Detecting Deadlocks in Concurrent Systems

Lisbeth Fajstrup
Martin Raußen

BRICS Report Series RS-96-16

ISSN 0909-0878 May 1996

Detecting Deadlocks in Concurrent Systems

Lisbeth Fajstrup and Martin Raußen

BRICS[*]

Abstract

We use a geometric description for deadlocks occuring in scheduling problems for concurrent systems to construct a partial order and hence a directed graph, in which the local maxima correspond to deadlocks. Algorithms finding deadlocks are described and assessed.

Keywords: deadlock, partial order, search algorithm, concurrency, distributed systems.

1 Introduction – from discrete to continuous

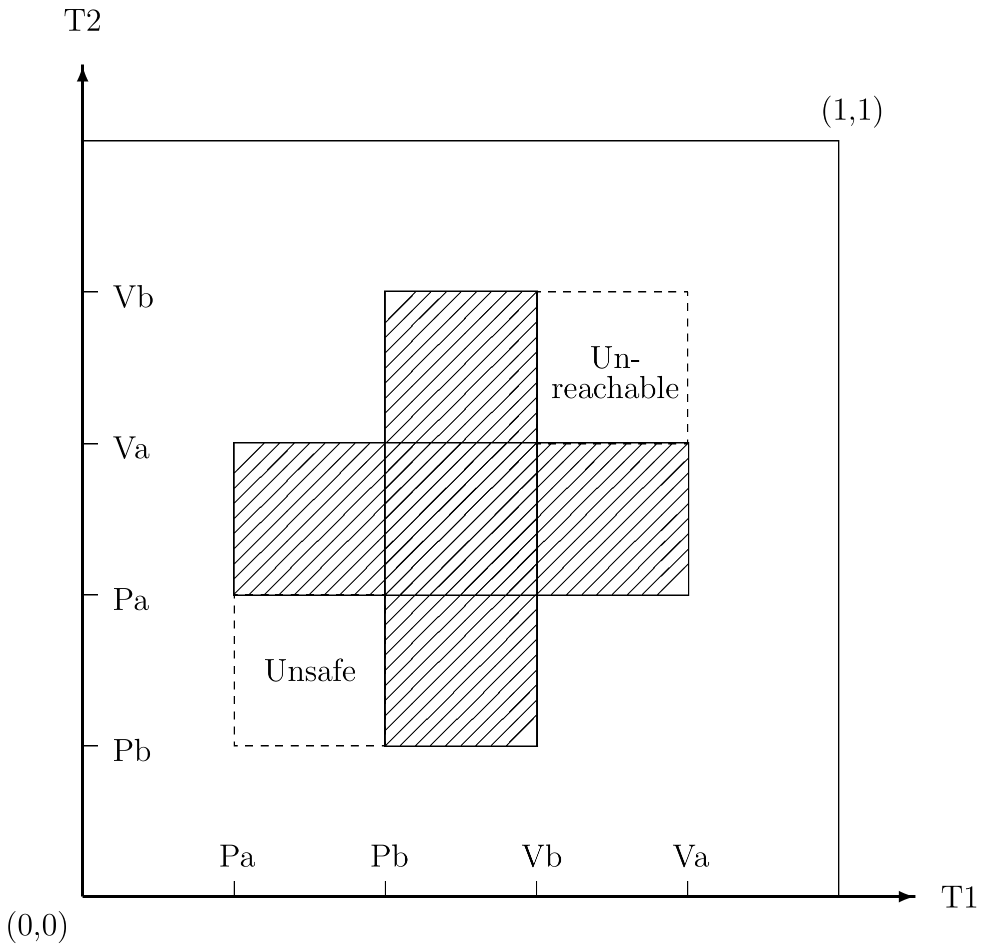
This paper deals with the detection of deadlocks motivated by applications in data engineering, e.g., scheduling in concurrent systems. A description of deadlocks in terms of the geometry of the progress graph had been given earlier by Carson and Reynolds [1], and we stick to their terminology.

The main idea in [1] is to model a *discrete* concurrency problem in a *continuous geometric* set-up: A system of n concurrent processes will be represented as a subset of Euclidean space $\mathbb{R}^n$. Each coordinate axis corresponds to one of the processes. The state of the system corresponds to a point in $\mathbb{R}^n$, whose i'th coordinate describes the state of the i'th processor. An execution is then a *continuous increasing path* within the subset from an initial state to a final state.

In recent years a number of people have used ideas from geometry and topology to study concurrency: First of all, using geometric models allows one to use spatial intuition; furthermore, the well-developped machinery from geometric and algebraic topology can serve as tools to prove properties of concurrent systems. A more detailed description of this point of view can be found in Gunawardena's paper [5] – including many more references – which contains a geometrical description of safety issues.

[*]Basic Research in Computer Science,
Centre of the Danish National Research Foundation.

Example 1.1 Consider a centralized database, which is being acted upon by a finite number of transitions. Following Dijkstra [2], we think of a transaction as a sequence of P and V actions known in advance – locking and releasing various records. We assume that each transaction starts at (local time) 0 and finishes at (time) 1; the P and V actions correspond to sequences of real numbers between 0 and 1, which reflect the order of the P's and V's. The initial state is $(0, \ldots, 0)$ and the final state is $(1, \ldots, 1)$. An example consisting of the two transactions $T_1 = P_a P_b V_b V_a$ and $T_2 = P_b P_a V_a V_b$ gives rise to the following two dimensional picture:



The shaded area represents states, which are not allowed in any execution path, since they correspond to mutual exclusion. Such states constitute the *forbidden area*. An *execution path* is a path from the initial state $(0, 0)$ to a final state $(1, 1)$ avoiding the forbidden area and increasing in each coordinate - time cannot run backwards.

In Ex. 1.1, the dashed square marked "Unsafe" represents an *unsafe area*: There is no execution path from any state in that area to the final state $(1,1)$. Actually it is also a *deadlock*. Likewise, there are no execution paths starting at the initial state $(0,0)$ entering the *unreachable area* marked "Unreachable". Concise definitions of these concepts will be given in §2.

Finding deadlocks and unsafe areas is hence the geometric problem of finding n-dimensional "corners" as the one in Ex. 1.1. Carson and Reynolds indicated in [1] an iterative procedure to achieve this. We give a much shorter treatment by translating the underlying geometry to properties of a *directed graph*. In particular, *deadlocks* correspond to *local maxima* in the associated partial order.

In general, the forbidden area may represent more complicated relationships between the processes like for instance general k-semaphores, where a shared object may be accessed by k, but not $k+1$ processes. This is reflected in the geometry of the forbidden area F, that has to be a union of higher dimensional rectangles or "boxes". The set-up allows us to describe, for instance, k-semaphores, asynchronous message passing and other distributed systems as long as there are no cycles.

Moreover, similar partially ordered sets can be defined and investigated in more general situations than those given by Cartesian progress graphs. By the same recipe, deadlocks can be found in concurrent systems with a variable number of processes involved. In that case, one has to consider partial orders on sets of "boxes" of variable dimensions. This allows the description and detection of deadlocks in the *Higher Dimensional Automata* of [6] and [7] (cf. [4] for an exhaustive treatment) as long as these have no cycles. Certainly, this latter restriction can be overcome by considering toral geometries (instead of rectangles) with a number of vector fields .

Furthermore, *non determinism* can be modelled by glueing partial orders together along the nodes where decisions have to be made.

The geometrical and combinatorial definitions and results from §2 and §3 are applied in §4 to describe and assess two *algorithms* for the detection of deadlocks.

This paper was inspired by but is not depending on the insight provided by tools from algebraic topology as used in [3].

Both authors participated in the workshop "New Connections between Mathematics and Computer Science" at the Newton Institute at Cambridge in November 1995. We thank the organisers for the opportunity to get new inspiration.

2 From continuous to discrete

Let I denote the unit interval, and $I^n = I_1 \times \cdots \times I_n$ the unit cube in n-space. We call a subset $R = [a_{01}, a_{11}] \times \cdots \times [a_{0n}, a_{1n}]$ an *n-rectangle*, and we consider a set $F = \bigcup_1^r R^i$ that is a finite union of n-rectangles $R^i = [a^i_{01}, a^i_{11}] \times \cdots \times [a^i_{0n}, a^i_{1n}]$. We think of F as the "forbidden region". Furthermore, suppose that $\mathbf{0} = (0, \ldots, 0) \notin$

F, and $\mathbf{1} = (1, \dots, 1) \notin F$.

Definition 2.1 1. A continuous path $\alpha : I \to I^n$ is called *increasing*, if *all* compositions $pr_i \circ \alpha : I \to I,\ 1 \le i \le n$, are increasing.

2. A point $x \in I^n \setminus F$ is called *admissible*, if there exists an increasing path $\alpha : I \to I^n \setminus F$ with $\alpha(0) = x$ and $\alpha(1) = \mathbf{1}$; and *unsafe* else.

3. Let $\mathcal{A} \subset I^n$ denote the *admissible region* containing all admissible points, and $\mathcal{U} \subset I^n$ the *unsafe region* containing all unsafe points.

In semaphore programs, the n-rectangles R^i characterize states where two transactions have accessed the same record, a situation which is *not* allowed in such programs. Such "mutual exclusion"-rectangles have the property that only two of the defining intervals are proper subintervals of the I_j. Furthermore, serial execution should always be possible, and hence F should not intersect the 1-skeleton of I^n consisting of all edges in the unit cube. These special features will *not* be used in the present section.

For $1 \le j \le n$, the set $\{a^i_{0j}, a^i_{1j} | 1 \le i \le r\} \subset I_j$ gives rise to a partition of I_j into at most $(2r+1)$ subintervals: $I_j = \bigcup I_{jk}$, with an obvious ordering $\le$ on the subintervals I_{jk}. The partition of intervals gives rise to a partition $\mathcal{R}$ of I^n into n-rectangles $I_{1k_1} \times \cdots \times I_{nk_n}$ with a partial ordering given by

$$I_{1k_1} \times \cdots \times I_{nk_n} \le I_{1k'_1} \times \cdots \times I_{nk'_n} \Leftrightarrow I_{jk_j} \le I_{jk'_j},\ 1 \le j \le n.$$

Remark 2.2 1. Admissibility with respect to the forbidden region F can be defined in terms of these n-rectangles: Two points in the same n-rectangle of the partition above are either both admissible or both unsafe points.

2. The rectangle $R_{\mathbf{1}}$ containing $\mathbf{1}$ is the *global maximum* for $\mathcal{R}$, the rectangle $R_{\mathbf{0}}$ containing $\mathbf{0}$ is the *global minimum*.

The partially ordered set $(\mathcal{R}, \le)$ can be interpreted as a *directed, acyclic graph*, denoted $(\mathcal{R}, \to)$: Two n-rectangles $R, R' \in \mathcal{R}$ are connected by an edge from R to R' – denoted $R \to R'$ – if $R \le R'$ and if R and R' share a face. R' is then called an *upper neighbour* of R, and R a *lower neighbour* of R'.

For any subset $\mathcal{R}' \subset \mathcal{R}$ we consider the *full* directed subgraph $(\mathcal{R}', \to)$. Particularly important is the subgraph $\mathcal{R}_{\bar{F}}$ consisting of all rectangles $R \subset I^n \setminus F$.

Definition 2.3 1. Let $\mathcal{R}' \subset \mathcal{R}$ be a subgraph. An element $R \in \mathcal{R}'$ is a *local maximum* if it has no upper neighbours in $\mathcal{R}'$. *Local minima* have no lower neighbours.

2. A rectangle $R \in \mathcal{R}_{\bar{F}}$ is called a *deadlock* if $R \neq R_{\mathbf{1}}$, and if R is a local maximum with respect to $\mathcal{R}_{\bar{F}}$.

3. An *unsafe* n-rectangle $R \in \mathcal{R}_{\bar{F}}$ is characterized by the fact, that *any* increasing path α starting at R hits a deadlock sooner or later [1].

Remark 2.4 1. An element $R \in \mathcal{R}_{\bar{F}}$ is a deadlock if $R \neq R_{\mathbf{1}}$, and if all its upper neighbours in $\mathcal{R}$ are contained in F. Deadlocks in $\mathcal{R}_{\bar{F}}$ are the maximal corners of the unsafe regions.

2. *Unreachable* rectangles can be defined similarly. Local minima ($\neq R_{\mathbf{0}}$) are their minimal corners.

In order to find the set $\mathcal{U}$ of all unsafe points – which is the union of *all* unsafe n-rectangles – apply the following

Algorithm 2.5 *1. Remove F from I^n giving rise to the directed graph $(\mathcal{R}_{\bar{F}}, \rightarrow)$.*

2. Find the set S_1 of all deadlock n-rectangles (local maxima) with respect to $\mathcal{R}_{\bar{F}}$. Let $F_1 = F \cup S_1$.

3. Let $\mathcal{R}_{\overline{F_1}}$ denote the full directed subgraph on the set of rectangles in $I^n \setminus F_1$, i.e., after removing S_1.

4. Find the set S_2 of all deadlock n-rectangles with respect to $\mathcal{R}_{F_1}$. Let $F_2 = F_1 \cup S_2$.

5. etc.

Notice that it is enough to search among the lower neighbours of elements in F in step 2, and that the only candidates for deadlocks in step 4 are the lower neighbours of elements of S_1. Since there are only *finitely many* rectangles, this process stops after a finite number of steps, ending with S_r and yielding the following result:

Theorem 2.6 *1. The unsafe region is determined by $\mathcal{U} = \bigcup_1^r S_i$.*

2. The set of admissible points is $\mathcal{A} = I^n \setminus (F \cup \mathcal{U})$. Moreover, any increasing path starting in $\mathcal{A}$ will eventually reach $\mathbf{1}$.

Proof: Only the last assertion has still to be shown. The set $\mathcal{A}$ is non-empty since it contains the global maximum $R_{\mathbf{1}}$. Now fix any increasing path starting from an arbitrary n-rectangle in $\mathcal{A}$. It will run through (finitely many) n-rectangles in $\mathcal{A}$ until it reaches a local maximum. This local maximum must be the global maximum $R_{\mathbf{1}}$, since $\mathcal{A}$ does not contain any deadlock.

□

Remark 2.7 We suggest that a better geometric understanding of the situation can lead to much quicker algorithms finding the unsafe regions: Instead of searching among lower neighbours one at a time, one would like to find their extreme points: It is not difficult to see that every unsafe region is again a union of n-rectangles with extent (i.e., maximal corner) the deadlock. We conjecture that the vertices (i.e, minimal corners) of those unsafe n-rectangles can be found by looking at certain *critical points* on the hyperplanes $\{x_i = a^i_{0j}\}$ defining the deadlock; see also [3]. Details – using Morse theory – will be worked out elsewhere.

3 Deadlocks in semaphore programs

In this section, we give an alternative description of the deadlocks discussed in §2 in the case of a *semaphore program.* Remember that deadlocks occur at some point of an execution, when *every* transaction demands access to a record, which is already locked by another transaction. Hence one should keep track of the set of records that are already accessed by some transaction at a given time. As in Ex. 1.1, we follow Dijkstra [2] in our definition of records and transactions:

Definition 3.1 Let S be a finite set, and let $\{T_1, T_2, \cdots, T_n\}$ be a set of words T_i on the alphabet $\{P_u, V_u | u \in S\}$. For an initial partial word $T_{i,j}, j > 0$, consisting of the first j symbols of T_i, and every $u \in S$, let $a(u,i,j) = \#\{P_u \in T_{i,j} | u \in S\} - \#\{V_u \in T_{i,j} | u \in S\}$. We require that

1. $a(u,i,j) \in \{0,1\}$ for any choice of u, i and j.
2. When j equals the length of the word T_i, i.e., j is maximal, then $a(u,i,j) = 0$ for any $u \in S$.

Furthermore we define $A(T_{i,j}) = \{u \in S | a(u,i,j) = 1\},\ 1 \le i \le n.$

In the language of records and transactions, S is the set of records, $\{T_1, T_2, \cdots, T_n\}$ is the set of transactions and $A(T_{i,j})$ is the set of records accessed by T_i after j steps.

Deadlocks can be characterized as follows:

Proposition 3.2 *Let $T_{1,j_1} \le T_1, T_{2,j_2} \le T_2, \cdots, T_{n,j_n} \le T_n$ be a collection of initial partial strings such that not all $T_{i,j_i} = T_i$. The system of transactions is in a deadlock at time $(j_1, j_2, ..., j_n)$ if and only if*

1. *$\forall i \neq k : A(T_{i,j_i}) \cap A(T_{k,j_k}) = \emptyset$;*
2. *For each i, either $T_{i,j_i} = T_i$, or there is a $k \neq i$ such that $A(T_{i,j_{i+1}}) \cap A(T_{k,j_k}) \neq \emptyset$.*

Proof: The characterization above is an immediate translation of the conditions for a deadlock found in §2: The condition that not all $T_{i,j_i} = T_i$ means, that the system is not in the final position $R_{\mathbf{1}}$. Furthermore condition 1) expresses that the state (n-rectangle) considered is not already in the forbidden region F, while condition 2) states that any of its upper neighbours is contained in F. This is exactly what characterizes a total deadlock.

□

4 Algorithms and complexity considerations

This final section describes two *algorithms* for finding deadlocks in general situations and gives estimates concerning their *complexity*. Certainly, one can do much better under special well-described circumstances.

The first algorithm

is based on the description of deadlocks in §2 and involves the following *data structures*: We represent the partial order $\mathcal{R}$ by the associated directed graph. We assume that a node (respresenting an n-rectangle) is equipped with *pointers* to its lower neighbours, i.e., its parents, and to its upper neighbours, i.e., its sons. Furthermore, every node is equipped with an *integer record* counting the number of sons, two *booleans* indicating whether the node is in F, and whether it is a leaf, and a *pointer* to a *list* of leaves. Then a rough sketch of the algorithm is as follows:

1. Mark all the n-rectangles which are in F and nil all pointers to and from these, i.e., discard their parents and sons.
2. The deadlocks are then all the leaves of the *resulting* graph except $R_{\mathbf{1}}$.

More specifically: Let $F = \bigcup_1^r R_i$. Then, for step 1 in the algorithm, go through all the $R_i \subset F$; if a node representing an n-rectangle $R \subset R_i$ is not yet marked in F, mark it, nil the pointers to its sons and nil all pointers to it. If one of the parents becomes a leaf by this operation, add it to a list representing "potential deadlocks", and set a pointer to its place in the list. If R itself was marked a leaf previously, then remove it from the list of potential deadlocks.

If the node has already been marked in F, do nothing. When this is done for all nodes in $R = \bigcup_1^r R_i$, the list of potential deadlocks contains only actual deadlocks.

We let the *volume* $Vol(S)$ of a set S of nodes (n-rectangles) in $\mathcal{R}$ be the number of its elements. For every element $R \in R_i$, one has to check, whether R had been marked earlier. Only if the answer is no, the $2n$ nil operations and possibly, a single addition to, resp. removal from, the list, has to be performed. This implies:

Proposition 4.1 *In a concurrent system of n transactions with a forbidden region $F = \bigcup_1^r R_i$, the deadlocks can be found by an algorithm of complexity $nVol(F) + \Sigma_1^r Vol(R_i)$.*

Remark 4.2 This estimate is worst, when the term $\Sigma_1^r Vol(R_i)$ dominates the term $nVol(F)$, i.e., when F consists of many large n-rectangles with large overlap. The absolute *worst case* occurs in the following situation of a two-phase locked semaphore program, where n transactions access k records: Suppose that each transaction wants access to each record, and that each transaction frees the records in the same order as it locks them. Then there are $N = (2k+1)^n$ states, and moreover $k\begin{pmatrix} n \\ 2 \end{pmatrix}$ n-rectangles R_i, which all have volume $k^2(2k+1)^{n-2}$. The volume of F is at most $(2k)^n$. Hence the *complexity* is n^2kN.

Examples of this kind have a high amount of global synchronization, which should be avoided in the programs involved. Hence one would expect a much better behaviour in the average situation. In fact, if $nVol(F)$ is the dominating part, the complexity is at most nN.

The second algorithm

below yields more favourable complexity estimates if the number r of n-rectangles $R_j \subset F$ modelling mutual exclusions is somehow restricted.Let again $F = \bigcup_1^r R_i$. Let $\mathcal{R}_F$, $\mathcal{R}_{R_i}$, denote the partial orders on the set of rectangles in F, resp. R_i.

Definition 4.3
1. Let $\mathcal{R}$ denote the directed graph on n-rectangles in I^n from above, and let $\mathcal{R}'$ be a full subgraph. Then the *lower boundary* $\partial_-\mathcal{R}'$ of $\mathcal{R}'$ is the set $\{R \in \mathcal{R}' | R \text{ has a lower neighbour outside } \mathcal{R}'\}$.
2. An n-rectangle R is called a *deadlock candidate* if it is contained in $\mathcal{R}_{\bar{F}}$ and if *all* of its upper neighbours are contained in at least one of the sets $\partial_-\mathcal{R}_{R_i}$.

Obviously, any deadlock is a deadlock candidate. The algorithm below consists of two steps:

1. Find (and mark) all deadlock candidates;
2. Find out, which of those are actually deadlocks.

For $F = \bigcup_1^r R_i$, let $r_j \leq r$ denote the number of n-rectangles R_j whose projection to the interval I_j is a proper subinterval, $1 \leq j \leq n$. An n-rectangle is a deadlock candidate, if its "extent", i.e., its maximal vertex, is contained in an intersection $\bigcap_1^n \{x_j = a_j^i\}$ of hyperplanes with $a_j^i = a_{0j}^i$ or $a_j^i = 1$. Hence, the number of deadlock candidates is given by $\prod_1^n (r_j+1)$. Since every n-rectangle in $\mathcal{R}$ can be labelled by its extent, every deadlock candidate is found in a single step. In order to find out whether a deadlock candidate actually is a deadlock, one has to check whether

1. $R \in \mathcal{R}_{\bar{F}}$;

2. Each of the n upper neighbours of R is contained in $\mathcal{R}_F$.

Each of these $n+1$ steps involves $4r$ operations, i.e., 4 inequality checks for each of the r n-rectangles constituting F. Multiplying these estimates, and comparing with the *number of states* $N = \prod_1^n (2r_j+1)$ of the system, we get the following complexity estimate:

Proposition 4.4 *In a concurrent system with a forbidden region $F = \bigcup_1^r R_i$, the deadlocks can be found by an algorithm of order $\frac{nr}{2^{n-2}}N$.*

More concrete estimates can be given in the case of a semaphore program:

Proposition 4.5 *For a semaphore program with n transactions and at most r mutual exclusions, the deadlocks can be found by an algorithm of order $2^{n+2}(\frac{r}{n})^n rn$. In particular, if there is a constant C such that $r \le Cn$, then the number of steps can be estimated by $2^{n+2}C^{n+1}n^2$. The algorithm is of order n^2 for $C \le \frac{1}{2}$.*

Proof: It was noted in the beginning of §2 for the model of a semaphore program, that the projections of one n-rectangle R_i to the coordinate intervals I_s will yield the whole interval I_s in $n-2$ cases, and a proper subinterval $[a_{0i}^s, a_{1i}^s]$ in 2 cases. Hence, one has to find the maximal value of $\prod_1^n (r_j+1)$ under the constraint $2r = \sum_1^n r_j$. This maximum occurs for $r_j = \frac{2r}{n}$ for all $1 \le j \le n$. As in the general case, the estimate $2^n(\frac{r}{n})^n$ has to be multiplied by $4nr$.

□

Remark 4.6

1. It would be interesting to know, whether it is reasonable to assume that r grows linearly as a function of n.

2. For several applications, a relative situation should be studied: Given a deadlockfree transaction system, to which a single transaction is added. How difficult is it to decide, whether the new system is deadlockfree, resp., where the new deadlocks can be found?

References

[1] S.D. Carson and P.F. Reynolds, *The geometry of semaphore programs*, ACM TOPLAS **9** (1987), no. 1, 25–53.

[2] E.W. Dijkstra, *Co-operating sequential processes*, Programming Languages (F. Genuys, ed.), Academic Press, New York, 1968, pp. 43–110.

[3] L. Fajstrup and M. Raußen, *Algebraic topology in scheduling problems*, Preprint R-96-2013, Institute of Electronic Systems, Aalborg University, 1996.

[4] E. Goubault, *The Geometry of Concurrency*, Ph.D. thesis, Ecole Normale Superieure, Paris, 1995.

[5] J. Gunawardena, *Homotopy and concurrency*, Bulletin of the EATCS **54** (1994), 184–193.

[6] V. Pratt, *Modeling concurrency with geometry*, Proc. of the 18th ACM Symposium on Principles of Programming Languages. (1991).

[7] R. van Glabbeek, *Bisimulation semantics for higher dimensional automata*, Tech. report, Stanford University, 1991.

Department of Mathematics, Institute for Electronic Systems, Aalborg University, Fredrik Bajers Vej 7E, DK 9220 Aalborg Ø
E-mail address: fajstrup@@iesd.auc.dk, raussen@@iesd.auc.dk
Fax: +45 98 15 81 29