TWO PHASE TECHNIQUE FOR ASSEMBLY LINE BALANCING

Jüri Vain, Ingmar Randvee, Tiit Riismaa

Tallinn Technical University, Institute of Cybernetics
Akadeemia tee 21, 12618 Tallinn, Estonia

Abstract: A two-phase technique for solving flexible assembly line balancing problems is proposed. In the first phase a global solution is found to the task assignment problem using known algorithmic branch-and-bound techniques. In the second phase the workstations with critical workload are selected and the workstation time is re-calculated/reduced using task models of finer granularity. The workstation models in the second stage are represented as parallel compositions of timed automata to which the parametric model checking technique can be efficiently applied. The method combines the advantages of coarse level line balancing algorithms and fine grain model checking. The modeling rules of second stage are defined which guide the model construction and property specification for estimating the workstation load and parameters in the presence of specific operational and timing constraints. *Copyright © 2005 IFAC*

Keywords: assembly lines, structural optimization, finite automata, models, verification

1. INTRODUCTION

The line balancing problem consists of repetitive distributing of tasks among workstations while optimizing some criteria such as cost, productivity, reliability, maintainability, etc., subject to the previously defined and currently added constraints. An overview and extensive analysis of different line balancing problem settings and methods are given in (Scholl, 1999; Hopp and Spearman, 1996; Scholl and Klein, 1997). One intensively studied group of methods (Scholl, 1999; Hoffmann, 1992) is aimed at solving simple assembly line balancing problems (SALBP) for real size industrial assembly lines. SALBP is stated as follows: minimise the balance delay time provided cycle time, task times and task precedence graph are given. For example, SALOME technique (Scholl and Klein, 1997) can optimize the structure of an assembly line with several hundreds of tasks and provide a minimal set of workstations (WS) together with the tasks assigned to each WS. Line balancing problems, where models and constraints of finer granularity have to be taken into account, are very hard to solve with given algorithms.

On the other hand model checking (MC), has shown to be a promising method for analysis of systems with irregular, timing and other quantitative constraints (Clarke, et al.,1999; Lindahl, et al., 1998). As any enumerative method, MC may be inefficient in case the model includes a large number of parallel components. Feasible results can be achieved in cases where some tens of tasks and few parallel machines are incorporated in the WS (Vain and Küttner, 2001; Vain, et al., 2002).

The aim of this paper is to show the advantages of integrating the traditional branch-and–bound methods with MC. A two-phase approach for solving assembly line balancing problem is suggested. In the first phase the line balancing problem is solved on the coarse-grain model which is given in terms of line tasks assuming that all WS-s are functionally uniform and the precedence relation between tasks is given. In the second phase, the set of task models of operation level granularity with finer performance and synchronization constraints is studied. To obtain more accurate estimate of WS time the behavioral, cost, reliability and other constraints may be included. The goal of detailed analysis is to detect if the cycle time of the assembly line may be reduced, preserving the same assignment of tasks prescribed by the first phase solution.

The paper is organized as follows. In section 2 we give a general description of the two-phase approach to line balancing problem. Section 3 is concerned with construction of detailed WS models using timed automata. In section 4 detailed analysis of WS time using MC is considered. The last section illustrates the modeling approach of Section 3 with an example.

2. DESCRIPTION OF THE APPROACH

The proposed approach to the control of flexible assembly line balancing is based on repetitive search of best distribution of tasks by solving the task assignment problem for current time step in two phases:

- Phase 1: Solve the SALBP using coarse grain methods, e.g., such as SALOME (Scholl and Klein, 1997), and select the critical WS-s, i.e. the WS with highest workload for every optimal assignment of tasks**.**
- Phase 2: Analyze the critical WS (i.e., construct the detailed model of each critical WS, use parametric model checking to get enhanced estimates of the WS time), and adjust the cycle time of given assembly line (Vain, et al., 2002).

There are three possible outcomes of a current search step:

1. The solution of phase 1 is optimal, i.e., at given task times detailed model does not provide any reduction of the critical WS time and thus the cycle time cannot be reduced.
2. The critical WS time estimate calculated in phase 2 is less than the rough estimate calculated in phase 1. That allows to reduce the cycle time.
3. The critical WS time estimate calculated in phase 2 exceeds that of phase 1, i.e., the task assignment found in phase 1 is infeasible because of violating the cycle time requirement.

In the first case the procedure terminates.

In the second case: if after adjustment the critical WS time it turns to be less than other WS times then the phase 2 has to be repeated with those WS-s. If not, the cycle time can be reduced to adjusted critical WS time and the procedure stops.

In the third case the task times have to be corrected, and both phases repeated with new task time estimates.

It is obvious that if the task time estimates are robust enough then the procedure converges after repeating the phase 2 maximally n times (n – the number of workstations).

Technically the decision procedure of phase 1 is implemented using a bi-directional branch and bound procedure SALOME under general assumptions (Scholl and Klein, 1996). The solution of SALBP consists of minimal number of WS-s, assignment of tasks to WS, minimal balance delay time, and minimal cycle time (for given minimal number of WS-s).

The MC problem of phase 2 is stated as follows: check if the WS time is less or equal to the given cycle time, provided the task assignments and refined constraints on task execution (including capabilities of WS machines) are given. Possible reduction of cycle time is based on more accurate WS model and its specialised/rational structure. The used model of critical WS is based on the precedence graph of operations, operation times, and synchronization constraints between concurrent tasks. An operation represents a subtask or its component down to the elementary activity of a processing machine. For instance, the WS operations can be performed by more than one machine (WS can handle more than one operation at a time), the processing rate may depend on the number of tasks assigned to the station, splitting of tasks may be allowed (in the case of partially ordered tasks), etc. Also, the used models of assembly lines are deterministic in the sense that processing times are supposed to be fixed. In fact, introduction of non-determinism into low level automata models is trivial and thus the approach can be extended to interval parameter valuation functions.

3. CONSTRUCTING THE TIMED AUTOMATA MODELS

The construction of timed automata which represent refined models of workstations is a most laborious step of the phase 2. In this section the model templates are defined that facilitate the modeling process. The model checking is supported by UPPAAL MC tool (Larsen, et al., 1997). The usage of a timed modal logic-based property specification language TCTL is described in section 4.

The WS model is given in terms of machine operations. The set of WS operations is performed by WS machines. A WS machine must manage similar operations of several tasks. Involved configuration constraints, detailed assumptions about operation times, their timing, ordering and cost are taken into consideration. The fragment of precedence graph $\boldsymbol{G}_i \subseteq \boldsymbol{G}$ of tasks that are allocated to a workstation ws_i is decomposed into a set of synchronized sequential processes $\boldsymbol{P}^i = \|_j \boldsymbol{P}^i_j$. The constraints coming from the machine configuration of given workstation are represented by parallel composition of machine models $\boldsymbol{M}^i = \|_k \boldsymbol{M}^i_k$ and operation level scheduling constraints (if any) are encoded in planner automaton $\boldsymbol{A}^i$. The workstation model to be analyzed by model checking is then $\mathcal{M}^i = \boldsymbol{P}^i \parallel \boldsymbol{M}^i \parallel \boldsymbol{A}^i$ and the operation-level workstation model constructing process can be described by individual components of $\mathcal{M}^i$.

3.1. Timed automata

The models $\mathcal{M}^i$ belong to the class of timed transition systems that are syntactically described by networks of timed automata (Alur and Dill, 1994). A timed automaton is a finite state automaton extended with a finite collection of real-valued clocks C ranged over x, y etc. Let Act be a finite set of actions and $\mathcal{B}(C)$ the clock constraints that can be an atomic constraint of the form: $x \sim n$ or $x - y \sim n$ for $x, y \in C$, $\sim \in \{\leq, \geq, <, >\}$ ($n \in \mathbf{N}$) or a conjunction of such formulas.

Definition. A timed automaton (TA) A over actions Act, atomic proposition Θ and clocks C is a tuple $\langle N, l_0, E, V\rangle$. N is a finite set of nodes (control nodes), l_0 is the initial node, $E \subseteq N \times \mathcal{B}(C) \times Act \times 2^C \times N$ corresponds to the set of edges, and V: $N \rightarrow 2^{\Theta}$ is a proposition assignment function. In the case $\langle l, g, a, r, l'\rangle \in E$, it is written, $l \rightarrow_{g,a,r} l'$.

The semantics of a timed automaton is given in terms of real valued clock assignments. A clock assignment u for C is a function u: $C \rightarrow$ R. Let R^C denotes the set of clock assignments for C. For $u \in \mathrm{R}^C$, $x \in C$ and $d \in$ R, $u + d$ denotes the time assignment which maps each clock x in C to the value $u(x) + d$. For $C' \subseteq C$, $[C' \rightarrow 0]u$ denotes the assignment for C which maps each clock in C' to the value 0 and agrees with u over $C \backslash C'$. A state of an automaton A is a pair (l, u) where l is a node of A and u a clock assignment for C. The initial state of A is (l_0, u_0) where u_0 is the initial clock assignment mapping all clocks in C to 0. The semantics of A is given by the timed transition system $\mathrm{S}_A = \langle S, \sigma_0, \rightarrow, V\rangle$, where S is the set of states of A, σ_0 is the initial state (l_0, u_0), $\rightarrow$ is the transition relation defined as follows:

- $(l, u) \rightarrow_{\boldsymbol{a}} (l', u')$ if there exist r, g such that $l \rightarrow_{g,a,r} l'$, $g(u)$ and $u' = [r \rightarrow 0]u$;
- $(l, u) \rightarrow_{\varepsilon(d)} (l', u')$ if $(l = l')$, $u' = u + d$, and V is extended to S by $V(l, u) = V(l)$.

Finally, for a pair of timed automata A and B and synchronization function f, the parallel composition $A||_f B$ denotes the timed transition systems $S_A \parallel_f S_B$.

3.2 Defining linear processes P^i

To transform the precedence graph $\boldsymbol{G}_i$ of tasks to a composition of timed automata we consider two transformations of ordering relation $\boldsymbol{R}_k \in \mathcal{P}(\boldsymbol{R})$: isomorphic (denoted by $PG \rightarrow_{\mathrm{iso}} TA$) and partial order reduced (denoted by $PG \rightarrow_{\mathrm{por}} TA$) transformations.

Transformation $PG \rightarrow_{iso}$ *TA*. We call the transformation $PG \rightarrow_{\mathrm{iso}} TA$ isomorphic w.r.t. $\boldsymbol{R}_k$ if in TA-representation the ordering of tasks given by $\boldsymbol{R}_k$ is preserved completely. This transformation is appropriate when all possible task execution sequences, allowed by $\boldsymbol{R}$, should be preserved in the operation-level model. This is the case when equivalent in line-level task sequences may have important differences in later design phases, e.g., when order dependent features appear to be of design concern.

The transformation $PG \rightarrow_{\mathrm{iso}} TA$ is accomplished in following steps:

(i) *Sequencing of tasks assigned to the workstation* ws_i. The goal of this step is to define a set of sequential processes $\boldsymbol{P}^i$, that consist of totally ordered sequences of local tasks T^i. Each process $\boldsymbol{P}^i_j \in \boldsymbol{P}^i$ is defined by a subset of relations $\boldsymbol{R}_j \subseteq \boldsymbol{R}$, where $\boldsymbol{R}_j = \{\langle T_k, T_l\rangle \in \boldsymbol{R}$: $T_k, T_l \in T^i \wedge \forall\langle T_k, T_r\rangle \in \boldsymbol{R}_j \Rightarrow T_r = T_l\}$. The problem of finding sequential processes $\boldsymbol{P}^i$ on $\boldsymbol{R}_k$ can be stated as a digraph analysis problem on local precedence graph $\boldsymbol{G}_k$: "Find the minimal set of paths in $\boldsymbol{G}_k$ so that each node $T^k_j \in T^k$ is lying exactly on one path". By adding auxiliary arcs and nodes to $\boldsymbol{G}_k$, the solving of this problem is reducible to the recursive search of Hamiltonian cycles. This is generally *NP-complete* problem and applicable in practice only when the number of partially ordered tasks is small.

(ii) *Constructing timed automata of processes* $\boldsymbol{P}^i$. Each process $\boldsymbol{P}^i_j \in \boldsymbol{P}^i$ is modeled by a timed automaton $\mathcal{TA}^i_j$, using following steps (the indexes of workstations and processes are omitted for tasks and corresponding to them elements of TA-models when it is understood from the context):

- define a local clock cl, that simulates the execution times of tasks $T_k \in T(\boldsymbol{P}^i_j)$;
- for each task $T_k \in T(\boldsymbol{P}^i_j)$ define a state s_k with state invariant $Inv(s_k) \equiv cl \leq d_k$;
- for each local state s_k introduce an auxiliary state s'_k so that the whole set of states $\mathcal{S}(\mathcal{TA}^i_j) = \cup_k (s_k \cup s'_k)$;
- for each pair $\langle T_k, T_l\rangle \in \boldsymbol{R}^i_j$ two transitions (s_k, s'_k) and (s'_k, s_l) are introduced;
- transitions (s_k, s'_k) are supplied with clock guards $\mathcal{G}(s_k, s'_k) \equiv cl = d_k$ and transitions (s'_k, s_l) with clock resets $\mathcal{A}sgn(s'_k, s_l) \equiv cl := 0$.

(iii) *Modeling interprocess precedence constraints*. The arcs of precedence graph $\boldsymbol{G}_k$ connecting tasks of different processes constitute the set of interprocess synchronization constraints. For each pair $\langle T_p, T_r\rangle \in \boldsymbol{R} \backslash (\cup_{\mathrm{j}} \boldsymbol{R}^i_j)$, where $T_p \in T^i{}_k$ and $T_r \in T^i{}_l$, we define a global Boolean variable ℓ_{pr}; supply the transition (s_p, s'_p) of $\mathcal{TA}^i_k$ with assignment ℓ_{pr} := *true*; extend the guard $\mathcal{G}(s_o, s_r)$ with conjunct ℓ_{pr} = *true*; and add an assignment ℓ_{pr}:= *false* to the reset function $\mathcal{A}sgn(s_o, s_r)$.

(iv) *Modeling "no waiting time"-assumption between task executions*. There is no time delay between executions of tasks within a workstation ws_k. To model this assumption we extend the descriptions of all auxiliary states s'_k and transitions from and to s'_k as follows:

- Define a clock invariant $\mathcal{Inv}(s'_k) \equiv cl \leq gcd$, where *gcd* is a greatest common divisor of constants occurring in clock conditions of the $\mathcal{TA}^i$;
- Add the guard $\mathcal{G}(s'_k, s'_k) \equiv cl = gcd$, and reset function $\mathcal{Asgn}(s'_k, s'_k) \equiv cl := 0$
- Extend the guard $\mathcal{G}(s'_k, s_l)$ with conjunct $cl = gcd$;
- Extend the reset function $\mathcal{Asgn}(s_k, s'_k)$ with assignment $cl := 0$.

(v) *Modeling non-deterministic choice between partially ordered tasks.* Parallel composition of finite automata (according to the interleaving semantics) models partial order between states (transitions) of automata in untimed case. In case of timed automata the pure parallel composition is not sufficient for avoiding simultaneous time progress in processes. To guarantee that tasks are executed strictly one after other, we should ensure the mutual exclusion between processes. Critical sections are unprimed states S^u of $\mathcal{TA}^i_k$. Mutual exclusion is implemented as follows:

- Introduce a global Boolean variable $\mathcal{lock}$;
- Extend the guards of all transitions from primed to unprimed states (s'_k, s_l) with conjunct $\mathcal{lock} = true$;
- Extend the reset functions of all transitions from unprimed to primed states (s_l, s'_l) with assignment $\mathcal{lock} := false$.

As it is shown in case of Fischer's mutual exclusion protocol (Kristoffersen, et al., 1997), the modeling of mutually exclusive non-deterministic behaviors is computationally very expensive and analysis of systems with more than 10 processes is practically undecidable by ordinary non-compositional methods.

Transformation $PG \rightarrow_{por} TA$. As an alternative to computationally hard $PG \rightarrow_{iso} TA$ transformation we introduce the *partial order reduced* (POR) *transformation* $PG \rightarrow_{por} TA$ that provides instead of a (possibly large) set of synchronized parallel processes a single totally ordered sequence of tasks. The idea of the transformation is following: the task precedence graph fragment $\boldsymbol{G_k}$ defines a set of task execution sequences, where some sequences differ only by the order of partially ordered tasks. Since the execution times of tasks do not depend (by assumption) on the order of tasks, the total execution times of sequences are equal. We call such sequences *partial order equivalent sequences* and the whole set of partial order equivalent sequences *partial order equivalence class.*

By choosing an arbitrary sequence from the equivalence class, we get the sequence that represents the properties of the whole class. Applying this reduction procedure recursively on the set of representative sequences, we end up with a single sequence $\boldsymbol{P}^*$, that represents the whole precedence graph fragment $\boldsymbol{G_k}$.

The POR approach is appropriate when the further design refinements do not need comparative exploration of all possible task sequences, i.e., the partial order equivalence relation is invariant w.r.t. applied design refinements. The $PG \rightarrow_{por} TA$ transformation can be easily implemented using topological sorting algorithm TOPSORT (Reingold, et al., 1997) having time complexity $O(|\boldsymbol{T_k}| + |\boldsymbol{E}|)$, where $|\boldsymbol{T_k}|$ is number of nodes (tasks), and $\boldsymbol{E}$ number of edges of $\boldsymbol{G_k}$.

The $PG \rightarrow_{por} TA$ transformation has several advantages over the transformation $PG \rightarrow_{iso} TA$:

1. The step (*i*) being at least *NP-complete* is replaced by fast $O(|\boldsymbol{T_k}| + |\boldsymbol{E}|)$ algorithm;
2. The step (*iii*) is omitted because there is no inter-process ordering constraints;
3. The constraint of step (*iv*) "no waiting time between executions" is trivially satisfied since all primed states can be defined now as committed states and all transitions from and to unprimed states will be synchronized with clock constrained transitions of machine automata $\boldsymbol{M^i}$ (see step (*iii*)).
4. The step (*v*) is omitted since POR procedure eliminates non-deterministic choices between partially ordered tasks.
5. The arbiter automaton can be omitted since the fixed order of P^* does not leave the room for alternative selection strategies such as bounded fairness, dynamic priorities etc. that are natural for cyclic non-deterministic processes.

3.3 Constructing machine models M^i

As a rule, operation-level models refine the line-level modeling assumptions. A WS model represents a set of machines with operations that are subject to configuration constraints, detailed assumptions about operation times, their timing, ordering and cost. By the workstation's ws_i configuration model $\boldsymbol{M^i}$ we mean the composition of machine models $\boldsymbol{M^i} = \|_l \boldsymbol{M_l}$ where each machine performs its operations sequentially and machines are synchronized through processes $\boldsymbol{P^i}$.

(*i*) *Operation models.* A machine M_l is characterized by a set of its operations $\boldsymbol{Op^l}$ and operational modes $\mathcal{M}^l$. The attributes of an operation $op_j \in \boldsymbol{Op^l}$ may be priority, cost, time, pre-, post-condition etc. For simplicity we consider only execution time and cost further. To construct a machine model $\boldsymbol{M_l}$ we define:

- a set of states $\mathcal{S}(\boldsymbol{M_l}) = \{s_j: j = [1,|\boldsymbol{Op^l}|]\} \cup \{s_{idle}\}$ s.t. for each operation $op_j \in \boldsymbol{Op^l}$ there is a state s_j; the state s_{idle} is a special state that models the idle state of the machine M_l;
- a set of transitions $\mathcal{T}(\boldsymbol{M_l}) = \cup_{l=[1,|Opl|]}\{(s_j, s_{idle}), (s_{idle}, s_j)\}$;
- the duration d_j of operation op_j is modeled using the state invariant $\mathcal{Inv}(s_j) \equiv cl \leq d_j$ and the guard $\mathcal{G}(s_j, s_{idle}) \equiv cl = d_j$, where cl is a clock variable of the machine model $\boldsymbol{M_l}$;

- the cost of operation op_j is modeled as an assignement $\mathcal{A}sgn(s_{idle}, s_j) \equiv a\_cost := a\_cost + Cost(op_j)$, where *a_cost* denotes the accumulated cost of performing the operation op_j. Alternatively *a_cost* may model common cost for all operations of the machine or even of the workstation. If the accumulated cost is limited by some value *Limit*, it is represented as an operation guard $\mathcal{G}(s_{idle}, s_j) \equiv (a\_cost + Cost(op_j)) < Limit$.

(*ii*) *Operational modes*. Operations of a machine M_l are grouped into modes $\mathcal{M}^l$. Being in the mode $\mathcal{M}^l_k \in \mathcal{M}^l$ the machine is able to perform only operations $op_i \in \mathcal{M}^l_k$. To perform an operation $op_j \notin \mathcal{M}^l_k$ the machine should switch over to the mode $\mathcal{M}^l_r$ where $op_j \in \mathcal{M}^l_r$. Switching takes time and has a cost and may be constrained so that only specified switching sequences are legal. That needs extension of machine model $\boldsymbol{M}^l$ by introducing a *mode switching fragment*. Assume that each *k*-th mode $\mathcal{M}^l_k$ is modeled separately as described in (*i*) above and has its idle state s^k_{idle}. Then the mode switching fragment consists of a set of transitions between the idle states s^k_{idle} and states modeling switching operations exactly in the same way as any other machining operations (see step (*i*) above).

(*iii*) *Synchronizing processes and machine operations*. The workstation processes $\boldsymbol{P}^i$ define the ordering of tasks. Each task can be implemented as a sequence of operations. The process *Planner* $\boldsymbol{A}^i$ makes planning choosing appropriate operation sequences to execute the task on the given workstation configuration. Machine operations define the proper timing of operation sequences. Therefore, to model the cooperative behavior of these three parties (processes, planner, machines), initiations and terminations of tasks, operation sequences and individual operations must be synchronized.

Synchronization is modeled using two types of channels: *start* and *stop*. Channel *start* synchronizes initiation and channel *stop* synchronizes completing the task and operation executions in the process, task and machine models. *Start* channels are directed from process models to task models and further from task models to machine models. *Stop* channels, on the contrary, are directed from machine to task and from task to process models.

4. ANALYSIS BY MODEL CHECKING

The problem of estimating workstation time t^i_{ws} can be formulated now as a model checking problem on *TA*-model: $\boldsymbol{M} \models \varphi$, where φ denotes the behavioral property to be checked and $\boldsymbol{M}$ the model representing the behavior to be checked. Finding parameter values using model checking is generally called a parametric model checking.

The properties that the model must satisfy are given in timed modal logic $\mathcal{L}_s$ studied in (Alur and Dill, 1994) and used currently in the verifier of UPPAAL2k. The BNF-grammar of $\mathcal{L}_s$:

$\varphi ::= \mathbf{A}\square\, P_s \mid \mathbf{E}\Diamond\, P_s \mid \mathbf{E}\square\, P_s \mid \mathbf{A}\Diamond\, P_s$

$P_s ::= AP \mid \neg P_s \mid (P_s) \mid P_s \vee P_s \mid P_s \wedge P_s \mid P_s \Rightarrow P_s$

$AP ::= Id_1.Id_2 \mid CGuard \mid IGuard$

$CGuard ::= Id \sim n \mid Id \sim Id \mid Id \sim Id + n \mid Id \sim Id - n,$ where $n \in \mathbf{N}$

$IGuard ::= IExpr \sim IExpr \mid IExpr \neq IExpr$

$IExpr ::= Id \mid Id[IExpr] \mid n \mid -IExpr \mid (IExpr) \mid IExpr\ Op\ IExpr$

$\sim ::= < \mid \leq \mid \geq \mid > \mid =$

$Op ::= + \mid - \mid * \mid /,$

where P_s is a state formula, *AP*- atomic state formula, *CGuard* and *IGuard* are the guards over clocks and integer variables respectively, *Id* identifier name; $Id_1.Id_2$ – an identifier in the form "*automaton name.state name*", *n* - natural number (including 0), and temporal modalities: **A** - *always*; **E** – *sometimes*; $\square$ - *globally*; $\Diamond$ - *eventually*.

For example, the formula $A \square (v_1 < v_2)$ says that invariantly $v_1 < v_2$ holds and the formula $E\Diamond(A_1.s_i \wedge A_2.s_i)$ is true iff the system can reach a global state where both automata A_1 and A_2 are in their states s_i.

The tasks to be solved by MC in the context of line balancing problems are related to time estimates but may consider other model parameters such as production deadlines, store capacity, lot size, etc. All these problems can be stated formally as parametric constraint solving tasks. Specifying global constraints by the formula φ to be checked and local to some ***i***-th component (operation, task, workstation) constraints as transition guards or assignment conditions of that component's model A_i, we transform the problem into a standard model checking problem $\ldots\|A_i\|\ldots \models \varphi$, where $\ldots\|A_i\|\ldots$ denotes the composition of models including the model A_i where the constraints are encoded.

Time estimates needed for LB are expressed generally as *bounded liveness properties* meaning that being in some specified state s_i there exists a path in the model reaching the state s_j within *t* time units. Bounded liveness properties can be expressed formally as safety properties and checked efficiently using, e.g., the technique of test automata (Larsen *et. al.*, 1997). For instance, the *line*-level WS time estimate t_{ws} can be checked in operation-level model by an auxiliary automaton *Stop_watch*. The automaton *Stop_watch* is constructed so that it takes transition to state *Time_out* if tasks allocated to the WS are not completed within the time period t_{ws}. The MC task to be solved now is: $\mathcal{M}^i \models E \Diamond (Stop\_watch.time\_out)$. If this property is satisfied then the operation-level estimate of t_{ws} exceeds the value used in line-level model. The actual operation-level estimate of t_{ws} can be reached by varying the *time_out* parameter of *Stop_watch* automaton.

5. FINE-GRAIN MODELING EXAMPLE

The proposed approach is tested using representative open test data set for SALB problems (Scholl and Klein, 1996).

Consider the robotic workstation, which is described in (Vain *et. al.*, 2002) as a WS of a flexible assembly line. It picks up, checks, adjusts, and assembles two details. The WS time is found to be critical, and must be checked in detail. The WS processes the following precedence of subtasks. The loading machine takes a blank (type 1 or type 2) from the main conveyor and puts it on the Conveyor1 (task 1, subtask 1, 12s). The Conveyor1 transports the blank to the robot Mentor (task 1, subtask 2, 3s) that picks it from Conveyor 1, and places to (local) Conveyor 2 (task 1, subtask 3, 5s). On the Conveyor 2 by the aid of robot Serpent the blank is measured, classified, and positioned (task 2, subtask 1, 13s) where the CNC Finishing Mill works it into the finished part (task 2, subtask 2, 40s). After completion of subtask 2 the robot Serpent removes the processed item from finishing and places it into a box (task 2, subtask 3, 17s) that is positioned under Serpent by the Index table. Similar sequence of subtasks is completed with the blank of another type. The Mentor picks different details from boxes and positions them (task 3, subtasks 1,2, 20s). Finally, the details are assembled by an Assembling unit (task 3, subtask 3, 40s). The results of checking show that it is possible to reduce the WS time (and the cycle time of the whole assembly line) from 280 sec to 259 sec.

6. CONCLUSION

In this paper an algorithm is proposed which is aimed at reduction of the cycle time of (possibly flexible) assembly lines. The algorithm is based on a repetitive two-phase solution of a line balancing problem, allocating the tasks between workstations at current time step in an optimal way. It combines the advantages of branch-and-bound algorithms efficient on the level of coarse-grain task precedence models and, on the other hand, model checking methods that allow fine grain analysis in the presence of different operational and timing constraints. Traditional drawback of the model checking – exponential complexity growth in the number of parallel components of the model – is avoided by solving subtasks in isolation and considering only local to a workstation information. Critical aspect in applying the method is constructing detailed operational models that requires experience and time. It is shown that detailed model construction can be enhanced using a small set of well-defined model construction and problem specification rules.

ACKNOWLEDGEMENTS

This work is supported by the Estonian Science Foundation under grants 5086 and 5775.

REFERENCES

Alur, R., D. Dill (1994), Automata for modeling Real-Time Systems. *Theoretical Computer Science*. **126**, 183-236.

Clarke, E.M., O. Grumberg and D.A. Peled (1999). *Model Checking*. The MIT Press.

Hoffmann, T.R. (1992). EUREKA: A hybrid System for Assembly Line Balancing. *Management Science*, **38**, 39-47.

Hopp, Wallace J.and Mark L. Spearman. *Factory physics. Foundations of Manufacturing Management*. Irwin/McGraw-Hill.

Kristoffersen, K.J., F. Laroussinie, K.G. Larsen, P. Pettersson, W. Yi (1997). A Compositional Proof of a Real-Time Mutual Exclusion Protocol. In: *Proceedings of the 7/th/ International Joint Conference on the Theory and Practice of Software Development*. (Michel Bidoit and Max Dauchet, (Ed)), 565-579. Lille.France.

Larsen, K., P. Pettersson, W. Yi (1997), UPPAAL in a Nutschell. *Int. Journal on Software Tools for Technology Transfer* **1**, 134-152.

Lindahl, M., P. Pettersson, W. Yi (1998). Formal Design and Analysis of a Gear Controller. *Lecture Notes in Computer Science* 1384, 281-297.

Reingold, M., J. Nievergelt and N. Deo. (Ed). (1977) *Combinatorial Algorithms: theory and practice*. Prentice Hall, Englewood Cliffs.

Scholl, A., R. Klein (1996). *Assembly line balancing*. Technical University of Darmstadt, http://www.bwl.tu-darmstadt.de/bwl3/forsch/projekte , Darmstadt.

Scholl, A., R. Klein (1997). SALOME: A Bidirectional Branch and Bound Procedure for Assembly Line Balancig. *INFORMS Journal on Computing* **9**, 319-334.

Scholl, A. (1999) *Balancing and Sequencing of Assembly Lines. 2nd edition* <http://www.wiwi.uni-jena.de/Entscheidung/bucher2.htm>, Physica-Verlag, Heidelberg.

Vain,J. and R.Küttner (2001). Model Checking - a New Challenge for Design of Complex Computer-Controlled Systems. In: *Proc. of 5-th International Conference on Engineering Design and Automation.* (H.R.Parsaei *et al* eds.). 593-598, CD-ROM, CRC Press, Inc., USA.

Vain, J., I. Randvee, T. Riismaa, and J. Ernits (2002). Solving line balancing problems with model checking, *Proceedings of Estonian Academy of Sciences. Engineering* **8**, 211-222.