Evaluation of Static Variable Ordering Heuristics for MDD Construction*

Rolf Drechsler

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
email: drechsle@informatik.uni-bremen.de

Abstract

After designing of Multi-Valued Logic Networks (MVLNs), the resulting circuits have to be verified to guarantee functional correctness. The most promising technique to cope with increasing device sizes are formal methods. Ordered Multi-Valued Decision Diagrams (OMDDs) have been proposed for formal verification of MVLNs. But OMDDs are very sensitive to the chosen variable ordering and several ordering heuristics have been proposed in the past. The most promising with respect to OMDD size are dynamic variable ordering techniques, but these algorithms often cannot be applied in formal verification approaches due to their long runtimes. Alternatively, static variable ordering heuristics have been developed that determine an ordering from the circuit topology, but these heuristics often cannot guarantee good quality.

In this paper an evaluation technique is proposed that uses a pool of static variable ordering heuristics. Each heuristic is applied and the OMDD construction is started until a node or time limit is reached. Then the heuristic performed best so far is selected for the complete construction. The choice of the node and time limit allows to smoothly trade off runtime vs. quality. Experimental results are given to demonstrate the efficiency of the approach. The technique allows to save time and memory, since only promising orders are considered.

1 Introduction

Recently, there is a renewed interest in designing *Multi-Valued Logic Networks* (MVLNs). Several new synthesis techniques have been proposed (see e.g. [18, 15, 9]). Existing synthesis tools, like SIS from Berkeley, have been extended to also cope with multi-valued networks [12]. Like in the binary case, after the design phase, the circuits have to be verified. As one important aspect the check of two MVLNs for functional equivalence has to be carried out. One method to do this is verification based on ordered DDs as proposed in [11, 17] for two-valued circuits. An extension to OMDDs has been discussed in [6]. There several static heuristics known from OBDDs have been shown to be also applicable to OMDDs, since the motivation of tree-like circuits also holds for OMDDs (see [6] for the proof). Alternatively, clever dynamic variable ordering techniques have been considered (see e.g. [14]), but in fast equivalence checking tools used in formal verification, these algorithms are too time consuming [16].

In this paper an evaluation technique is proposed that is based on the idea of starting OMDD constructions for several static heuristics in parallel. After some limits, i.e. number of nodes or runtime, are reached, only the most promising heuristic applied so far is continued. The choice of the limits for nodes and runtime allows to trade off overall runtime and quality of the result. By this, in the starting phase of the algorithm some overhead can be observed, but the method prevents to generate overly large OMDDs that do not fit in the main memory or too time consuming operations. In [19] a similar approach has been considered for OBDDs. For OMDDs it seems to be even more difficult to find good orderings, as the results in [6] show. In [19] the evaluation was done on a functional level. Here, only structural properties are considered, i.e. it is evaluated how many gates of the circuit have been traversed. Experimental results are reported that demonstrate the quality of the approach.

The paper is structured as follows: In Section 2 MVLNs and OMDDs are defined. The basics of our verification procedure are described in Section 3. Sec-

*This work was supported by DFG grant Dr 287/9-1.

tion Section 4 addresses the evaluation procedure. In Section 5 experimental results are described. Finally the results are summarized.

2 Preliminaries

We provide an introduction to basic notions which are important for the understanding of this paper.

2.1 Multi-Valued Logic Networks

In general, a *Multi-Valued Logic Network* (MVLN) can be modeled as a directed acyclic graph $C = (V, E)$ with some additional properties: Each vertex $v \in V$ is labeled with the name of a basic cell or with the name of a *Primary Input* (PI) or *Primary Output* (PO). The collection of basic cells available is given by a fixed library. This library contains MIN-, MAX-, INV- and LITERAL-gates[1]. Of course, basic cells with arbitrary complexity, especially with an arbitrary number of inputs, are possible. There is an edge (u, v) in E from vertex u to v, iff an output pin of the cell associated to u is connected to an input pin of the cell associated to v, i.e. edges contain additional information to specify the pins of the source and sink node they are connected to. Vertices have exactly one incoming edge per input pin. Nodes labeled as PI (PO) have no incoming (outcoming) edges.

To simulate the circuit each PI may assume the values of a given ordered finite set $P = \{0, \ldots, k-1\}$ where k denotes the number of elements of the logic. The complement (INV-gate) of a signal x is defined as $\overline{x} = (k-1) - x$. A LITERAL-gate (a, b) $(a, b \in P, 0 \leq a \leq b < k)$ has one input and one output[2]. For a given input x the behavior of such a gate is defined by:

$$f(x) = \begin{cases} k-1 & : \quad a \leq x \leq b \\ 0 & : \quad otherwise \end{cases}$$

2.2 Multi-Valued Decision Diagrams

As well-known each Boolean function $f : \mathbf{B}^n \rightarrow \mathbf{B}$ can be represented by an *Ordered Binary Decision Diagram* (OBDD) [4], i.e. a directed acyclic graph where a Shannon decomposition is carried out in each node.

Obviously, OBDDs can be extended to represent functions $f : \mathbf{B}^n \rightarrow \{0, .., k-1\}$ and the resulting graphs are denoted as *Multi-Terminal BDDs* (MTBDDs). The operations on MTBDDs can be carried out as efficiently as in the case of two terminals [5].

It is straightforward to extend MTBDDs to *Multi-Valued Decision Diagrams* (MDDs) [21] representing functions $f : \{0, .., k-1\}^n \rightarrow \{0, .., k-1\}$. For this each internal node has k outgoing edges[3]. In [21] it has been shown that the efficient operations known for BDDs can also be carried out on MDDs using a *case*-operator instead of the *ite*-operator [1].

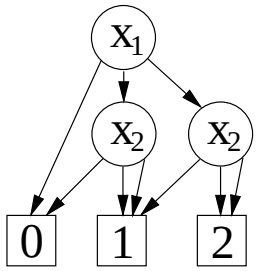

Figure 1. Reduced OMDD

A DD is called *ordered* if each variable is encountered at most once on each path from the root to a terminal and if the variables are encountered in the same order on all such paths. A DD is called *reduced* if it does not contain vertices either with isomorphic sub-graphs or with all successors pointing to the same node.

In the following we only consider reduced, ordered MDDs, i.e. reduced OMDDs.

Example 1 Figure 1 shows an OMDD of the two-variable three-valued function f given by the following truth-vector:

$$\mathcal{F} = [000011122]$$

3 Verification of MVLNs

For the equivalence check of two circuits based on DDs two main problems have to be solved that are considered in this section, i.e. the construction of an OMDD from a given circuit description and the check for isomorphism of the OMDDs.

3.1 Construction of OMDDs

We assume that a MVLN is given by a directed acyclic graph as described in Subsection 2.1. First, terminal nodes for the k constant functions are created. For each PI of the MVLN a variable in the OMDD is created, where the i-th outgoing edge points to the terminal node labeled i $(i \in \{0, .., k-1\})$.

Then the gates of the MVLN are visited in topological order and the corresponding OMDD operation is carried out. The topological order guarantees that

[1] In the binary case MIN- and MAX-gates correspond to AND- and OR-gates, respectively.

[2] These LITERAL-gates are also called *window literals*.

[3] In our application we restrict ourselves w.l.o.g. to the case that all variables are defined over the same set of values.

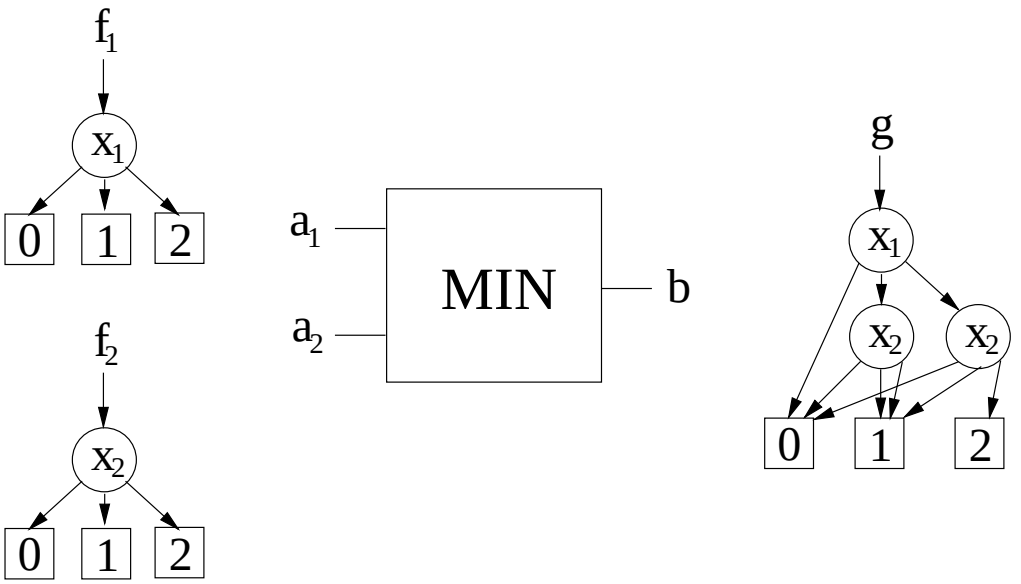

Figure 2. Symbolic simulation for MIN-gate

all inputs of a gate are known before it is evaluated. By this method, at the end OMDDs for the POs are created.

Example 2 In Figure 2 a simple example for a three-valued simulation for a MIN-gate is shown. The input a_1 (a_2) corresponds to the OMDD f_1 (f_2). The output of the gate b corresponds to the function that is represented by the OMDD g.

3.2 Verification of Equivalence

To verify that two MVLNs M_1 and M_2 realize the same function we only have to construct the OMDDs for both MVLNs as described above and then compare the OMDDs for equivalence. Since OMDDs (with the same ordering) are a canonical representation this can be done efficiently. With the methods described in [1, 21] this can even be carried out in constant time, since hash-tables are used for the implementation of the package.

4 Evaluation of Variable Ordering Candidates

From [6] it especially follows that *depth-first-search* orders (that first consider the larger circuit) create good OMDDs for tree-like circuits, i.e. the size of the OMDD remains polynomial for constant k.

The fact that DDs remain small for tree-like circuits was also the motivation for many OBDD heuristics that have been developed in the past (see e.g. [17, 10]). We implemented these heuristics that were originally designed for the binary case and saw that they work also well for MVLNs (see also [6]).

It is assumed that a set of static variable ordering heuristics is available. In the following six heuristics are used:

Initial (INI): Initial ordering as given in the benchmark description

Inverse Initial (INV): Inverse ordering

Topological (TOP): A topological sorting of the circuit

Dependent Count (DEP): Counting the number of outputs this input influences

Fanin (FAN): Similar to [17]

Interleaving (INT): Similar to [10]

Usually, INT gives the best results on average, but sometimes the heuristic totally fails. This can be seen by the following small examples for OBDDs, i.e. OMDDs with $k = 2$:

Example 3 *For benchmark* cs01423 *INT needs more than 80.000 nodes, while DEP needs less than 30.000 for the complete construction. On the contrary, for benchmark* cs05378 *INT can build the graph within 50.000 nodes, while DEP does not terminate within a 250.000 node limit.*

For this, it makes sense to also consider alternatives when selecting the variable ordering. When the user relies on one heuristic only, a lot of time and memory might be wasted.

First, the algorithms starts to build OMDDs for all orderings until a given limit is reached. In the following we consider two types of limits:

1. number of nodes and

2. runtime

These limits can be chosen fixed, or dependent on the problem size. For our experiments a dynamic measure dependent on the number of inputs of the circuit has been used:

$$c \cdot \textit{number of inputs}$$

Here, c is a problem specific constant that can be chosen by the user. The constant allows to trade off runtime vs. quality. If c is chosen very small, the heuristic is chosen very early, what saves runtime. But, if it is chosen too early, the "wrong" heuristic might be selected. Experimentally, the following numbers have been determined that are used in our experiments in the next section:

$$\textit{time limit} = \textit{number of inputs} * 0.05 \textit{ CPU seconds}$$

$$\textit{node limit} = \textit{number of inputs} * 500$$

5 Experimental Results

For the experiments the OMDD package from [6] has been used. It is parameterized in k and has been implemented in C++. The methods discussed in this paper have been applied. For the experimental results the benchmarks from [3] and [2] have been used analogously to [8, 6][4].

Remark 1 *The numbers of nodes used during the OMDD construction might slightly vary compared to [6], since improved traversal techniques from [7] are used.*

[4]Since in case of multi-valued applications no standard benchmark set of large circuits is available we used some of the ISCAS85 [3] and the combinational parts of the ISCAS89 [2] benchmarks. We interpreted the benchmarks as k-valued circuits by transforming AND-gates into MIN-gates and OR-gates into MAX-gates. XOR-gates contained in these circuits are interpreted as AND/OR-realizations. (Notice that this interpretation differs from the interpretation if each XOR-gate is substituted by the 4-NAND-equivalence.) In doing so the resulting circuits do not contain LITERAL-gates. Therefore the strength of our results is limited. But we expect that they describe a trend which is also valid for multi-valued circuits containing LITERAL-gates.

Table 1. Benchmarks

name	*in*	*out*	*signals*	*gates*
c0017	5	3	11	13
c0095	5	7	32	39
s00027	7	4	17	21
s00208	18	9	122	131
s00298	17	20	136	156
s00344	24	26	184	210
s00400	24	27	186	213
s00444	24	27	205	232
s00510	25	13	236	249
s00641	54	43	433	476
s00713	54	42	447	489
s00820	23	24	312	336
s00832	23	24	310	334
s00838	66	33	512	545
s00953	45	52	440	492
s01238	32	32	540	572
s01423	91	79	748	829
s01488	14	25	667	692
s01494	14	25	661	686
s05378	214	228	2993	3221

To give an impression of the size of the considered circuits some information on the benchmarks is provided in Table 1. *name* is the name of the benchmark. *in* (*out*, *signals*, *gates*) denotes the number of PIs (POs, signals, gates) of the corresponding benchmark. All measurements were performed on a *SUN ULTRA 10*.

A node limit of 250.000 nodes and a limit of 3.600 CPU seconds was used.

In a first series of experiments we compare the number of nodes needed for the representation of the outputs of small MVLN for different values k, i.e. $k = 2, 3, 4, 5$. The results are given in Table 2. For each benchmark the results in the first and second row are determined using the initial variable ordering (INI) and *interleaving* (INT) [10], respectively. The third row is determined by the evaluation technique (EVAL) described above. *PO* denotes the number of nodes that is used for the representation of the POs and *max* denotes the total number of different nodes needed during the whole construction (also called the *peak size*). As has already been observed in [6] for many MVLNs the initial variable ordering gives very good results, but this mainly holds for the smaller examples, where the MDD can be constructed independent of the chosen method. But there also exist cases, where the new technique clearly outperforms the initial ordering. In case of $k = 5$ and benchmark *s00344* the maximal number of nodes needed is reduced by more than a factor

Table 2. Smaller benchmarks

name	method	$k=2$		$k=3$		$k=4$		$k=5$	
		PO	*max*	*PO*	*max*	*PO*	*max*	*PO*	*max*
c0017	INI	11	23	24	57	42	106	65	164
	INT	9	28	19	64	32	110	48	166
	EVAL	15	7	17	34	28	53	48	89
c0095	INI	22	78	43	190	67	305	99	460
	INT	20	84	55	204	81	312	126	469
	EVAL	22	78	43	190	67	305	99	460
s00027	INI	27	41	62	102	112	194	177	323
	INT	16	33	36	84	63	157	97	254
	EVAL	19	25	43	61	76	112	118	180
s00208	INI	77	346	864	1580	3729	5161	19803	22778
	INT	78	360	1077	1920	4467	6557	30310	35813
	EVAL	77	346	864	1580	3729	5161	19803	22778
s00298	INI	126	388	327	1099	642	2180	1147	3944
	INT	111	367	299	980	659	2085	1183	3739
	EVAL	126	358	299	980	659	2085	1183	3739
s00344	INI	258	1340	712	5350	1705	14847	3461	35408
	INT	162	708	324	2004	625	4222	955	7618
	EVAL	265	577	732	1938	631	4218	965	7610
s00400	INI	235	1213	622	3465	1243	7006	2302	12624
	INT	174	916	474	2465	873	4607	1683	8045
	EVAL	195	563	577	1507	1188	2952	2366	5614
s00444	INI	187	718	462	1836	884	3395	1660	6171
	INT	181	1116	491	3038	916	5892	1764	10399
	EVAL	194	635	532	1720	1070	3367	1660	6171
s00510	INI	183	967	830	3630	2749	10368	7697	26442
	INT	204	1077	904	4149	2957	12078	8060	30868
	EVAL	183	967	830	3630	2749	10368	7697	26442
$\sum$	INI	1126	5114	3946	17309	11173	43462	36411	108314
$\sum$	INT	**955**	4689	**3679**	14908	10673	36020	44226	97371
$\sum$	EVAL	1096	**3556**	3937	**11640**	**10197**	**28621**	**33939**	**73083**

of four[5]. Even though the limits were often reached for larger k, the choice of the heuristic was never worse than *interleaving*. In the last three rows the sums for all three approaches are given. The best sum for each choice of k is given in **bold**. As can be seen, the peak size is reduced in all cases. For larger k the maximal number of nodes **and** the nodes for the POs can be reduced significantly, i.e. the peak size is 25% on average.

In a second series of experiments larger benchmarks are studied. For some of these, the OMDDs cannot be constructed for larger k. Again, in almost all cases the evaluation approach gives better results than interleaving alone. In several cases the memory consumption can be reduced significantly, i.e. up to a factor of two (see *cs01238*). Notice that for the larger benchmarks not all heuristics succeed in building the OMDD: the initial variable ordering fails for *cs05378* for $k = 2$.

Finally, we briefly show the influence of the constant to determine *time_limit* and *node_limit*. Consider the smallest circuit in the benchmark set, i.e. *c0017*. The results for all variable ordering heuristics used are given in Table 4. If the constant for *node_limit* is chosen as 5, the heuristic selected is FAN, while a value greater or equal to 10 returns the optimal choice DEP. Similar observations hold for *time_limit*. The choice of the constants showed a good trade-off between runtime and memory needed for the evaluation phase.

6 Conclusions

A method has been proposed to select one heuristic out of a pool of candidates based on some informa-

[5] All tie-breaking was done to optimize the maximal number of nodes and not the output size.

tion derived during the construction phase. Limits on the runtime and the number of nodes allow to trade off runtime vs. quality of the construction process. Experiments have shown that this technique gives very robust results while keeping the number of nodes needed during OMDD construction small. Significant reductions in the peak size can be observed, i.e. in some cases more than a factor of four.

It is focus of current work to improve the netlist traversal algorithms to further reduce the memory peak size. As has been observed in [7] in the case of larger k the decision procedures become more complex than in the binary case. More studies are needed to get a better understanding of the relation between the technique proposed in this paper and the ordering of network traversal.

As pointed out by one of the reviewers, this work can also be seen in the context of sampling [20, 13]. Future work, can study the relation between the two approaches and how sampling can be integrated in the technique presented in this paper.

References

[1] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[2] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Int'l Symp. Circ. and Systems*, pages 1929–1934, 1989.

[3] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational circuits and a target translator in fortran. In *Int'l Symp. Circ. and Systems, Special Sess. on ATPG and Fault Simulation*, pages 663–698, 1985.

[4] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[5] E. Clarke, M. Fujita, P. McGeer, K.L. McMillan, J. Yang, and X. Zhao. Multi terminal binary decision diagrams: An efficient data structure for matrix representation. In *Int'l Workshop on Logic Synth.*, pages P6a:1–15, 1993.

[6] R. Drechsler. Verification of multi-valued logic networks. *Multiple Valued Logic - An International Journal*, 3(1):77–88, 1998.

[7] R. Drechsler, A. Hett, and B. Becker. Symbolic simulation using decision diagrams. *Electronic Letters*, 33(8):665–667, 1997.

[8] R. Drechsler, R. Krieger, and B. Becker. Random pattern fault simulation in multi-valued circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 98–103, 1995.

[9] E. Dubrova, Y. Jiang, and R. Brayton. Minimization of multiple-valued functions in post algebra. In *Int'l Workshop on Logic Synth.*, 2001.

[10] H. Fujii, G. Ootomo, and C. Hori. Interleaving based variable ordering methods for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 38–41, 1993.

[11] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvements of Boolean comparison method based on binary decision diagrams. In *Int'l Conf. on CAD*, pages 2–5, 1988.

[12] M. Gao, J. Jiang, Y. Jiang, Y. Li, S. Sinha, and R. Brayton. MVSIS. In *Int'l Workshop on Logic Synth.*, 2001.

[13] J. Jain, W. Adams, and M. Fujita. Sampling schemes for computing OBDD variable orderings. In *Int'l Conf. on CAD*, pages 631–638, 1998.

[14] D. Janković, W. Günther, and R. Drechsler. Lower bound sifting for MDDs. In *Int'l Symp. on Multi-Valued Logic*, pages 193–198, 2000.

[15] J. Jiang, Y. Jiang, and R. Brayton. An implicit method for multi-valued network encoding. In *Int'l Workshop on Logic Synth.*, 2001.

[16] A. Kuehlmann, M. Ganai, and V. Paruthi. Circuit-based Boolean reasoning. In *Design Automation Conf.*, pages 232–237, 2001.

[17] S. Malik, A.R. Wang, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Int'l Conf. on CAD*, pages 6–9, 1988.

[18] A. Mischenko, B. Steinbach, and M. Perkowski. Bi-decomposition of multi-valued relations. In *Int'l Workshop on Logic Synth.*, 2001.

[19] D.E. Ross, K.M. Butler, R. Kapur, and M.R. Mercer. Fast functional evaluation of candidate OBDD variable ordering. In *European Conf. on Design Automation*, pages 4–9, 1991.

[20] A. Slobodová and C. Meinel. Sample method for minimization of OBDD. In *Int'l Workshop on Logic Synth.*, pages 311–316, 1998.

[21] A. Srinivasan, T. Kam, S. Malik, and R.E. Brayton. Algorithms for discrete function manipulation. In *Int'l Conf. on CAD*, pages 92–95, 1990.

Table 3. Larger examples

name	method	$k=2$		$k=3$		$k=4$		$k=5$	
		PO	*max*	*PO*	*max*	*PO*	*max*	*PO*	*max*
s00641	INI	1992	4918	22566	57562	-	-	-	-
	INT	830	2847	4040	14051	-	-	-	-
	EVAL	703	1689	4040	14051	-	-	-	-
s00713	INI	1992	5026	22491	85299	-	-	-	-
	INT	830	2965	4050	17936	-	-	-	-
	EVAL	683	1663	4050	17936	-	-	-	-
s00820	INI	321	1457	1238	5042	3420	12470	7782	26540
	INT	319	1594	1217	5759	3390	15324	7981	35279
	EVAL	321	1457	1238	5042	3390	15324	7981	35279
s00832	INI	321	1486	1226	5145	3400	12703	7711	26980
	INT	319	1624	1226	5887	3381	15487	7969	35653
	EVAL	321	1486	1226	5145	3381	15487	7969	35653
s00838	INI	707	2746	-	-	-	-	-	-
	INT	666	5315	-	-	-	-	-	-
	EVAL	707	2746	-	-	-	-	-	-
s00953	INI	538	2627	2688	10495	8925	29761	-	-
	INT	526	2853	2545	10727	8127	29097	-	-
	EVAL	538	2627	2688	10495	8127	29097	-	-
s01238	INI	2913	5913	30481	52475	-	-	-	-
	INT	4011	10268	46521	99528	-	-	-	-
	EVAL	2353	5458	27420	45566	-	-	-	-
s01423	INI	52825	199694	-	-	-	-	-	-
	INT	14516	88038	-	-	-	-	-	-
	EVAL	14516	87987	-	-	-	-	-	-
s01488	INI	490	2163	1716	7641	4246	18618	8510	37688
	INT	516	2833	2085	10881	5837	29058	13318	64932
	EVAL	490	2163	1716	7641	4246	18618	8510	37688
s01494	INI	490	2155	1710	7580	4236	18456	8481	37312
	INT	516	2836	2044	10658	5733	28348	12911	62410
	EVAL	490	2155	1710	7580	4236	18456	8481	37312
s05378	INI	-	-	-	-	-	-	-	-
	INT	10079	47145	-	-	-	-	-	-
	EVAL	10079	47145	-	-	-	-	-	-

Table 4. Results for *c0017*

method	$k=5$	
	PO	*max*
INI	65	164
INV	69	144
TOP	65	164
DEP	48	89
FAN	48	166
INT	48	166