective function $\gamma : \mathcal{P} \to C$. This means that each pair executing in a resource has a corresponding boolean vector which indicates which single capacity resource or resources the pair uses.

$$\sum_{i=0}^{R.c-1} \gamma(P)_i = m \quad \forall \{P : P \in \mathcal{P}\}, \{R \in \mathcal{R}\} \quad (8a)$$

This first equation expresses the number of single capacity resources which are used by the allocation pair. Next the mutual exclusion for each posible combination of pairs in each possible single capacity resource is generated. For each pair, there must also be a boolean describing which one executes first, we define a bijective mapping $\delta : \mathcal{P} \times \mathcal{P} \to D$, where D is a set of boolean variables. If the boolean $\delta(P_1, P_2)$ is true, then this implies that P_1 executes before P_2. Thus, $\delta(P_1, P_2) \neq \delta(P_2, P_1)$.

$$\alpha(P_1.A_2) \leq \alpha(P_2.A_1) + \sum_{b \in \beta(Q_2)} M(1-b) + M(3 - \gamma(P_1)_i - \gamma(P_2)_i - \delta(P_1, P_2))$$
$$\forall \{P_2 : P_2 \in \mathcal{P} \backslash P_1\}, \{P_1 : P_1 \in \mathcal{P}\}, \{i \ : i \in \mathbb{N}_+, i \leq R.c\}, \{R \ : R \in \mathcal{R}\} \quad (8b)$$

$$Q_2 = \bigcup_{i,j \in \{1,2\}} \mathbb{P}[P_i.A_j.N]$$

The boolean set Q_2 includes all booleans related to the nodes included in the two pairs P_1 and P_2. This ensures that all intervals connected to the allocation pair are executed, if not, there is no need for a mutual exclusion. The γ functions ensure that both pairs are actually performed on the same machine i, and the δ funtion models which pair should go first. The definition of $\alpha(A)$ is the same as in the previous section.

Resources v2: This formulation is based on checking the number of executing nodes at the starting time of each node. Use the previously defined δ function and also and define another bijective function $\eta : \mathcal{P} \times \mathcal{P} \to H$, where H is a set of boolean variables. The function $\delta(P_1, P_2)$ maps to a boolean which is one if P_2 is executing at the start time of P_1 and zero if not.

$$\sum_{P_2 \in \mathcal{P} \backslash P_1} \delta(P_1, P_2) \leq R.c - P_1.m + \sum_{b \in \beta(B)} M(1-b) \quad \forall \{P_1 : P_1 \in \mathcal{P}\}, \{R \ : R \in \mathcal{R}\} \quad (9a)$$

This first equations makes sure that the sum of executing pairs at the start of P_1 does not exceed the resource capacity minus the allocation magnitude of P_1. This constraint must hold only if both nodes in P_1 as well as their upwards located nodes are executing, that is B is defined as before.

$$\begin{cases} M(1 + \eta(P_1, P_2) - \delta(P_1, P_2)) \leq \\ \quad \alpha(P_1.A_1) - \alpha(P_2 A_2(x)) + \sum_{b \in \beta(\hat{B})} M(1-b) \\ -M\eta(P_1, P_2) + 1 \leq \\ \quad -\alpha(P_1.A_1) + \alpha(P_2 A_1) + \sum_{b \in \beta(\hat{B})} M(1-b) \end{cases} \quad \forall \{P_2 : P_2 \in \mathcal{P} \backslash P_1\}, \{P_1 : P_1 \in \mathcal{P}\}, \{R \ : R \in \mathcal{R}\} \quad (9bc)$$

These two equations ensure the specified behaviour of D using the set of booleans H.

Linear constraints: The linear constraints in $\mathcal{C}$ are applied in the same style as (7abcd). The only difference is if more nodes than two are included in the constraint then,

$$Q_1 = \bigcup_{N \in \tilde{\mathcal{N}}} \mathbb{P}[N], \quad (10)$$

where $\tilde{\mathcal{N}}$ is the set of nodes subject to a constraint.

V. BENCHMARKS

Three models are considered in this paper: (i) The CP model in (1)-(4), (ii) MILP$_{V1}$ given by (5)-(7) and the first resource model (8) and (iii) MILP$_{V2}$ given by (5)-(7) and the second resource model (9). For these models we present two benchmarks. The first will compare the performance of the three models with regards to multi capacity resources. They are based on the standard JSSP, but with additional resource capacity. The second benchmark is geared towards systems with a large amount of alternatives, where multiple operations in a job may visit the same resource. The execution time for each operation is determined by a uniform random distribution ranging from 1 to 20.

A. Multi Capacity JSSP

To benchmark the models concerning resource capacity, we generated a number of problems. We have considered problem of a size where in most cases, an optimality proof can be found within 1800[s] (half an hour) using at least one of the three models. Each data entry is a results from at least 20 random generated problem instances. The parameters for these were number of jobs N_j, operations in a job N_o, resources N_r, resource capacity C. As we consider the multi-capacity case, the number of operations and resources will be the same, $N_r = N_o$, while number of jobs will be scaled by the capacity, $CN_r = N_j$. The top most part of Table I shows the results for the standard JSSP case, $C = 1$. The middle and bottom part of Table I shows the compiled results for $C = [2, 3]$. For each problem instance there are two rows, the top most indicates the percentage of optimal solutions as well as the mean value of the execution time. The second row shows the same properties but for the optimality proof.

For the single resource capacity case, both MILP methods perform quite well while the CP model struggles with both finding an optimal solution as well as proving it. The MILP$_{V2}$ method seems slightly faster for finding the optimum while MILP$_{V1}$ manages to prove optimality faster. It should be noted that the CP model results in a close to optimal solution in the initial few seconds of the optimization. It is usually the last percentage of optimailty gap that takes most of its time. For some of the largest problem instances (14), none of the algorithms could prove optimality. Therefore we have specified an upper and lower bound for optimal solution. The lower bound conciders the case where none of the algorithms are concidered optimal for the instances where optimality could not be proven. The upper bound is based on the number of times a model achieved the best (or as good as any other) solution.

In the two cases $C = [2, 3]$, CP performs very well, both for finding an optimal solution as well as proving its optimality. For larger instances in the double capacity resource case, we note that MILP$_{V2}$ outperforms CP when it comes to optimality proof. But as in the previous case, CP still outperforms both MILP methods in terms of finding an optimal solution in the shortest time. In both double and tripple resource capacity, the new resource model in MILP$_{V2}$ is more robust when it comes to optimality proof compared to MILP$_{V1}$. For some cases it is slower finding an optimal solution however.

B. Multi level alternatives

To generate challenging examples with several levels of alternative behaviour we did the following. Each sheduling problem will as before consist of N_j jobs and N_r resources with capacity C. The operations N_o will however with probability p_1, include K subnodes. Also with probability p_2, only 1 of the subnodes may execute. Due to space limitations, we will not provide an indepth complexity analysis of multi level alternative system. This benchmark only compares the three methods for two sets of paramters. Note that operations in one job may now visit the same machine more than once. For this benchmark the maxumum time was set to 300 [s].

For the following results, we chose $N_j = N_r = N_o$, $C = 1$, $p_1 = 0.15$, $p_2 = [0.3, 0.5]$ and $K = 3$. This method creates a large number of embedded extra nodes as p_1 is fairly large. A total of 124 problem instances were generated and the resulting problems had a total number of nodes in the range of $[52...126]$. Figure 2 shows a scatter plot and linear regression curves for the computational time of the three models as a function of the total number of nodes. Note that in 33% of the instances, neither MILP$_{V1}$ nor MILP$_{V2}$ could find an optimal solution, these instances were omitted from the figure. It can be seen that the CP model (dashed) is robust for an increased number of alternatives. Also, MILP$_{V1}$ (solid) outperforms MILP$_{V2}$ (dotted) when it comes to alternatives.

VI. DISCUSSION AND CONCLUSION

This paper presents a flexible scheduling which model allows the modeling of systems with complex alternative behaviour. The scheduling model is based on a tree-strucutre

TABLE I
JOB SHOP SCHEDULING RESULTS

Model	MILP$_{V1}$		MILP$_{V2}$		CP	
Single resource capacity						
10.10.10	100%,	7.47s	100%	6.02s	84%,	141s
	100%,	8.76s	100%	7.78s	9%,	5.57s
12.12.12	100%,	172s	100%	97.5s	43%,	380s
	93%,	151s	96%,	200s	4%,	282s
14.14.14	21-47%		21-63%		0-26%	
	20%	657s	5%	143s	0%	
Double resource capacity						
3.6.3	100%,	0.32s	100%,	0.29s	100%,	0.05s
	100%,	0.44s	100%,	0.29s	100%,	0.71s
4.8.4	100%,	0.80s	100%,	2.74s	100%,	0.07s
	93%,	22.6s	100%,	3.27s	89%,	0.09s
5.10.5	100%,	8.0s	100%,	16.4s	100%,	0.37s
	33%,	146s	100%,	56s	56%,	0.42s
Triple resource capacity						
3.9.3	100%,	1.34s	100%,	1.37s	100%,	0.07s
	81%,	23.4s	100%,	4.64s	85%,	0.16s
4.12.4	100%,	39.5s	100%,	17.7s	100%,	0.09s
	69%,	3.45s	95%,	44.1s	95%,	0.178s
5.15.5	57-67%		52-67%		76-100%	
	57%,	348.8s	52%,	236s	76%,	0.37s

of nodes and it is shown how this model can be used to automatically generate both CP and MILP constraints. We also present a new way to model resources using MILP. This new resource model can be used for multicapacity resources without increasing the number of boolean variables with added capacity. We also present some benchmarks for the optimization models for both the multi capacity JSSP case, as well as for systems with many alternatives. Results show that the new resource model finds optimality proofs faster than the standard model for JSSPs, when resources have capacity two or more. The CP model is shown to perform well for systems with high resource capacity. The benchmark on systems with many alternatives show the CP model to be very robust.

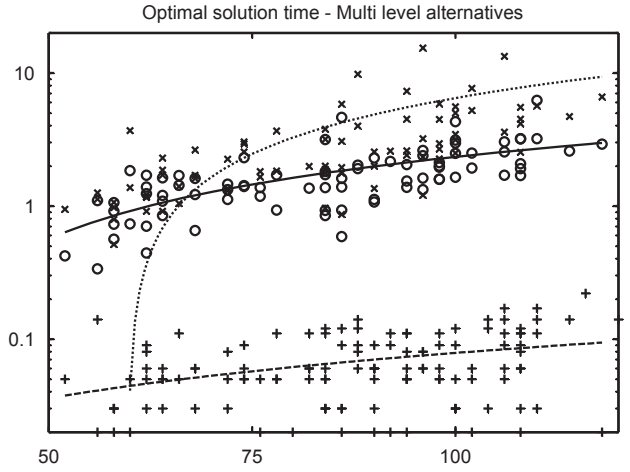

Fig. 2. Computational time for optimal solution [s] as a function of total number of nodes in the problem for 124 instances. The dashed line and $+$-symbols represents the CP model, solid line and circles show MILP$_{V1}$ and finally the dotted line and crosses are MILP$_{V2}$.

REFERENCES

[1] P. M. Pardalos and M. G. C. Resende, editors. *Handbook of Applied Optimization*. Oxford University Press, USA, 1 edition, February 2002.

[2] P. Brucker, S. Heitmann, J.L. Hurink, and T. Nieberg. Job-shop scheduling with limited capacity buffers. *OR Spectrum*, 28(2):151–176, January 2006.

[3] P. Lacomme, N. Tchernev, and M.J. Huguet. Dedicated constraint propagation for job-shop problem with generic time-lags. In *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1 –7, sept. 2011.

[4] Chung-Yee Lee. Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9:395–416, 1996.

[5] A. Baykasoglu. Linguistic-based meta-heuristic optimization model for flexible job shop scheduling. *International Journal of Production Research*, 40(17):4523–4543, 2002.

[6] P Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41:157–183, 1993. 10.1007/BF02023073.

[7] Y. Mati, N. Rezg, and Xiaolan Xie. An integrated greedy heuristic for a flexible job shop scheduling problem. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, volume 4, pages 2534 –2539 vol.4, 2001.

[8] O. Wigström, B. Lennartson, A. Vergnano, and C. Breitholtz. *High level scheduling of energy optimal trajectories*. Accepted for publiction, IEEE Transactions on Automation Science and Engineering 2012, 2012.

[9] P. Fattahi, M.S. Mehrabad, and F. Jolai. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, 18:331–342, 2007.

[10] Cemal ¨Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 34(6):1539 – 1548, 2010.

[11] B. Lennartson, K. Bengtsson, Chengyin Yuan, K. Andersson, M. Fabian, P. Falkman, and K. Akesson. Sequence planning for integrated product, process and automation design. *Automation Science and Engineering, IEEE Transactions on*, 7(4):791 –802, oct. 2010.

[12] S.C. Graves. A review of production scheduling. *Operations Research*, 29(4), 1981.

[13] N. Eugeniusz and S. Czeslaw. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8:145–159, 2005.

[14] Jianyang Zhou. A constraint program for solving the job-shop problem. In *In Second International Conference on Principles and Practice of Constraint Programming (CP'96*, page 150. Springer Verlag, 1996.

[15] D. Grimes, E. Hebrard, and A. Malapert. Closing the open shop: Contradicting conventional wisdom. In *CP'09*, pages 400–408, 2009.

[16] J-Pl Watson and C.J Beck. A hybrid constraint programming / local search approach to the job-shop scheduling problem. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 5015:263–277, 2008.

[17] C. Gueret N. Jussien A. Langevin L.-M. Rousseau A. Malapert, H. Cambazard. An optimal constraint programming approach to the open-shop problem. Technical report, 2009.

[18] A. Aggoun and N. Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57 – 73, 1993.

[19] Petr Vilím, Roman Barták, and Ondřej Čepek. Unary Resource Constraint with Optional Activities. pages 62–76. 2004.

[20] IBM Corp. *IBM ILOG CPLEX Optimization Studio V12.2*, 2010.

[21] Thörnblad K. *On the Optimization of Schedules of a Multitask Production Cell*. Licentiate thesis, Chalmers University of Technology, 2011.

[22] Alan S. Manne. On the job-shop scheduling problem. *Operations Research*, 8(2):pp. 219–223, 1960.

[23] H.M. Wagner. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140, 1959.

[24] E.H. Bowman. The schedule-sequencing problem. *Operations Research*, 7(5):pp. 621–624, 1959.