A Heuristic Search Algorithm for Re-routing of On-Chip Networks in The Presence of Faulty Links and Switches

Nima Honarmand, Ali Shahabi and Zain Navabi
CAD Laboratory, School of ECE, University of Tehran, Tehran, IRAN
{nima, shahabi}@cad.ece.ut.ac.ir, navabi@ece.neu.edu

Abstract

The decreasing manufacturing yield of integrated circuits, as a result of rising complexity and decreased feature size, and the emergence of NoC-based design techniques, has necessitated the search for network reconfiguration techniques for reusing NoCs with faulty components. In this paper, we propose a new method to cope with the problem of faulty components in mesh-based on-chip networks. The method is based on using programmable routing tables in network switches. We propose a heuristic algorithm to search for a valid configuration for these routing tables when several physical faults occur in communication links, switch ports, routing tables and routing logics. The algorithm considerably reduces the required search effort as compared to the exhaustive search method.

1. Introduction

By the end of the decade, the 45-nm transistors operating below 0.7 volt will make it possible to put 4 billion transistors running at 15 GHz on a single chip [1]. System-on-chip (SoC) design methodology [2] has been proposed as a way to manage the increased complexity of these ICs. In this technique, pre-designed and pre-verified Intellectual Property (IP) cores are integrated on a single chip. Communication between the IP cores is one of the most important challenges in the SoC design methodology. Networks on Chip (NoC), [3] and [4], is an emerging paradigm which addresses this problem. A typical NoC consists of four major components: Processing Elements (PEs), Network Interface Units (NIUs), Switches and Physical Links. PEs do the actual processing while the others constitute the communication fabric. Several NoC architectures and their implementation details have been presented in [5] and [6].

With the emergence of nano-scale feature sizes, the produced ICs become more and more susceptible to manufacturing faults and the process yield decreases [1]. Because manufacturing faults tend to be local and affect a limited area of the die, it is possible (and very desirable) to find methods to make such faulty dies reusable. Such failures can occur in PEs and/or in communication fabric, *i.e.*, switches and communication links. One possible method to work around the permanent faults is to use fault tolerant architectures, [7] and [8]. Such techniques tend to introduce some redundancies into the circuit in order to cope with faulty elements and thus result in less-than-full hardware utilization. Reconfiguring the faulty IC to avoid using the faulty modules (PEs, switches or links) and get the work done using the remaining non-faulty ones is another approach, that is usually referred to as *degradability* [9]. The reconfigured circuit is likely to have a degraded performance, compared to the non-faulty one, because of the decreased number of available resources. Generally, reconfiguration-based methods imply a design and manufacturing flow including the following steps: First, at the design stage, special algorithms should be used which result in reconfigurable circuits. Then, after the IC was built, diagnosis techniques [10] should be used to detect the faulty components of the circuit. Then, based on the obtained fault pattern, a proper configuration, which bypasses the faulty elements, should be chosen and programmed into the circuit. Widely-used built-in test provisions such as JTAG or scan chains can be used for this purpose.

In this paper, we focus on static reconfiguration-based methods for recovering a faulty communication fabric in networks with mesh [5] topology. Mesh-based networks are appearing as a de facto standard in the NoC realm because of their regular and efficient layout and simple routing mechanism. In Section 2, we introduce a simple and efficient implementation of routing logic in Mesh-based network switches. In Section 3, we define the problem and provide the proposed reconfiguration algorithm. Section 4 provides the experimental results and Section 5 concludes the paper.

2. Proposed routing mechanism

In an NoC, switches are responsible for routing the packets between nodes. Each switch has a set of bidirectional ports through which it is connected to neighboring switches or PEs. It also contains a router to define a path between input and output ports, buffers to store intermediate data and an arbiter to grant access to a given port when multiple input requests arrive in parallel.

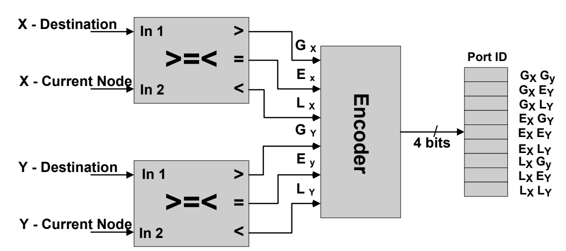


Figure 1. Mesh-based routing mechanism

One important aspect of the switch design is the implementation of the routing algorithm. Routing algorithm determines the path that a packet should traverse to reach its destination. Typically, the whole path is determined either at the source node (*source routing*) or on a node-by-node basis while the packet traverses the network (*distributed routing*) [11]. Also, the routing algorithm might be adaptive or deterministic [11]. In the adaptive routing, the path taken between a source and destination pair might vary depending on dynamic network parameters like link congestion and power consumption while in the deterministic routing the path remains the same. Simple and efficient hardware implementation of deterministic and distributed routing for mesh networks is likely to result in their acceptance as the de facto standard for routing in the NoC realm. Hence, in this work, we focus on these algorithms.

In NoC switches, the routing algorithm can be implemented as a hardwired module or as a Programmable Routing Table (*PRT*). In the former case, no post-manufacturing programming is required but the switch cannot be changed to cope with the physical failures. Thus, the chip will become unusable whenever some components become faulty unless fault-tolerance provisions have been made into the circuit. But, in the latter case, the switches can be reconfigured to bypass the faulty elements. This approach, inevitably, adds a PRT programming step to the manufacturing process.

The simplest way to implement a PRT is to use a lookup table with as many entries as the number of nodes in the network. We call this technique Per-Address Routing (*PAR*). The index of the table will be the destination address of a packet and each entry will contain the identifier of the proper output port for the given destination address. But PAR is not the only possible, or even the best, implementation of PRT. In fact, PAR suffers from two major drawbacks: (1) The size of the lookup table will grow linearly with the number of NoC nodes. Besides the area overhead, since table lookup should take place at least once for every received packet, the large size of the table and the resulting delay of lookup operation will decrease the throughput of the switch. (2) PAR is not amenable to network scaling and design reuse because the switch cannot be used in networks with more nodes than the number of lookup table entries, and thus redesign will be needed.

In mesh-based networks, the address of a node is a pair (x,y) which is the coordinates of the node in the mesh. Each mesh switch, in general, has five ports: one attached to the local PE (local port) and the other four to the neighboring switches (system ports). When a packet arrives, it should be delivered to the attached PE, through the local port, if it is destined for that node. Otherwise, one of the system ports (Left, Right, Up and Down) should be selected according to the destination address. A simple dimension routing algorithm, called XY [11], has been proposed for the mesh topology. In this algorithm, a packet is first routed in X direction (left or right) and then in the Y direction (up or down). Based on this algorithm, Fig. 1 shows the proposed routing mechanism. Two small comparators compare the x- and y-coordinates of the current node with those of the destination. The outputs of each comparator can assume three different one-hot coded states (G for greater, E for equal and L for less). Thus, we can have 9 different situations for the combination of comparator outputs. An encoder will generate a 4-bit signal indicating which of these 9 situations has occurred. The output of the encoder will be used to index a 9-entry, programmable lookup table. Each entry of the lookup table contains a port identifier to indicate one of the five ports that should be used. Unlike PAR-based routers, this routing hardware has a small and fixed structure that does not depend on the number of nodes in the network. Throughout this paper, we refer to this method as Mesh-Based Routing (*MBR*) because it is tailored to the mesh topologies.

3. Programming PRTs Under Failures

In an NoC, physical faults might occur in the processing elements, communication links and switches components, *i.e.*, ports, routing table and router logic (comparators and the encoder in Fig. 1). To provide degradability in the presence of faulty PEs, techniques like Virtual Binding [9] can be used. In this work, we focus on the degradability in presence of faulty communication fabric. The following text describes an algorithm that tries to find a proper set of MBR-based PRT configurations to compensate for the effects of link and switch failures. To increase the generality of the algorithm, we consider two unidirectional links between each two neighboring nodes and assume that faults might affect each of these links separately. Also, we assume that faults might affect each input and output port of a switch independently. In addition, Faults in the individual entries of the routing table are considered separately.

3.1. Structural vs. Routing Connectivity

The network topology could be considered as a graph with switches being its vertices and links being its edges. If this graph is connected, then the network will be *structurally connected*. We say that the network is *routing connected* if for every source and destination (*SRC*, *DST*) pair of nodes, a packet originated in *SRC* can be routed to reach *DST*. Whether this is possible or not depends on the routing table configuration of the nodes that the packet visits. Improper configurations might prevent the packet from reaching its destination. We use the term *valid configuration* to refer to configurations resulting in a routing connected network.

We call a network *routing connectable* if there is a set of PRT configurations to make the network routing-connected. It is possible for a network of MBR-based PRTs to be structurally connected but not routing-connectable. Figure 2(a) shows an example of such a network. In this figure, the crosses indicate broken links. No configuration can be found for the PRT of node B because when B has a packet destined for E, it should send it using BE link and when it has a packet destined for H it should not use BE link. Thus, there is no feasible port identifier for (E_x, G_y) entry of B's routing table.

One possible method to find a proper configuration is the exhaustive search. In this method we should consider all the possible network configurations and check each one to see whether it creates a routing connected network. This is not a feasible approach because, for example, in a small 4 by 4 network, there are 16 nodes, 8 PRT entries per node that might use the system ports and 4 possible ports per entry, resulting in 4^{8*16} or 2^{256} different configurations.

In this work, we propose a heuristic search algorithm to find a set of PRT configurations to make a faulty mesh-based communication network routing connected. The algorithm considers the faults in the communication links, the switch ports, the PRT entries and the routing logics. The algorithm uses the heuristic of imposing constraints on the possible values of PRT entries. The constraints are obtained through considering local effects of the failures. The emphasis is on the speed of the algorithm because typically it should run during IC testing process to find a distinct configuration for each faulty IC. To the best of our knowledge, no fast algorithm has been previously proposed for this purpose.

Because of the local nature of the constraints, the algorithm can not globally guarantee that every generated configuration is valid and thus a check should be conducted to ensure that a path exists between any possible source and destination pair. However, as the results indicate, the required number of final complete checks is in most cases very small, if the network is routing connectable.

3.2. Link Failures

When a packet arrives at a switch, either it is destined for the attached processing element or it should be routed using one of the four system ports. In what follows, we use L, R, D and U to indicate moving in left, right, down or up directions respectively. The decision of where to route the packet is based on the result of address comparison between current switch and the destination switch.

Consider a packet which is currently at a switch addressed (x_{cur}, y_{cur}) and is destined for switch (x_{dst}, y_{dst}). We define the distance of the switches as

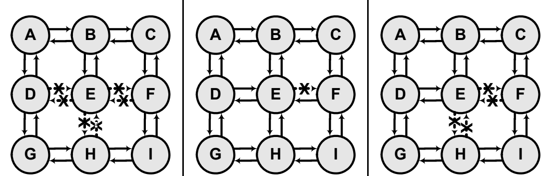
$$dist(cur, dst) = (| x_{dst} - x_{cur} | + | y_{dst} - y_{cur} |)$$

This distance is the minimum number of hops that the packet should traverse until it reaches its destination. If a routing algorithm routes a packet in such a way that its distance from the destination switch decreases with each move, the algorithm will be live-lock free. The live-lock refers to the situation in which a packet will not reach its destination, although it never gets blocked permanently [11]. A live-locked packet will visit a switch twice and this cannot happen with a decreasing sequence of distances. The conventional XY routing algorithm has this property and thus is live-lock free. We call every move that decreases the distance of packet from its destination a *positive* move. Otherwise, we call it a *negative* move. In mesh-based networks, such moves will inevitably increase the distance.

Sometimes there are multiple positive moves for a packet. For example, if the destination is both to left and above the current switch, taking either direction will be a positive move. In this case, the chosen direction depends on the routing table of the current switch. An MBR-based routing table has one entry for every possible combination of positive moves, as shown in Fig. 1. The contents of this table indicate the relative priority of moving in different directions. For example, if in the entry corresponding to positive moves in up and left directions, $(L_x\ L_y)$ in Fig. 1, the port identifier of the left port is given, then the priority of moving in the left direction is more than upward move. We use the "<" operator to show the priority of movements. In this case, we write U < L.

Reconfiguration Procedure. When a node becomes faulty, it will need some help from one of the neighboring nodes (helping node) to route the packets. Since the switch with a faulty link might be forced to perform a negative move, the distance may increase and if the helping node does not have its priorities properly set, it might return the packet to the original switch and cause live-lock. Hence, it would be necessary to put some constraints on the possible priority combinations that the helping node can use. As indicated before, these constraints are based on local considerations and can just guarantee routing-connectivity in absence of other failures in the network

An example with one faulty link. To demonstrate our technique, Fig. 2(b) shows a mesh structure with one broken link between nodes E and F. Since the EF link is faulty, Node E can select one of its two adjacent nodes in the other dimension (*i.e.*, B and H) as the helping node. In this algorithm we do not consider the neighbor in the same dimension as the faulty link (D, in this case) as a possible helping node, because this will necessitate non-local constraint assignments and will complicate the search process greatly. Suppose that we choose node B as the helping node. This will impose some restrictions on the routing table of B to prevent live-lock situations. Suppose node E has a packet destined for node F. Since the EF link is faulty, E will send the packet to B, instead. Now, node B has a packet that should move both right and down. If, in routing table of B, down moves have a higher priority than



(a) (b) (c)

Node	Hard Constraints	Soft Constraints
B	D < R	D < L
C	D < L	-
D	R < D	R < U
E	-	R < U, D < L
F	-	L < U, L < D
G	R < U	-
H	-	U < L, U < R

(d)

Figure 2. (a) a network which is not routing connectable using MBR-based routing tables (b) a network with one faulty link (c) a network with four faulty links (d) routing constraints in network of (c)

right moves, B will send the packet back to E and will cause a live-lock. Thus, for B, the priority of the down move should be less than that of the right move, or D < R. This is a Hard Constraint (HC) for B, *i.e.*, it must be met to have a live-lock free routing.

When we consider all the possible faulty links and extract all the required constraints, it might be the case that for a switch, the constraints are conflicting. For example, a switch might have both L < R and R < L constraints. Obviously, these are conflicting constraints and cannot be satisfied simultaneously. Suppose that in our example, we first choose B as the helping node of E and after considering other nodes, we get trapped in a conflicting situation. It might be the case that this conflict has happened because of choosing B as the helping node. Hence, we should now check the alternative choice and test the selection of H instead of B as the helping node. Thus our algorithm has a backtracking nature and whenever it encounters some conflicts, it backtracks to the last point where it had an alternative choice and tests that choice.

Moreover, we can use another constraint type, called Soft Constraint (SC) to reduce the average path lengths. Suppose that in Fig. 2(b) E is helped by node B to route in the right direction. It is better to force switch E to first route the packets using its non-faulty links (up, left and down) if it is a positive move. This means that switch E will use the helping node B to route when no other choice exists. Thus we consider the following SCs for node E: {R < U, R < D}. R < L is a useless constraint because moves to right and left are exclusive and cannot happen at the same time. As another example, Fig. 2(c) shows a network with four faulty links. Figure 2(d) shows the assigned hard and soft constraints for each node in the network of Fig. 2(c). The nodes missing in the table have no constraints. Figure 3(b) shows the constraints that will be considered for each faulty output link of node A in Fig. 3(a). For example, the first row indicates that if the upward link is faulty, one of the two constraints "L < U on node C" or "R < U on node E" will be considered. The choice among the two depends on the chosen helper node, *i.e.*, if the node C was chosen as the helper, L < U will be used, otherwise, were the node E chosen as the helper, R < U will be used.

3.3. Switch failures

In addition to the link failures, physical failures might occur in the ports, the routing logic and/or the routing table of the switches. Next, we will discuss the reconfiguration method in presence of such failures.

Port failures. We model port failures with link failures. Suppose that the right output port of switch E, in Fig. 2(b), becomes faulty. This condition implies that the corresponding link (*i.e.*, EF) cannot be used. Thus, we can assume that EF link is a faulty link. Since this imitates the effect of the faulty output port, we call the added failure a Virtual Link Failure (VLF). Now, consider a physical fault in the right input port of switch E. In this case switch F should be forced not to use its FE link. Consequently, a new VLF should be added to the failure set to indicate the invalidity of FE link.

Routing table failure. Each of the nine entries in a routing table can be affected by a physical defect. In these cases, we consider a proper combination of VLFs and routing constraints (on neighboring nodes) to prevent packets which might use this entry from reaching the affected node. These VLFs and constraints are again based on local considerations and can just guarantee routing-connectivity in absence of other failures in the network. Figure 3(c) summarizes the combination used for each faulty entry.

Router logic failure. When a fault occurs in the routing logic, the switch will not be able to route the received packets. Thus we should omit this node from the mesh structure and mark all of its outgoing and ingoing links as faulty, as shown in Fig. 3(c).

3.4. Reconfiguration Algorithm

Figure 4 shows the reconfiguration algorithm. CONFNETWORK() routine accepts three lists as its input. *prt_failure_set* containts the list of faulty routing table entries. *router_failuer_set* contains the switches with faulty routing logics and *link_failure_set* is the list of faulty links and switch ports. CONFNETWORK() first considers the constraints and VLFs imposed by faulty PRT entries and faulty routing logics (according to Fig. 3(c)) and then calls recursive CONFFAILURES() to configure the network for link failures. The recursive CONFFAILURES() routine has two inputs: the set of faulty links that have not been handled yet, and the constraints resulting from handling previous faults. It selects one faulty link from the list and, one by one, examines all possible helping nodes for the node affected by the faulty link. It repeats this process until there remains no faulty link in the list, i.e., all the faults are handled, or at some point, the set of constraints could not be satisfied. In the former case, it checks all the routing table configurations that satisfy the constraints. If a configuration results in a routing connected network, it will be returned as the result. Otherwise, the algorithm will backtrack to the point of the last choice. In the latter case, the algorithm will consider alternative helping nodes for the node affected by current faulty link. If there is no unchecked alternative, current invocation of CONFFAILURES() returns NULL to indicate failure. If the first invocation returns NULL then the algorithm has failed to find a livelock free configuration.

4. Experimental Result

To assess the proposed technique, we considered a 4 by 4 mesh network. This network has 16 nodes and 48 unidirectional links. Table 1 shows the results for 15 randomly generated faulty networks. Examples 1 to 6 have 4 to 16 unidirectional faulty links. Examples 7 to 15 include a combination of faults on links, PRT entries and routing logics. In TABLE I, the second to fourth columns give the number of faulty links, faulty PRT entries and faulty routing logics, respectively. The fifth column gives the number of network nodes that can still be used as a source or destination of the packets. Note that when all the incoming or outgoing links of a switch becomes faulty or faults happen in the PRT or the routing logic of a switch, that node can no longer be used as a packet source or destination.

Since the most time consuming part of the algorithm is the final routing-connectivity check, in TESTROUTING-CONNECTIVITY() routine, it is of utmost importance to reduce the number of times that this routine should be called. As indicated before, for the blind exhaustive search, this check should take place 2^{256} times for a 4 by 4 network. The sixth column of TABLE I shows the number of the performed checks that in all cases are very small compared to that of the exhaustive search. The last two columns provide the average path length and link load under the uniform traffic. Load of a particular link indicates the number of different (*SRC*, *DST*) pair of nodes whose packets should traverse that link The row indicated with XY in the table gives the performance parameters for a network with no faulty components in which all the routing tables are configured according to XY routing mechanism. The values in parentheses are normalized with regard to the corresponding values of non-faulty XY network. In case of networks with less usable nodes, the obtained average value might be less than that of the XY network because of the lighter traffic.

5. Conclusion

In this paper, we have considered the problem of faulty communication components (*i.e.*, faulty communication links, faulty ports in the switches, faulty router logics and faulty routing tables) in NoCs and have proposed a heuristic method for reconfiguring the routing mechanism in order to bypass the faulty elements. This method is based on using network switches with programmable routing tables. The experimental results show that the algorithm could, with a reasonable complexity, reconfigure the faulty networks to achieve acceptable performance parameters with regard to non-faulty networks using XY routing mechanism.

6. References

[1] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors*, World Semiconductor Council, Edition 2005, 2005.

[2] R. Saleh et al., "System-on-Chip: Reuse and Integration," *Proc. IEEE*, vol. 94, no. 6, pp 1050-1069, Jun 2006.

[3] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70-78, Jan. 2002.

[4] P.P. Pande et al., "Design, Synthesis, and Test of Network On Chips," *IEEE Des. Test. Comput*., vol 22. no. 5, pp. 404-413, Sept./Oct. 2005.

[5] S. Kumar et al., "A Network on Chip Architecture and Design Methodology," *in Proc. ISVLSI'02*, pp. 117-124, 2002.

[6] P. P. Pande et al., "Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures," *IEEE Trans. Comput.*, vol. 54, no. 8, pp. 1025-1040, Aug. 2002 .

[7] R. Marculescu, "Networks-on-Chip: The Quest for On-Chip Fault-Tolerant Communication", *in proc. ISVLSI'03*, pp. 8-12, Feb. 2003.

[8] M. Yang, T. Li, Y. Jiang, and Y. Yang, "Fault-Tolerant Routing Schemes in RDT(2,2,1)/a-Based Interconnection Network for Networks-on-Chip Designs," *in Proc. ISPAN'05*, pp. 1-6, Dec. 2005.

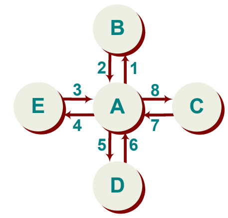
[9] N. Honarmand, A.Shahabi, H. Sohofi, M. Abbaspour, and Z. Navabi, "High Level Synthesis of Degradable ASICs Using Virtual Binding," *in Proc. VTS'07*, pp. 311-317, *May 2007.*

[10] C. Grecu et al., "On-line Fault Detection and Location for NoC interconnects, " *in Proc. IOLTS'06*, pp. 145-150, 2006.

[11] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks—An Engineering Approach*, Morgan Kaufmann, 2002.

Link and Port Failure	Details	Imposed Constraints		
Link and Port Failure	Up (1)	C: L < U	or	E: R < U
	Down (5)	C: L < D	or	E: R < D
	Left (4)	B: D < L	or	D: U < L
	Right (8)	B: D < R	or	D: U < R

(b) Constraints resulting from each faulty output link of Node A



(a) Nodes and Links used in Tables 3(b) and 3(c)

	Details	Imposed Constraints			Imposed Virtual Link Failures (VLF)
Routing Table Entry Failure	G_xG_y	B: D < R	and	E: R < D	-
	G_xE_y	B: D < R	and	D: U < R	3
	G_xL_y	D: U < R	and	E : R < U	-
	E_xG_y	C: L < D	and	E: R < D	2
	E_xE_y		-		-
	E_xL_y	C : L < U	and	E: R < U	6
	L_xG_y	B : D < L	and	C: L < D	-
	L_xE_y	B: D < L	and	D: U < L	7
	L_xL_y	C: L < U	and	D: U < L	-
Router Logic Failure	-		-		1, 2, 3, 4, 5, 6, 7, 8

(c) Constraints and Virtual Link Failures resulting from faulty PRT entries and routing logic

Figure 3. Constraints and Virtual Link Failures that should be considered for different faulty network elements

```
ConfNetwork(prt_failture_set, router_failuer_set,
link_failure_set)
begin
  current_constraints = the constraints imposed by
prt_failure_set;
  link_failure_set += VLFs imposed by prt_failure_set +
    VLFs imposed by router_failure_set;
  if CouldBeSatisfied(current_constraints) then
    return ConfFailures (link_failure_set, current_constraints);
  else
    return NULL;
  end if;
end
```

```
ConfFailures (link_failure_set, current_constraints)
begin
  if IsEmpty(failure_set) then
    for each configuration conf satisfying all the constraints
      TestRoutingConnectivity(conf)
      if the test succeeds then return conf; end if;
    end for;
    return NULL;
  else
    f = First(link_failure_set); n = node affected by f;
    for each possible helping node h for n
      new_constraints = current_constraints +
          {constraints from using h as the helper};
      if CouldBeSatisfied(new_constraints) then
        conf = ConfFailures (link_failure_set -f, new_constraints);
        if conf ≠ NULL then return conf; end if;
      end if;
    end for;
    return NULL;
  end if;
end
```

Figure 4. Reconfiguration algorithm for MBR-based routing tables

Table I. Experimental Results

	Faulty Links	Faulty Entries	Faulty Routers	Usable Nodes	Required Checks	Average Path Length	Average Link Load
XY	0	0	0	16	1	2.66	13.3
EX1	4	0	0	16	1	2.97(1.12)	16.18(1.22)
EX2	6	0	0	16	1	3.1(1.17)	17.62(1.32)
EX3	8	0	0	16	1	3.13(1.18)	18.8(1.41)
EX4	10	0	0	16	25	3.4(1.28)	21.52(1.62)
EX5	13	0	0	16	49	3.5(1.32)	24.14(1.82)
EX6	16	0	0	16	97	3.7(1.39)	28(2.11)
EX7	5	2	0	14	1	2.99(1.12)	13.95(1.05)
EX8	5	6	0	11	28	2.75(1.03)	9.44(0.71)
EX9	8	1	0	15	1	3.23(1.21)	17.84(1.34)
EX10	8	4	0	12	25	3.6(1.35)	12.8(0.96)
EX11	6	0	1	15	1	3.5(1.32)	20.44(1.54)
EX12	10	0	1	15	25	3.65(1.37)	23.27(1.75)
EX13	5	0	2	14	385	3.97(1.49)	14.46(1.09)
EX14	5	5	1	11	49	3.42(1.29)	14.46(1.09)
EX15	6	2	1	14	73	3.44(1.29)	18.41(1.38)