Integration sequencing in complex manufacturing systems

Systems Engineering Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands
http://seweb.se.wtb.tue.nl/

SE-Report: Nr. 2006-02

Integration sequencing in complex manufacturing systems

R. Boumen, I.S.M. de Jong, J.M.G. Mestrom,
J.M. van de Mortel-Fronczak and J.E. Rooda

ISSN: 1872-1567

SE Report: Nr. 2006-02
Eindhoven, October 2006
SE Reports are available via http://seweb.se.wtb.tue.nl/sereports

Abstract

The integration and test phase of complex manufacturing machines, like an ASML [1] lithographic machine, are expensive and time consuming. The tests that can be performed at a certain point within the integration phase depend on the modules that are integrated. Therefore, the test sequence depends on the integration sequence. Thus, by optimizing the integration sequence of these modules, more tests can be done in parallel and valuable integration and test time can be reduced. In this paper, we introduce a mathematical model to describe an integration sequencing problem and we propose an algorithm to solve this problem optimally. Furthermore, we propose two heuristics to solve large industrial problems in limited computation time. Also, we show with a case study within the development of a lithographic machine that the described method can be used to solve real-life problems.
TANGRAM, test strategy, test sequencing, manufacturing machines, semiconductor industry, integration sequencing.

Chapter 1

Introduction

In today's industry, time-to-market of systems is more and more important. Therefore, the development of systems is done concurrently. During the integration phase of a system, the different sub-systems are assembled into a system and tested. This integration and test phase typically takes more than 45% of the total development time of a complex manufacturing system. Reducing this time reduces the time-to-market of a new system.
An integration plan describes the integration actions and the test cases that are performed in the integration and test phase of a system. For new ASML machines, this integration plan is currently made by hand and is usually sub-optimal for time. Creating an optimal integration plan could decrease integration and test time and planning effort.
In literature, integration plans for complex systems are seldom discussed. Generally, the basic integration test phases identified are unit testing, integration testing, system testing and acceptance testing as for example explained in [2]. However, this integration and test plan does not define the sequence of integration: which modules are integrated when? It only states that tests should be done before and after integration. More research is done within the software discipline on integration strategies. The three basic strategies are: "big bang", where all modules are integrated at once, "top-down" where stubs are used to simulate the behavior of lower-level modules while testing the high-level modules first, and "bottom-up" where first all lower level modules are tested and then the higher-level module is integrated [3]. The disadvantage of these three strategies is that none of them is optimal given a certain integration problem: a combination of them is optimal. For object-oriented software, Hanh *et al.* [4] developed a technique that tries to either minimize the number of stubs used or the total test resource allocation (test effort). This is done using a test-dependance-graph model describing the modules and their test dependencies. This approach tries to optimize an integration strategy towards a combination of "big-bang", "bottom-up" and "top-down" strategy. The disadvantage of this approach is that it has only been used for object-oriented programming and that it does not take into account that certain modules are not physically present at the start time of the integration phase. Furthermore, it only considers one single system under test, while multiple sub-assemblies could be used as systems under test to test in parallel.
Within the mechanical assembly disciplines, methods and algorithms do exist that take late deliveries and parallel assemblies into account. For example De Mello *et. al* [5], [6] describes how to represent and optimize mechanical assembly sequences such that robot assembly actions are minimized. Also Boneschanscher [7] describes how to use these techniques to optimize assembly plans. However, these methods have never been applied on integration

and test plans. The difference between assembly and integration is that the cost of an assembly plan is dependant on the robot movements, while the cost of an integration plan is dependant on the tests that are performed.
In this paper, we extend the mechanical assembly sequencing method and algorithm towards an integration sequencing method. This is done by combining the assembly sequencing representation with the test-dependency-graph model of Hanh and by extending the AND/OR graph algorithms as described by De Mello *et. al.* We incorporate for certain tests that stubs or simulation models can replace modules that are not yet available. This way, we can optimize an integration plan towards time while taking into account that certain models can replace modules as is proposed by Braspenning *et al.* [8].
The paper is structured as follows. Section 0.2 explains system integration sequencing and shows an example. Section 0.3 shows the formal definition of an integration sequencing problem and the objective for solving this problem. Section 0.4 shows the solving algorithm used to solve the integration sequencing problem. In Section 0.5 a case study is done to show the benefits of applying this method on the development phase of a lithographic machine. Section 0.6 discusses the conclusions. Appendix 0.7 shows the notations used in Appendix 0.8, which shows a formal and a step-by-step description of the algorithm.

Chapter 2

System integration

System integration deals with assembling modules into a working system. To ensure that the system works, tests have to be performed that test the functionality and performance of the system. In the first subsection, we describe several system integration problems. In the second subsection, we introduce an illustration of an integration problem which is used throughout the paper.

2.1 Integration problem

Integration problems can be found in many different development phases of systems that all have their specific properties. The three main integration problems during the development and manufacturing of ASML lithographic machines are:

- Software integration
- First-of-a-kind machine integration
- Manufacturing integration

During software integration, multiple copies of the software are available and the assembly of two software modules into one system does not take any (significant) time. Often, stubs can be used to perform certain tests early, while modules are not yet available.
For the first-of-a-kind machine integration, the development cost of each module is extremely high and only one or maybe two modules of (almost) the same type are available. Furthermore, the assembly of two modules into a single system takes a significant amount of time.
For manufacturing integration, a large number of machines is created which means that multiple modules of the same type are available. Also, assembling modules takes time.
In general, an integration phase consists of assembling a number of modules, that are available at a certain time, into one system and performing a number of tests, each of which takes some time to perform and requires a certain set of modules to be assembled. To reduce the overal integration time, all development, assembly and test actions should be performed as parallel as possible. The integration sequencing problem is defined as choosing an assembly sequence of which the duration of the critical path is minimal which means that the development, assembly and test actions are performed as parallel in time as possible. The critical path is defined as the longest duration path in the integration sequence.

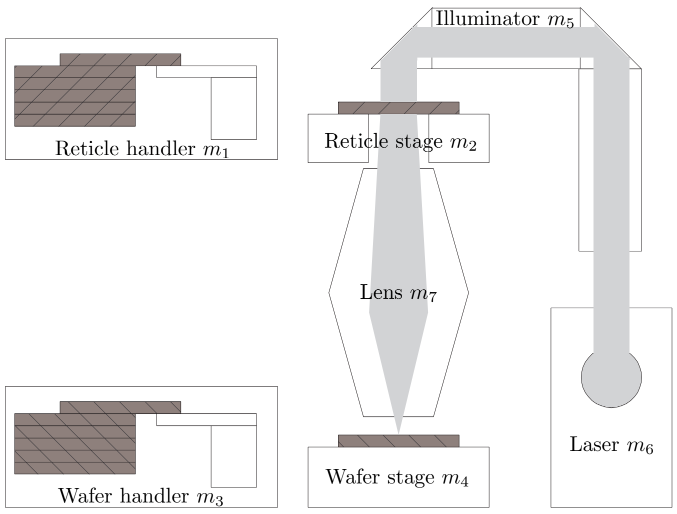


Figure 2.1: Scanner Illustration

2.2 Illustration

To illustrate an integration sequencing problem we use a simplified ASML wafer scanner. A scanner performs the lithographic step within the manufacturing of a semiconductor or IC. Two items are cycling through a scanner: a wafer that is the basis of the IC and contains a photo resistant, and a reticle that contains a part of the negative of an IC [9]. Our example scanner consists of 7 modules: a reticle handler (m_1) that brings and takes reticles to the reticle stage (m_2) which holds the reticle during the lithographic process, a wafer handler (m_3) that brings and takes wafers to the wafer stage (m_4) which holds the wafer during the lithographic process, a laser (m_6) that produces the light needed for the lithographic process, an illuminator (m_5) that uniforms the light produced by the laser and a lens (m_7) that shrinks and images the pattern from the reticle on the wafer. Reticles and wafers are not considered part of the machine in this example. Within this machine 3 essential types of interfaces exist: the light interfaces (i_1, i_2, i_4, i_5) which connect the laser to the illuminator to the reticle stage to the lens to the wafer stage, the reticle flow interface (i_3) which connects the reticle handler to the reticle stage and the wafer flow interface (i_6) which connects wafer handler to the wafer stage.
This illustration of an integration sequencing problem deals with the integration phase of a first type of a scanner. Besides the mentioned modules and interfaces, for this scanner, a model of a lens (m_8) is available which can be used for certain tests. This model also has light interfaces (i_7 and i_8) with the reticle handler and the wafer handler. The development time for each of the modules is known. For example, the development time of the lens model is 5 time units, while the development time of the actual lens is 25 time units. Also, the integration of two modules into a subsystem takes time. Furthermore, 25 tests have to be performed that each require certain modules to be integrated and cost a certain time to perform. The objective is to find an integration sequence that integrates each module into a complete system such that the overal integration and test time is minimized.

Chapter 3

Problem formulation

In this section, we formalize the integration sequencing problem. We define a model in the first subsection, show an illustration model in the second subsection and define the objective in the third subsection.

3.1 Integration model

The formal integration model consists of two parts: 1) the assembly part describing the system based on models from De Mello *et. al.* [5, 6], and 2) the test part describing tests and the required modules based on ideas from Hanh *et al.* [4]. The complete integration sequencing model is an 8-tuple D: $(M, I, T, C^m, C^i, C^t, R^{im}, R^{tm})$, where:

- M is a finite set of k modules.
- I is a finite set of l interfaces.
- T is a finite set of m tests.
- $C^m : M \rightarrow \mathbb{R}^+$ gives for each module in M the associated cost in time units of developing that module.
- $C^i : I \rightarrow \mathbb{R}^+$ gives for each interface in I the associated cost in time units of constructing that interface.
- $C^t : T \rightarrow \mathbb{R}^+$ gives for each test in T the associated cost in time units of performing that test.
- $R^{im} : I \rightarrow M \times M$ gives for each interface in I the modules the interface is constructed in between.
- $R^{tm} : T \rightarrow \mathcal{P}(\mathcal{P}(M))$ gives for each test in T its essential assemblies; where an essential assembly describes the modules that should be integrated with each other before the test can be performed.

The assumptions for this integration model are:

- All modules in M must be connected with each other, so there exists a path of interfaces that connects every module in M with every other module in M.

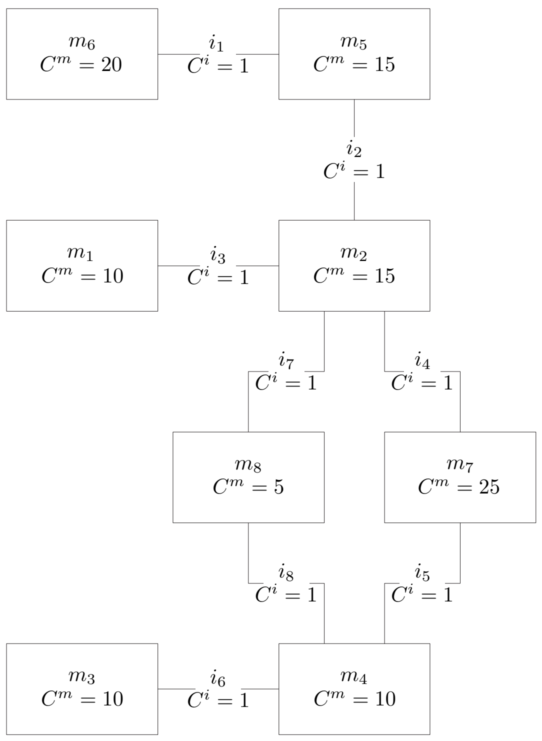


Figure 3.1: Elements M, I, C^m, C^i and R^{im} of the integration model

- For every test in T, there exists a module that is present in all essential assemblies of this test.
- Each test is performed exactly once at the moment that one of the essential assemblies of this test is integrated.

Furthermore, we define that an assembly consists of one or more modules which are integrated with each other and is therefore represented by an element of $\mathcal{P}(M)$ (except $\emptyset$). An integration action is defined as instantiating all interfaces between exactly two assemblies and is therefore represented by an element of $\mathcal{P}(I)$ (except $\emptyset$). A test phase consists of the set of tests that can be performed on a subassembly, but could not be performed before the last integration action. A test phase is therefore represented by an element of $\mathcal{P}(T)$ (except $\emptyset$).

3.2 Illustration

For the scanner illustration presented in Section 0.2, an integration model is created. Elements M, I, C^m, C^i and R^{im} are represented in Figure 2 (the integration part). In this figure, a square node is a module and an edge is an interface between two modules. Elements T, C^t and R^{tm} are shown in Table 1(the test part).

3.3 Objective

A solution to an integration sequencing problem is a sequence of test phases and integration actions for each individual module. This solution can be represented by a function $G : M \rightarrow (\mathcal{P}(T) \cup \mathcal{P}(I))^*$, ($B^*$ denotes a set of sequences over B), that gives for a subassembly consisting

Table 3.1: Elements T, C^t and R^{tm} of the integration model

T	R^{tm}	C^t
$t_1 \cdots t_6$	$\{\{m_1\}\}$	1
$t_7 \cdots t_8$	$\{\{m_1, m_2\}\}$	2
$t_9 \cdots t_{11}$	$\{\{m_4\}\}$	1
$t_{12} \cdots t_{13}$	$\{\{m_3, m_4\}\}$	2
$t_{14} \cdots t_{17}$	$\{\{m_5, m_6, m_7\}, \{m_5, m_6, m_8\}\}$	3
$t_{18} \cdots t_{20}$	$\{\{m_2, m_4, m_5, m_6, m_7\}, \{m_2, m_4, m_5, m_6, m_8\}\}$	3
$t_{21} \cdots t_{25}$	$\{\{m_1, m_2, m_3, m_4, m_5, m_6, m_7\}\}$	5

of a single element of M, a sequence of integration actions (sets of interfaces $\mathcal{P}(I)$) and test phases (sets of tests $\mathcal{P}(T)$) that integrates this single module subassembly into the completely integrated and tested system. The total time of such a solution is:

$$J(G) = \max_{m \in M}\left(C^m(m) + \sum_{t \in G(m) \restriction T} C^t(t) + \sum_{i \in G(m) \restriction I} C^i(i) \right) \tag{3.1}$$

The objective is to find an optimal solution G^* that has minimal expected test cost J^*, from all possible solutions $\mathcal{G}$:

$$J^* = J(G^*) = \min_{G \in \mathcal{G}} J(G) \tag{3.2}$$

Chapter 4

Solving algorithm

In this section, we propose an algorithm to solve the integration sequencing problem. This algorithm is based on the "assembly by disassembly" approach suggested by Delchambre *et al.* [10]. The approach takes the assembled, or in this case integrated, system as starting point and evaluates all possible sequences in which the complete system can be disassembled. De Mello *et al.* [5, 6], suggests to implement this approach in an algorithm that performs a search through an AND/OR graph. In this AND/OR graph, an OR node represents a system state and an AND node represents an assembly action.

To solve the integration sequencing problem we also propose an algorithm based on an AND/OR graph search. However, different definitions of the AND and OR nodes are used as opposed to De Mello. An OR node represents the system state x which is the set of integrated modules, with the domain $\mathcal{P}(M)$. In a system state, tests are applied that are only possible in that state and not in following states, according to the assumptions made in Section 0.3. The initial system state is $x_{init} = M$. An AND node defines the integration action of two OR nodes into one OR node and is denoted by the set of interfaces that are broken, which is an element of $\mathcal{P}(I)$.

An example of an AND/OR graph is shown in Figure 3. Each square node in the graph denotes an OR node, while each hexagonal node denotes an AND node, the edges denote the search direction. This AND/OR graph is constructed for a very simple integration model consisting of 3 modules (m_1, m_2, m_3), which are all connected to each other with 3 interfaces (i_1 connects m_1 and m_2, i_2 connects m_1 and m_3, i_3 connects m_2 and m_3) and 4 tests that need to be applied: 3 tests that each need one of the modules (t_1 requires m_1, t_2 requires m_2, t_3 requires m_3) and one system test t_4 that requires all modules. The graph in Figure 3 shows all possibilities in which the system can be disassembled, and therefore contains all solutions to the integration sequencing problem. In this example, there are three possible solutions (G_1, G_2, G_3) that can be distinguished at the root OR node where there are 3 AND nodes to choose from. For each solution G the cost $J(G)$ can be calculated, then the cheapest solution is chosen.

The algorithm that constructs AND/OR graphs is a depth-first algorithm. The basics of the AND/OR search algorithm are explained in this section. The complete algorithm is described in Appendix 0.8. The search starts with the initial OR node that denotes the complete integrated and tested system: x_{init}. The cost of this particular OR node is called $J_x(x_{init})$ and is determined as follows.

First, the possible set of disassembly actions is determined. The set of all possible integration actions A_x are all cut-sets that separate the integrated system with system state x into exactly

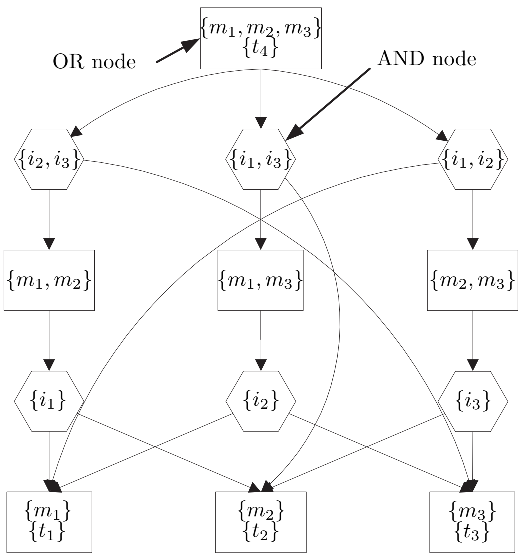


Figure 4.1: AND/OR graph illustration

two unique sub-systems (x_1 and x_2). For a given system state x, this can be determined as follows:

$$\begin{array}{l}
A_x(x) = \\
\{a : a \subseteq I_x : \\
\quad \exists (x_1, x_2) \subseteq x : \\
\qquad x_1 \cap x_2 = \emptyset \wedge x_1 \cup x_2 = x \\
\qquad x_1 \; /\!= \emptyset \wedge x_2 \; /\!= \emptyset \wedge \forall (m', m'') \in x_1 : \mathit{connected}(m', m'', I_x \setminus a) \\
\qquad \wedge \forall (m', m'') \in x_2 : \mathit{connected}(m', m'', I_x \setminus a) \\
\qquad \wedge \nexists m' \in x_1, m'' \in x_2 : \mathit{connected}(m', m'', I_x \setminus a) \\
\}
\end{array} \tag{4.1}$$

where $I_x(x)$ denotes all interfaces between the modules in system state x and function *connected* checks whether two modules are connected, i.e. there exists a path of interfaces between two modules. Cut-set algorithms exist in literature that determine all possible cut-sets of a system in lineair time per cut-set. We use the algorithm as described by Tsukiyama [11]. For the simple system of which the AND/OR graph is shown in Figure 3, the cut-sets are: $(\{i_1, i_2\}, \{i_2, i_3\}, \{i_1, i_3\})$. For the system illustrated in Figure 2, the possible cut-sets are: $(\{i_1\}, \{i_2\}, \{i_3\}, \{i_4, i_7\}, \{i_8, i_5\}, \{i_6\}, \{i_4, i_5\}, \{i_7, i_8\})$
Then for all cut-sets $a \in A_x(x)$ given system state x, an AND node is constructed. This AND node represents the disassembly of a system state into two system states x_1 and x_2 (determined in equation 3), by breaking the interfaces in a. For each of the two system states, x_1 and x_2, the tests that can still be performed, $T_x(x_1)$ and $T_x(x_2)$, can be calculated using:

$$T_x(x) = \{t : t \in T : (\exists M' \in R^{tm}(t) : M' \subseteq x)\} \tag{4.2}$$

Then, the required tests $T_r(x)$ that have to be performed in system state x can be calculated with:

$$T_r(x) = (T_x(x) \setminus (T_x(x_1) \cup T_x(x_2))) \tag{4.3}$$

The total cost of an AND node $J_a(x, a)$ for a system state x and a disassembly action $a \in A_x(x)$ which disassembles the system into two subsystems x_1, x_2, is defined as the maximal integration and test cost of each formed system state x_1 and x_2, and the cost of the disassembly of the system state x into the two system states, or:

$$J_a(x,a) = \sum_{i \in a} C^i(i) + \max(J_x(x_1), J_x(x_1)) \qquad (4.4)$$

Finally, the cost of the OR node is the minimal cost of each AND node that is constructed and the cost of performing the required tests $T_r(x)$, or the development cost of a module if one module remains and the cost of the tests $T_x(x)$ that can still be performed in system state x:

$$J_x(x) = \begin{cases} C^m(m) + \sum\limits_{t \in T_x(x)} C^t(t) & \text{if } x = \{m\} \\ \min\limits_{a \in A_x(x)} \left(J_a(x,a) + \sum\limits_{t \in T_r(x)} C^t(t) \right) & \text{else} \end{cases} \qquad (4.5)$$

where $|x|$ denotes the size (number of elements) of x. If the root node is solved, which means that $J_x(x_{init})$ is known, the complete solution is known and can be constructed. Then, the integration tree is the reverse sequence of the disassembly tree, i.e. starting with the separate modules and ending with the integrated system.

4.1 Illustration

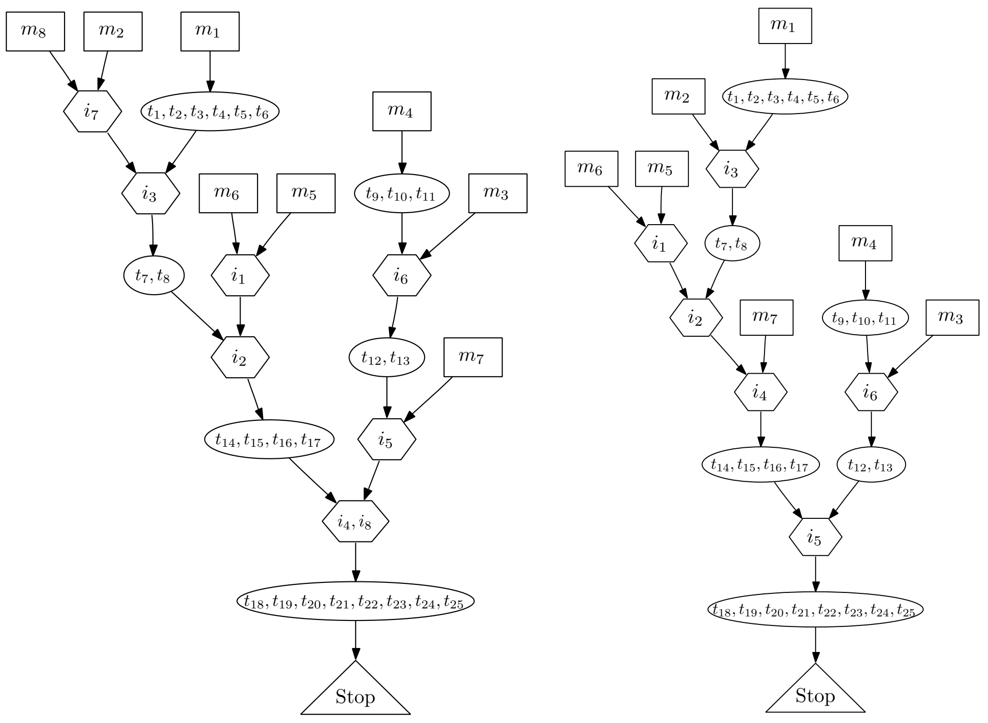
With the presented algorithm the solution for the problem described in Section 0.3 has been calculated. The solution for this problem is shown in Figure 4(a). This figure shows for each module (square node), the integration steps (hexagonal node) which consist of creating interfaces, and the test steps (circular node) that should be done. Note that this tree is the reverse of the disassembly tree. The edges denote the precedence relations between the actions, and therefore show the sequence of integration action and test phases for each module. The longest paths in this tree are the paths of modules m_1, m_2, and m_6 which are all 70 time units. Therefore the cost of this optimal solution is also 70. In this situation, the actual lens (m_7) is integrated late because it has a long development time, and the model of the lens (m_8) can be used to perform tests t_{14} through t_{17}.
In the situation shown in Figure 4(b) it is assumed that the model of the lens (m_8) is not available during the integration phase of this new scanner. The cost of this solution is 73 time units. The longest path in the tree is now the path of the lens (m_7), which is integrated earlier to perform tests t_{14} through t_{17}. For this illustration, we can conclude that the time-to-market of the scanner is 3 time units less when the model of the lens (m_8) is developed and used. Therefore, one has to consider whether this time reduction is worth the effort of building the model of the lens.

4.2 Computational reduction measures

The computational effort for searching an AND/OR graph is NP-hard:

$$\text{Number of OR nodes investigations} \approx 2^{|M|} - 1 \qquad (4.6)$$

according to De Mello in [5] for a strongly connected module graph with the check that only feasible cut-sets remain. This means that problems of 20 modules need over 1 million OR node investigations. Since industrial problems often have more than 20 modules, we need some computational reduction measures to be able to solve them in a reasonable computation time.
Two measures (heuristics) are proposed to reduce this computational effort. Both heuristics try to reduce the computational effort by reducing the number of possible cut-sets (AND nodes) per OR node. This is done by only investigating the most promising cut-sets.
The first heuristic is the 'Early Time' (ET) heuristic. The idea is that the cost for test and integration should be spent as early as possible in the integration sequence to make the sequence as parallel as possible. Thus, integration actions (cut-sets) are selected that have


m_8
m_2
m_1
i_7
$t_1, t_2, t_3, t_4, t_5, t_6$
m_4
i_3
m_6
m_5
t_9, t_{10}, t_{11}
m_3
t_7, t_8
i_1
i_6
i_2
t_{12}, t_{13}
m_7
$t_{14}, t_{15}, t_{16}, t_{17}$
i_5
i_4, i_8
$t_{18}, t_{19}, t_{20}, t_{21}, t_{22}, t_{23}, t_{24}, t_{25}$
Stop
m_1
m_2
$t_1, t_2, t_3, t_4, t_5, t_6$
m_6
m_5
i_3
i_1
t_7, t_8
m_4
i_2
m_7
t_9, t_{10}, t_{11}
m_3
i_4
i_6
$t_{14}, t_{15}, t_{16}, t_{17}$
t_{12}, t_{13}
i_5
$t_{18}, t_{19}, t_{20}, t_{21}, t_{22}, t_{23}, t_{24}, t_{25}$
Stop

(a) Using the lens model (m_8), $J^* = 70$ (b) Without using the lens model (m_8), $J^* = 73$

Figure 4.2: Solutions of the scanner illustration

Table 4.1: Results heuristics experiment

Experiment	J^*	OR nodes	Time
Optimal	70	531	100%
ET($n = 1$)	86	15	3,7%
ET($n = 2$)	76	89	14,9%
ET($n = 3$)	71	177	30,2%
ET($n = 5$)	71	345	59,8%
PT($n = 1$)	74	15	2,7 %
PT($n = 2$)	70	65	8,7%
PT($n = 3$)	70	113	17,4%
PT($n = 5$)	70	235	43,0%

the lowest test and integration cost, such that more cost are spent in the lowest parts of the AND/OR graph. The time spent on integration and test in an AND node $a \in A_x(x)$ is:

$$ET(a, x) = \sum_{i \in a} C^i(i) + \sum_{t \in T_r(x)} C^t(t) \tag{4.7}$$

Where $T_r(x)$ denotes the tests that are applied on that subsystem, as calculated by equation 5. The ET heuristic selects the n cut-sets that have the lowest ET to be investigated further, where n is a user-defined natural number.
The second heuristic is the 'Parallel Time' (PT) heuristic. The idea is that the more actions (test and integration) are done in parallel, the more optimal the total integration plan is. This is done by selecting cut-sets that disassemble a system state x into two system states x_1 and x_2 that have (almost) the same total duration. Therefore, for each cut-set from $a \in A_x(x)$, the total duration of the two formed sub-systems $J_{tot}(x_1)$ and $J_{tot}(x_2)$ is calculated using the following equation, for $x' = x_1 \vee x' = x_2$:

$$J_{tot}(x') = \sum_{m \in x'} C^m(m) + \sum_{i \in I_x(x_1)} C^i(i) + \sum_{t \in T_x(x')} C^t(t) \tag{4.8}$$

Where $I_x(x')$ denotes all interfaces that are present in system state x'. Furthermore, T_x can be calculated with Equation 4. The total duration for a system state is the time that it takes to develop all modules, create all interfaces and perform all tests sequentially, so the sum of development actions, integration actions and tests. Then, the information gain approach is used to determine to what extent the two total times are equal:

$$PT(a, x) = IG\big(J_{tot}(x_1), J_{tot}(x_2)\big) \tag{4.9}$$

Where the IG is defined by [12]:

$$IG(x, y) = -\left(\frac{x}{x+y} \cdot log_2(\frac{x}{x+y}) + \frac{y}{x+y} \cdot log_2(\frac{y}{x+y})\right) \tag{4.10}$$

The PT heuristic selects the n best cut-sets that have the highest PT to be investigated further, where n is a user-defined number.
With the following experiments, the influence of the proposed heuristics is investigated. For each experiment, different heuristics are used to calculate a solution for the Scanner illustration in Figure 2. As shown in Figure 4(a), the cost of the optimal solution is 70 time units. In Table 2, the cost and computational effort, denoted in the number of OR node calculations and in time (relative to the time needed to calculate the optimal solution), is shown for each of the experiments. For the ET heuristic it is shown that for $n = 1$ the solution is far from optimal, while for $n = 5$ the solution approaches the optimal one. For the PT heuristic it is shown that for $n = 1$ the solution is better than the ET($n = 1$) heuristic, and that for $n = 2$ the solution is already optimal. For this illustration, it is best to use the PT heuristic.

Chapter 5

Case study

The presented method has been applied to a case study within the development of a new type of lithographic machine at ASML. The case study deals with the integration plan for the development of a first-of-a-kind machine. The general approach of these integration plans is that an old version lithographic machine is upgraded to the new version by integrating new parts and testing these parts and the total performance of the machine.
The model properties of this case study are the following:

- 17 modules
- 20 interfaces
- 17 tests

Another interesting property of this case is that module m_0 is always needed for each test since it represents the initial old version of the machine. Therefore, no tests can be done in parallel and all integration actions and test phases will occur sequentially. In this case study, two integration sequences are calculated. The first sequence shown in Figure 5 gives the plan at the beginning of the project. The second sequence shown in Figure 6 shows a re-plan situation when it became clear that certain modules would arrive later than planned in the first sequence.

5.1 Situation 1

For the case study the resulting integration plan is shown as a Gantt chart in Figure 5. This Gantt chart displays each test phase consisting of a set of tests, each integration action consisting of a set of interfaces and each development action as a task. Each task has a certain duration and has a successor denoted with arrows between the tasks. The development tasks are numbered: 1, 3, 5, 8, 11, 13, 15, 18, 20, 21, 24, 28, 31, 34, 37, 40, 43, while the integration tasks are numbered: 4, 6, 9, 12, 14, 16, 19, 22, 23, 26, 29, 32, 35, 38, 41, 44, and the test tasks are numbered: 2, 7, 10, 17, 25, 27, 30, 33, 36, 39, 42, 45. It can be seen that only development actions run in parallel. The critical path (denoted in red) is formed by all integration actions and test phases and the development of the module with task ID 5.

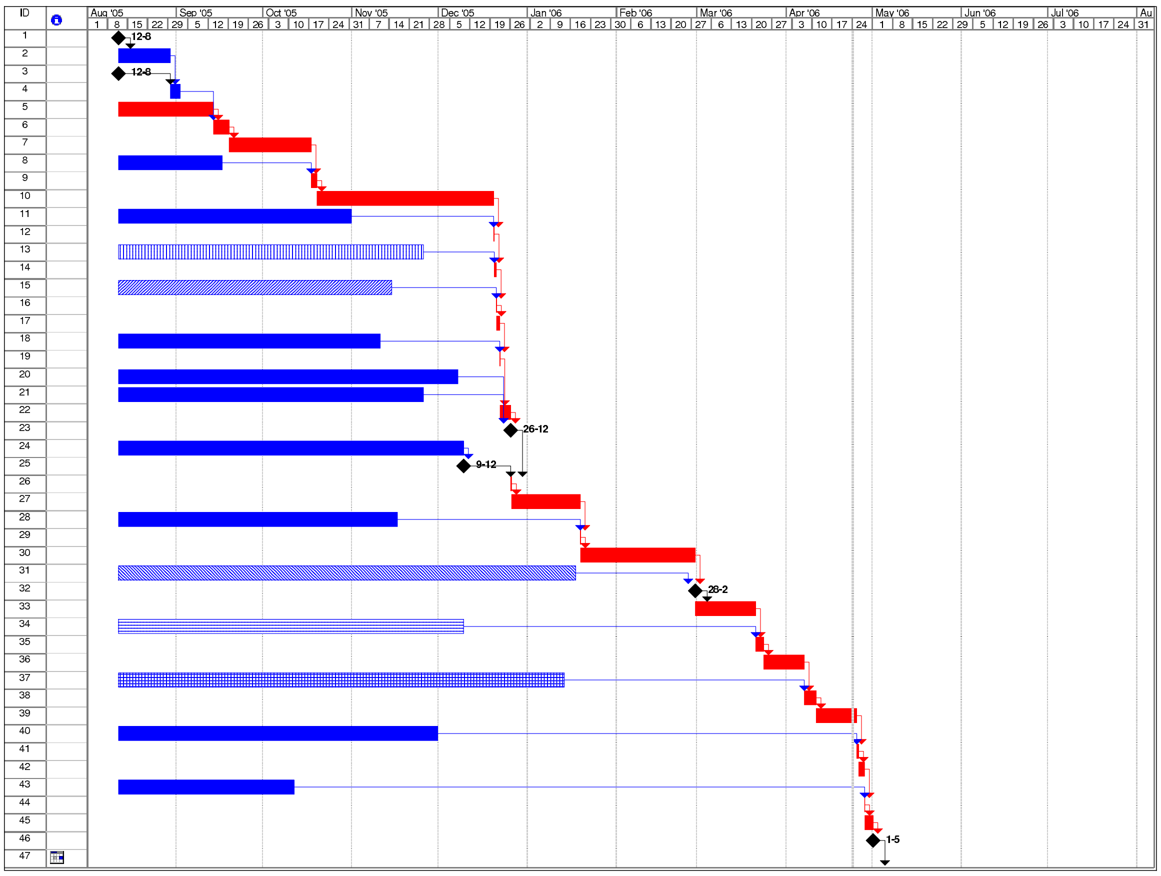


Figure 5.1: Gantt chart of solution 1 of the case

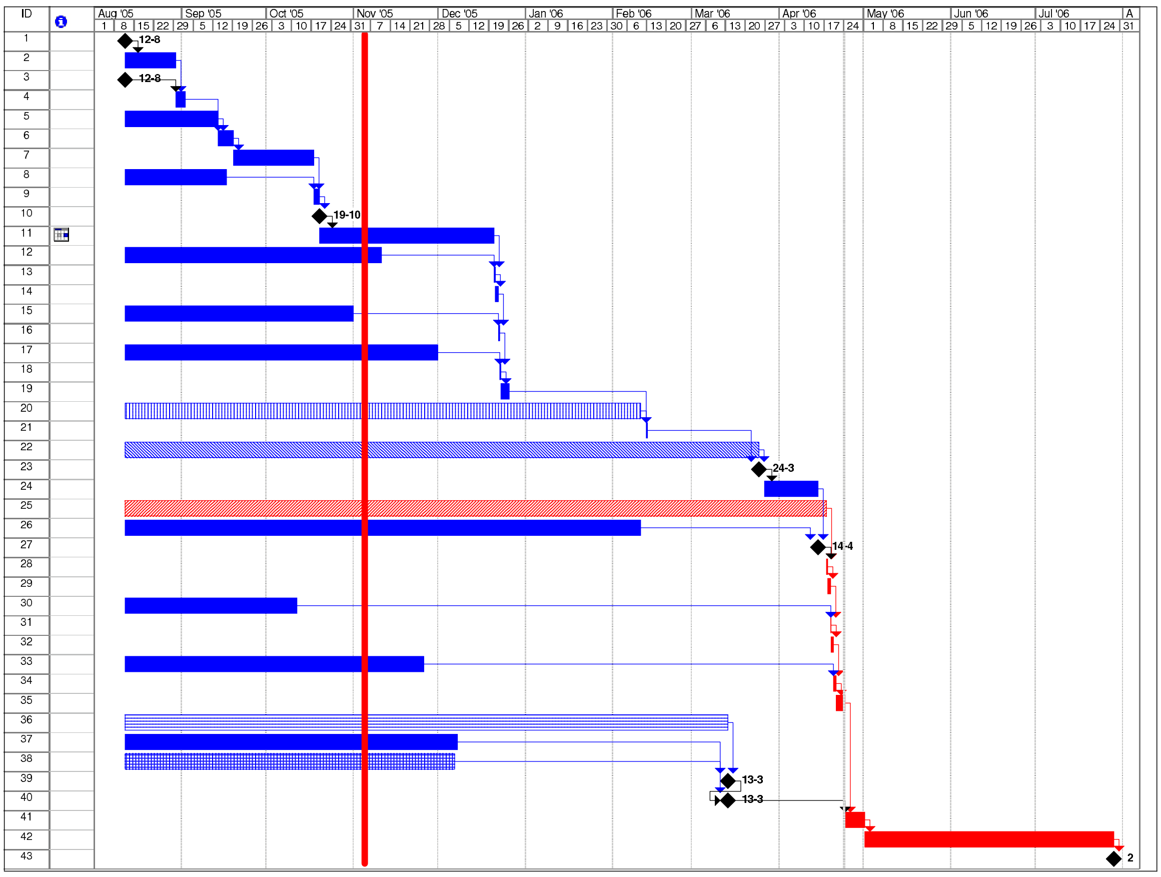


Figure 5.2: Gantt chart of solution 2 of the case

5.2 Situation 2

At a certain point in time, it appeared that the development times of 4 out of 17 modules were longer than planned and of 1 module shorter than planned. Furthermore, 1 module was removed from the system. Due to these large changes, a re-plan action was needed. A normal re-plan action would take some manual effort, however with the proposed method, only the model needs to be adapted and a new sequence can be calculated. The new plan is shown in Figure 6, where the red vertical line denotes the time the re-plan action was performed. All tasks that have a different arrival time are patterned. These patterns correspond to the same tasks as shown in Figure 5. The development tasks are numbered: 1, 3, 5, 8, 10, 12, 15, 17, 20, 22, 25, 26, 30, 33, 36, 37, 38, while the integration tasks are numbered: 4, 6, 9, 13, 16, 18, 21, 23, 27, 28, 31, 34, 39, 40, 41, and the test tasks are numbered: 2, 7, 11, 14, 19, 24, 29, 32, 35, 42. The critical path of this new plan is now formed by one of the late modules (task number 25). Therefore, all tests that used to be performed in the tasks with numbers 30, 33, 36 and 39 in the sequence shown in Figure 5, are now done at the end in the task with number 42. The total duration of the new plan is delayed almost 30% because these 4 modules arrive late.

Chapter 6

Conclusions

In this paper, we proposed a method to calculate an optimal integration sequence. The optimal integration sequence is the sequence with the shortest duration that develops, integrates and tests modules into a system, and therefore tries to perform the development, integration and test actions as much as possible in parallel. The method consists of defining a model which is able to describe an integration sequencing problem mathematically and an AND/OR algorithm which is able to determine the optimal solution.
The model consists of the modules or stubs/models to be integrated in the system and their development times, the interfaces that denote the possible assemblies of modules and their creation times, the tests that should be performed and the time it takes to perform them, for each test the modules that need to be present before the test can be performed and for each interface the modules it is cerated in between.
Furthermore, we introduce two heuristics for the algorithm to obtain a solution for large problems within reasonable time limits. The PT heuristic can be used best when dealing with a problem where many test and integration actions can be done in parallel. The ET heuristic can be used best in a situation where this is not the case.
We performed a case within the development of a first-of-a-kind lithographic machine to optimize the integration plan. With this case study, we learned that feasible and good solutions can be obtained using the model.
The benefits of this method are two-fold: 1) optimal integration plans can be calculated which are usually better than manually created integration plans, and 2) re-planning the integration plan when for example a module is delivered late takes less effort because only the model needs to be adjusted.
Within this method we assumed that alle tests should be applied exactly once. However, in previous work [13, 14] we have shown that not all tests must be applied, or that tests must be applied multiple times. In our future work, we will combine both the test sequencing method presented in previous work with the integration sequencing method such that optimal integration sequences with optimal test phases can be obtained.

Definitions and notations

In Table 3, the definitions and their descriptions used in this paper are shown.

Table 6.1: List of definitions

Definition	Description
D	Integration model: $(M, I, T, C^m, C^i, C^t, R^{im}, R^{t,})$
M	set of k modules.
I	set of l interfaces.
T	set of m tests.
C^m	the cost in time units of developing a module m.
C^i	the cost in time units of constructing an interface i.
C^t	the cost in time units of performing a test t.
R^{im}	the modules where interface i is constructed between.
R^{tm}	the essential assemblies of test t.
m, i, t	a single module, interface or test.
J, J^*	Cost and optimal cost of an integration sequencing problem.
$\mathcal{G}$, G, G^*	All solutions, a solution and the optimal solution of the integration sequencing problem.
X, x, x_{init}	Domain of system states, a single system state and the initial system state.
A_x, a	All possible cut-sets of a system state x, and a single cut-set.
J_x, J_a, J_{tot}	Cost of an OR node denoted with system state x and an AND node denoted with cut-set a, and total cost-to-go for a system state x.
I_x, T_x, T_r	All interfaces that are constructed in a system state x, all tests that can be performed and all tests that are performed in a system state x.
ET, PT	’Early time’ and ’Parallel time’ heuristics.
n	User-defined variable.
IG	Information Gain.
$\mathcal{H}$, H, H_{init}	Domain, instantiation, and initial instantiation of the function that returns the cost of solved OR nodes.

Algorithm

This section describes the algorithm that performs the AND/OR graph search. The AND/OR graph search is a recursive search over AND and OR nodes. Each OR node (except for the leave nodes) has one or more following AND nodes, depending on the number of possible cut-sets that are possible in that OR node. Each of those AND nodes has exactly two resulting OR nodes for each of the created sub-systems, and so on. This appendix gives a formal, based on recursion, functional style [15] description of the algorithm. A step-by-step description of the algorithm that relates to the functions defined, is shown in Figure 7.

To prevent double calculations of the same OR nodes we introduce $\mathcal{H} : X \rightarrow (\mathbb{R}^{+} \times \mathcal{P}(I)) \cup \{\perp\}$ which gives the optimal cost and the chosen cut-set for a solved OR node, or gives undefined for an unsolved OR node. The function H is an instantiation of $\mathcal{H}$.

To find the optimal integration cost J^{*} for an integration model $D = (M, I, T, C^{m}, C^{i}, C^{t}, R^{im}, R^{tm})$, the following expression can be used:

$$(J^{*}, H) = OR(x_{init}, A_{x}(x_{init}), H_{init}) \tag{6.1}$$

where $H_{init} : \mathcal{P}(M) \rightarrow \{\perp\}$ is the initial function that gives the cost of a solved OR node. $A_{x}(x_{init})$ are all cut-sets of the initial system state x_{init}, calculated using the algorithm presented in [11]. Here $x_{init} = M$, which denotes the initial state which is the complete integrated system. We only calculate the cut-sets for the initial system state. The cut-sets that are needed for the other system states (that are formed by disassembling the initial state) can be obtained by taking the initial set of cut-sets and then remove the cut-sets that do not split that system state into exactly two sub-assemblies. This prevents calculating the cut-sets for every system state which reduced computation time.

The resulting H can be used to construct the optimal solution G. This calculation gives the cost J^{*} of the optimal solution according to equation 2. For each OR node, all cut-sets are considered for the integration sequence except for the cut-sets that do not split the system into exactly two sub-systems, which is not allowed. The best cut-set per OR node is chosen based on the minimal integration and test cost per cut-set, starting from the last OR node. If more cut-sets have the same minimal cost, one of them is chosen.

The function $OR : X \times \mathcal{P}(\mathcal{P}(I)) \times \mathcal{H} \rightarrow \mathbb{R}^{*} \times \mathcal{H}$ calculates the cost of a system state x given the possible disassembly actions and the initial H. The cost of such an OR node is denoted by Equation 7. The OR function is defined as follows. If the state has already been solved (i.e. if $H(x)\ /= \perp$), the solution of that solved state is taken from H and returned. Otherwise, several options are possible. If x consists of one module, the node is a leaf node and the

Step-by-step algorithm

```
Input:    – System model $D$;
Output:   – The optimal solution tree $G$;
          – The cost of the solution;

Step 0: Initialize a graph $G$ consisting of the root node $x_{init} = M$, i.e., all modules are integrated,
        mark the node as unsolved.

Step 1: Repeat the following steps for the system state $x = x_{init}$ to construct an AND/OR graph
        until the root node is marked solved. Then exit with $J = J_x(x_{init})$ as expected test cost
        and the solution graph $G$ (these steps are performed by function $OR$).

    Step 1.0  If $x$ contains one element $m$ mark $x$ solved in G and assign the cost of the tests
              that must still be performed $T_r(x)$ and the development cost $C^m(m)$ to $x$, then stop.
              Else, continue with step 1.1
    Step 1.1  Determine the possible cut-sets $A_x(x)$ and perform for each cut-set $a$ in $A_x(x)$
              the following steps (these steps are performed by function $AND$):
        Step 1.1.0  Initialize a subgraph $G'$ consisting of root node $a$
        Step 1.1.0  Determine for $a$ the new sub-system OR nodes $x_1$ and $x_2$,
                    insert them in $G'$ and draw an edge from $a$ to both of them
        Step 1.1.1  If $x_1$ is not solved, mark $x_1$ unsolved and perform
                    steps 1.0 through 1.2 for $x$ replaced by $x_1$; do the same for $x_2$
                    (this is the recursion by function $OR$)
        Step 1.1.2  Determine for $a$ the cost of creating the interfaces in $a$,
                    the cost of tests that have to be performed $T_r$, and the maximal cost for $x_1$ or $x_2$,
                    and assign these cost to AND node $a$
    Step 1.2  Select the cut-set $a$ and the belonging subgraph $G'$ that has minimal cost.
              Mark $x$ solved and assign the cost $J_a(a, x)$ to $x$. Merge graph $G$ with subgraph $G'$,
              create an edge from node $x$ to the root node of $G'$ and exit.
```

Figure 6.1: Step-by-step algorithm description

resulting cost are the module development cost and the remaining test cost. If x consists of multiple modules, one or more cut-sets are available which are evaluated and compared with each other with the AND function:

$$OR(x, A_x, H) = \begin{cases} (H(x), H) & \text{if } H(x) \;/\!= \perp \\ (J', H(x/(J', \{\}))) & \text{if } H(x) = \perp \wedge |x| = 1 \\ (J'', H''(x/(J'', a''))) & \text{if } H(x) = \perp \wedge |x| > 1 \end{cases} \tag{6.2}$$

where:

- $J' = \sum_{m \in x} C^m(m) + \sum_{t \in T_x(x)} C^t(t)$
- $(J_a(a_i), H_i) =$
  $AND(x, rmv(A_x, a_i), a_i, H_{i-1}) + \sum_{t \in T_r(x)} C^t(t)$
  for $i = 1, \ldots, |A_x|$ (where $H_0 = H$), are the minimal test cost and updated H for each cut-set in A_x.
- $J'' = J(a'') = \min_{a \in A_x}(J_a(a))$, is the minimal cost of x, and a'' is the cut-set from A_x for which this holds.
- Function rmv removes all interfaces in the cut-set a_i from all cut-sets in A_x while ensuring that the resulting cut-sets still split the new system state into exactly two subsystems, and thus results in the new set of cut-sets ensuring that this set still satisfies Equation 3.

The function $AND : X \times \mathcal{P}(I) \times \mathcal{P}(\mathcal{P}(I)) \times \mathcal{H} \rightarrow \mathbb{R}^* \times \mathcal{H}$ takes a system state x as input and applies the disassembly action a to that system. It returns the cost made by that disassembly action and the path cost of the resulting subsystems.

$$AND(M, a, A_x, H) = \\ (\sum_{i \in a} C^i(i) + max(J', J''), H'') \tag{6.3}$$

Where:

- x_1 and x_2 are defined as the new system states resulting from applying cut-set a on system state x.
- $(J', H') = OR(x_1, A_x, H)$, calculates the cost of system state x_1 and the updated H'.
- $(J'', H'') = OR(x_2, A_x, H')$, calculates the cost of system state x_2 and the updated H''.

This appendix gives a formal, based on recursion, functional style [15] description of the AO^*_σ algorithm with inconclusive test and risk-based optimization.
The set of all possible OR nodes is defined as: $X^m = \mathcal{P}(\mathcal{P}(S)) \times \mathcal{P}(S \times \mathbb{R}^+) \times \mathcal{P}(T)$. This notation is adopted from the compact set notation as defined by Grunberg *et al.* [?].
$\mathcal{H}^m$ is redefined as a set of function: $X^m \rightarrow (\mathbb{R}^+ \times \{T\} \cup \{\top\}) \cup \{\perp\}$ and gives for a solved multiple-fault OR node the minimal test cost and the next test, or the remaining risk cost and a stop ($\top$) node, or gives undefined ($\perp$) for an unsolved OR node.
The multiple-fault algorithm consists of two functions, ORm and $ANDm$, that are defined for a system test problem D. Both functions are explained below.
To find the optimal expected test cost J' for D, the following expression is used:

$$(J', H) = ORm((\emptyset, SE_{init}, \emptyset), H_{init}, \emptyset) \tag{6.4}$$

where:

- $H_{init} : X^m \rightarrow \{\perp\}$ is the initial function that gives the cost of solved OR nodes.

- $SE_{init} = \{(s, 1.0) | s \in S\}$, is the excluded set of fault states which for all fault states in S the uncertainty p_u. The initial value of $p_u = 1.0$ denotes that the fault state has not been excluded, as explained before.

The resulting $H \in \mathcal{H}^m$ can be used to construct the optimal solution G. This calculation gives the cost J' of the optimal solution according to Equation ??.
Let function $ORm : X^m \times \mathcal{H}^m \times \mathcal{P}(T) \rightarrow \mathbb{R}^+ \times \mathcal{H}^m$ be a function that calculates the minimal expected test cost J' of an OR node and updates the set of solved OR nodes, given the OR node x, the current set of solved OR nodes H and the performed test set T_P. J' is:

- 0.0 if x is terminated,
- derived from H if x has already been solved,
- calculated with a fixed OR node if fault states are isolated,
- calculated with a diagnosed OR node if no fault states are isolated and further testing is of no use,
- the current risk cost if this cost is less than the cost of each test from T_C,
- calculated otherwise.

x is terminated if all fault states are excluded, x is solved if x is defined in H and fault states are isolated if they are within a candidate set of size 1, and T_P complies with R_{st}. The function is defined as follows:

$$
\begin{aligned}
&ORm(x, H, T_P) = \\
&\begin{cases}
(0.0, H) & \mathit{if}\ \{s | (s, r) \in x.1 \wedge r = 0.0\} = S \\
(H(x).0, H) & \mathit{if}\ \{s | (s, r) \in x.1 \wedge r = 0.0\}\ /= S \\
 & \quad \wedge H(x)\ /= \perp \\
(J_F, H_F) & \mathit{if}\ \{s | (s, r) \in x.1 \wedge r = 0.0\}\ /= S \\
 & \quad \wedge H(x) = \perp \\
 & \quad \wedge\ (\exists s : \{s\} \in x.0 : R_{st}(s) \in T_p) \\
(J_D, H_D) & \mathit{if}\ \{s | (s, r) \in x.1 \wedge r = 0.0\}\ /= S \\
 & \quad \wedge H(x) = \perp\ \wedge\ T_C = \emptyset \\
 & \quad \wedge\ (\nexists s : \{s\} \in x.0 : R_{st}(s) \in T_p) \\
(J_R, H(x/(J_R, \top))) & \mathit{if}\ \{s | (s, r) \in x.1 \wedge r = 0.0\}\ /= S \\
 & \quad \wedge H(x) = \perp\ \wedge\ T_C\ /= \emptyset \\
 & \quad \wedge\ (\nexists s : \{s\} \in x.0 : R_{st}(s) \in T_p) \\
 & \quad \wedge\ (\forall t : t \in T_C : C(t) > j_R) \\
(J, H_m(x/(J, t_j))) & \mathit{if}\ \{s | (s, r) \in x.1 \wedge r = 0.0\}\ /= S \\
 & \quad \wedge H(x) = \perp\ \wedge\ T_C\ /= \emptyset \\
 & \quad \wedge\ (\nexists s : \{s\} \in x.0 : R_{st}(s) \in T_p) \\
 & \quad \wedge\ (\exists t : t \in T_C : C(t) \leq j_R)
\end{cases}
\end{aligned}
\tag{6.5}
$$

Where:

- J_F is the minimal expected test cost of the fixed OR node and H_F is the updated H.
- J_D is the minimal expected test cost of the diagnosed OR node and H_D is the updated H.
- J_R is the cost of the current risk in the system.
- T_C is the candidate test set consisting of the tests of which the test signature is no subset of the excluded fault states of x (a certain pass), and none of the candidate sets of x is a subset of the test signature (a certain fail), together with tests that must be performed before fixing a candidate fault state. Also, non-repeatable tests in $x.2$ may not be performed again.

- J_T, is the minimal expected test cost of x, and t_t is the test from T_C for which this holds.
- $J = min(J_T, J_R)$, and t_j becomes t_t if J_T is the minimum or t_j becomes $\top$ if J_R is the minimum.

Let function $ANDm : X^m \times T \times \mathcal{H}^m \times \mathcal{P}(T) \rightarrow \mathbb{R}^+ \times \mathcal{H}^m$ be a function that determines the minimal expected test cost J of an AND node and updates H, given the OR node x, applied test t, the current H and the performed test set T_P in the same way as in the single-fault algorithm:

$$ANDm(x, t, H, T_P) = (p_p \cdot c_p + p_f \cdot c_f, H_f) \tag{6.6}$$

Where:

- c_p is the minimal expected test cost of the pass OR node.
- c_f is the minimal expected test cost of the failed OR node.

  H_f is the updated set of solved OR nodes.
- p_p is the pass probability of t calculated by dividing the sum of the fault probabilities of the suspected sets in the pass OR node by the sum of the fault probabilities of the suspected sets in x. The fault probability of a suspected set is calculated by multiplying the fault probabilities of the suspected fault states by the pass probabilities of the not-suspected, not-excluded fault states.
- $p_f = 1 - p_p$, is the fail probability of t.

Bibliography

[1] http://www.asml.com.

[2] R. D. Graig and S. P. Jaskiel, *Systematic Software Testing*. Artech House Publishers, 2002.

[3] B. Beizer, *Software testing techniques*. Van Norstrand Reinhold, 1990.

[4] V. L. Hanh, K. Akif, Y. L. Traon, and J.-M. Jézéquel, “Selecting an efficient oo integration testing strategy: An experimental comparison of actual strategies,” *Proceedings of ECOOP 2001*, pp. 381–401, 2001.

[5] L. S. H. de Mello and A. C. Sanderson, “Representations of mechanical assembly sequences,” *IEEE transactions on Robotics and Automation*, vol. 7, no. 2, pp. 211–227, April 1991.

[6] ——, “A correct and complete algorithm for the generation of mechanical assembly sequences,” *IEEE transactions on Robotics and Automation*, vol. 7, no. 2, pp. 228–240, April 1991.

[7] N. Boneschanscher, “Plan generation for flexible assembly systems,” Ph.D. dissertation, Delft University of Technology, 1993.

[8] N. Braspenning, J. van de Mortel-Fronczak, and J. Rooda, “A model-based integration and testing approach to reduce lead time in system development,” Eindhoven University of Technology, Systems Engineering Group, Technical report SE 420459, December 2005.

[9] R. Z. Lita Shon-Roy, Allan Wiesnoski, *Advanced Semiconductor Fabrication Handbook*, W. Philips, Ed. Integrated Circuit Engineering Corporation, 1998.

[10] A. W. A. Delchambre and P. Gaspart, “Knowledge based assembly by disassembly planning,” *Proceedings of the International Conference on Expert Systems in Engineering Applications*, October 1989.

[11] S. Tsukiyama, I. Shirakawa, and H. Ozaki, “An algorithm to enumerate all cutsets of a graph in linear time per cutset,” *Journal of the Association for Computing Machinery*, vol. 27, no. 4, pp. 619–632, October 1980.

[12] R. Johnson, “An information theory approach to diagnosis,” *Proceedings of the 6th Symposium on Reliability and Quality Control*, pp. 102–109, 1960.

[13] R. Boumen, I. S. M. de Jong, J. W. H. Vermunt, J. M. van de Mortel-Fronczak, and J. E. Rooda, “Test sequencing in complex manufacturing systems,” *Accepted for IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 2006.

[14] ——, “A risk-based stopping criterion for test sequencing,” Eindhoven University of Technology, Internal Report SE 420460, Submitted to IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 2006.

[15] R. Bird, *Introduction to Functional Programming using Haskell*, 2nd ed. Prentice Hall Press, 1998.

Biography

Roel Boumen received his M.Sc. degree in Mechanical Engineering from the Eindhoven University of Technology, the Netherlands, in 2004. During his work as a master student he worked in the field of supervisory machine control of lithographic machines. Since 2004 he is a Ph.D. student at the Eindhoven University of Technology. His research concerns test strategy within the TANGRAM project.

I.S.M. de Jong has a B.Sc. in Laboratory Informatics and Automation from Breda Polytechnic. He has been a software engineer in various companies in the USA and The Netherlands. Since 1996 he has worked with ASML in systems testing, integration, release and reliability projects. His specialization is in the field of test strategy. Since 2003 he is an active member in the TANGRAM project and a PhD student at the Eindhoven University of Technology.

J.M.G. Mestrom received his M.Sc. degree in Mechanical Engineering from the Eindhoven University of Technology, the Netherlands, in 2006. During his work as a master student he worked in the field of integration and testing of complex manufacturing systems. His research within the TANGRAM project concerned strategies and algorithms for integration and test sequencing.

J.M. van de Mortel-Fronczak graduated in computer science at the AGH University of Science and Technology of Cracow, Poland, in 1982. In 1993, she received the Ph.D. degree in computer science from the Eindhoven University of Technology, the Netherlands. Since 1997 she works as an assistant professor at the Department of Mechanical Engineering, Eindhoven University of Technology. Her research interests include specification, design, analysis and verification of supervisory machine control systems.

J.E. Rooda received the M.S. degree from Wageningen University of Agriculture Engineering and the Ph.D. degree from Twente University of Technology, The Netherlands. Since 1985 he is Professor of (Manufacturing) Systems Engineering at the Department of Mechanical Engineering of Eindhoven University of Technology, The Netherlands. His research fields of interest are modelling and analysis of manufacturing systems. His interest is especially in control of manufacturing lines and in supervisory control of manufacturing machines.