A Binary Decision Diagram based approach on improving Probabilistic Databases

Kaj Derek van Rijn
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
k.d.vanrijn@student.utwente.nl

ABSTRACT

This research focusses on improving Binary Decision Diagram algorithms in the scope of probabilistic databases. The driving factor is to create probabilistic databases that scale, the first order logic formulae generated when retrieving data from these databases create a bottleneck in the scalability in the number of random variables. It is believed that Binary Decision Diagrams are capable of manipulating these formulae in a more scalable way. This research studies the complexity of existing BDD algorithms with regards to characteristics such as the depth and breadth of a tree, the scale and other metrics. These results will be evaluated and compared to the characteristics of equations typically produced by probabilistic databases. We present an improved probabilistic BDD construction algorithm that scales better.

Keywords

Binary Decision Diagrams, Probabilistic Databases, Uncertain Data

1. INTRODUCTION

In recent years there has been an increasing amount of interest in dealing with large and imprecise quantities of data [6]. This interest arises from a variety of different fields such as finding patterns in big data, processing data with high uncertainties, data cleaning [7], data integration and information retrieval from natural language text.

Currently, work is being done at the University of Twente to develop a new probabilistic database called DuBio. This new PDB takes inspiration from the current state of the art database MayBMS [1]. One of the main improvements that DuBio hopes to achieve is a better handling of random variables assignments (rva) such that more variables can be stored and confidence computations can be calculated more efficiently as well as a better theoretical model for uncertain data. A big part of this functionality relies on the use of Binary Decision Diagrams (BDDs). A BDD is a data structure that can be used to represent boolean expressions in a conical form which can be both smaller in size and more efficient to evaluate. The scalability of some of the algorithms currently used in DuBio was determined to be insufficient [5]. This research hopes to come up with algorithms that can create, evaluate and perform logical operations on BDDs in a more efficient way.

33th Twente Student Conference on IT July. 3nd, 2020, Enschede, The Netherlands.

2. BACKGROUND

2.1 Probabilistic Databases

Many real world databases contain uncertainty in correctness of the data, probabilistic databases deal with such uncertainty. We assign probability values (between 0 and 1) to a group of entries in the database, the probabilities in this group add to one. For example, entry A and B have probability values of 0.4 and 0.6 respectively. Probabilistic databases keep track of multiple states and play out different scenarios, a query will not just return a single right answer, but rather a list of possible answers together with the probability that it is true. In the example above the query would return two answers, one where A exists and B doesn't which has a 40% likeliness of being true as well as a 60% chance that only entry B exists. Combinations of these different states can be represented using boolean formulae that grow exponentially as the number of possible worlds expand. In the simple example above there are $\prod_{i=1}^{n} x_n$ possible worlds where n denotes the groups of entries and x_n the number of entries in each group.

Using this probabilistic databases we can accurately represent uncertain data using complicated models, however due to the scalability problems it might prove very difficult to evaluate. Several Probabilistic Database Management Systems (PDBMS) have been developed, all for scientific purposes. As such, probabilistic databases are still an active area of research and this paper hopes to contribute.

2.2 Binary Decision Diagrams

In short, binary decision diagrams are a way to represent boolean expressions as rooted, directed acyclic graphs with some additional properties. BDDs can represent any logical formula, that is, accept the same truth table as that logical formula. A graph representing the BDD consist of the following:

- *internal* vertices with exactly two outgoing edges, labeled 1 and 0.
- Two *terminal* vertices with no outgoing edges, labeled 1 and 0.
- One internal *root* vertex with no incoming edges.

Every internal vertex v has a boolean variable $var(v)$ associated with it, if this variable evaluates to true we direct to the $high(v)$ child (the vertex incident to edge 1) and otherwise the $low(v)$ child (incident to edge 0). Figure 1 describes the BDD associated with common boolean operators.

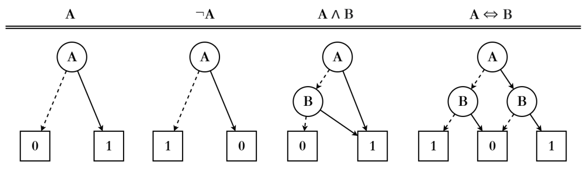


Figure 1: Examples of binary decision diagrams representing boolean formula.

2.3 Reduced Ordered Binary Decision Diagrams

Clearly, there are multiple BDD that can satisfy the same boolean formula, some of which may be compact while another representation might be extremely large. This is not ideal, and we need to work towards a more compact form. For a start we need to order the variables, let us say the place in the ordering of a vertex v is $index(v)$. We create the following restraint:

R1 For any internal vertex v, we have $index(v) > index(high(v))$ and $index(v) > index(low(v))$.

Figure 2 shows two BDDs representing the same boolean function but with different variable orderings.

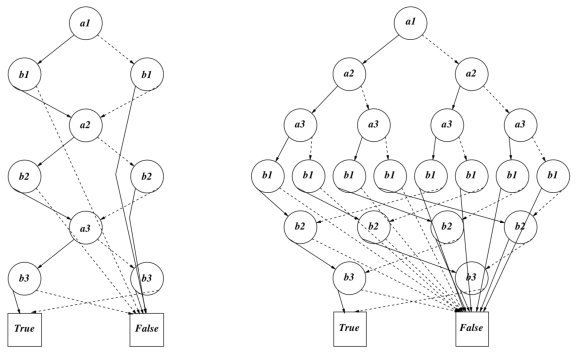


Figure 2: Two BDDs representing the same truth table but with different variable orderings. [5]

Building on the definition of the BDD we define a compact form, the Reduced Ordered Binary Decision Diagram (ROBDD). A ROBDD is an ordered BDD with the following additional properties:

R2 There is no vertex v such that $high(v) = low(v)$.

R3 Given two distinct vertices v and w where $var(v) = var(w)$, either $high(v) \neq high(w)$ or $low(v) \neq low(w)$.

Although multiple ROBDDs may exist with the same truth table, for every ordering of the variables in a boolean function only a single ROBDD exists. Thus ROBDDs are canonical on the ordering of the variables.

In much of the literature, as well as this particular paper, ROBDD are simply referred to as BDD.

Theory about algorithms for constructing and manipulating such BDDs can be found in many publications [4, 8].

2.4 Probabilistic Database ROBDDs

As touched upon earlier in this paper, probabilistic databases operate with random variable assignments (rva) to evaluate possible worlds. $x = 1$ represents the variable with name x is assigned to value 1. We call the number of values a variable can be assigned to its cardinality. Each variable generated by the probabilistic database can only be assigned to one value, for example a variable x with cardinality 2 has the following restraint.

$$(x = 1 \; and \; \neg x = 2) \; or \; (\neg x = 1 \; and \; x = 2) \qquad (1)$$

Applying BDD theory on rva requires a slight manipulation on the structure of the BDDs.

The concept of PDROBDD was proposed in prior research [5] accompanied with basic algorithms. By adding the rva constraint mentioned above we can describe the BDD in a more compact form. Given a vertex v, $name(v)$ denotes the variable name and $value(v)$ denotes its assigned value. The extra constraint imposed on the ROBDD is as follows:

R4 In the subtree induced by the high child of vertex v, there exists no vertex w such that $name(v) = name(w)$.

This constraint can be fulfilled by realizing that the assignment of rva v true means that all child vertices of that branch must evaluate to false. The parent vertices must simply point to the low child of vertex w after which the redundant nodes can be removed according to the ROBDD constraints.

An example of this is illustrated in Figure 3 and 4 which represent the following expression.

$$((x = 0 \wedge y = 1) \vee (\neg x = 0 \wedge \neg y = 1)) \vee ((x = 2 \wedge y = 2) \wedge y = 34) \qquad (2)$$

The PDROBDD, generated by Cincotta's algorithm, in Figure 4 represents the same expression in a graph with less vertices and shallower depth. When analyzing the truth table however, we can deduce a more compact bdd as seen in Figure 5. The reduction from Fig. 4 to 5 can be followed as follows, first we can redirect low edge of vertex $x0$ straight to $y34$ since either $x = 0$ or $x = 2$ must evaluate to be true. Furthermore we can eliminate vertex $y34$ (redirect to $y1$) by noticing that its high branch implies node $y1$ to be false.

We see that the added restraint **R4** does not result in BDDs that are canonical in the variable order.

3. PROBLEM STATEMENT

The main research question that this research hopes to answer is: *In what ways can Binary Decision Diagrams improve the efficiency in probabilistic databases?* This is divided into multiple sub-questions that help answering the main question.

- **RQ1** What metrics and scalability factors are important in determining the size and computational efficiency of BDDs?
- **RQ2** What are the characteristics of first order logic formulae typically produced by probabilistic databases?
- **RQ3** How can existing algorithms be improved to suit these characteristics?
- **RQ4** To what extent do these algorithms improve the scalability?

4. IMPROVED ALGORITHM FOR PDROBDD CONSTRUCTION

The MK and BUILD algorithms have been extensively explained by Andersen in his introduction to Binary Decision

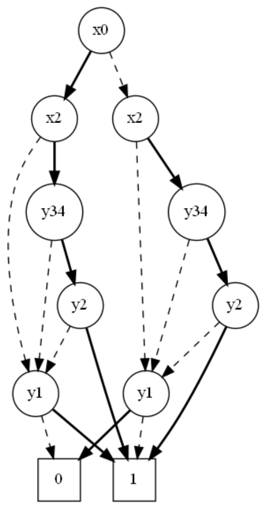


Figure 3: ROBDD approach.

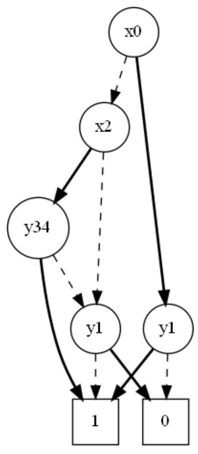


Figure 4: Probabilistic approach. [5]

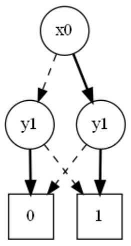


Figure 5: Minimized PDROBDD.

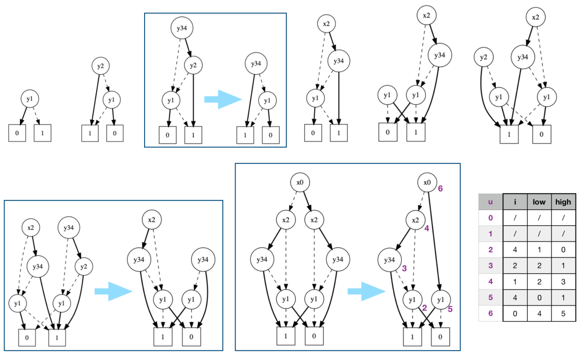
Diagrams [2]. Simply put, we have a table represenation of the BDD where each record stores the variable name, low and high node of each node. The MK algorithm receives the same input, and uses a hash table to lookup and return the index of the correct node or create a new node in case it is not present. The BUILD algorithm takes an expression and variable pointer, by shannon expansion it evaluates the variable in the expression to true and false, recursively calling itself with the new expression and iterating the pointer. When BUILD has parsed all variables the resulting expressions are evaluated and returns the index of the node added using MK.

As described in works by Bryant, the complexity for building reduced ordered binary decision diagrams is 2^n where n is the number of variables[3]. This is because the algorithm recursively performs a shannon expansion exploring each variable being assigned to 0 and 1, each expansion results in an MK call (which is constant in time because of the hash table). In the worst case (conjunction of variables) the depth of a BDD generated by such an algorithm is the number of disctinct variables in the expression. However, in many practical use cases the depth is much smaller than that.

4.1 Current approach

The current implementation of PDROBDDs in DuBio is based on the same two algorithms as well as an additional algorithm called CHECK [5]. The CHECK algorithm is called each time a new node is added to the tree and searches the subtree induced by the newly added node for violations of restraint **R4**. Because the variable ordering groups variable assignments, each tree only has to be searched for a maximum depth equal to the cardinality of each random variable. Other checks are implemented to further reduce the added computational load, such as a variable name check when traversing the branch.

Figure 6 shows the check function in action, the blue boxes indicate a reduction of the BDD.



u	i	low	high
0	/	/	/
1	/	/	/
2	4	1	0
3	2	2	1
4	1	2	3
5	4	0	1
6	0	4	5

Figure 6: Construction of PDROBDD using CHECK.

Although Cincotta's algorithms are capable of generating PDROBDDs following the additional constraint **R4**, thus resulting in more compact BDDs, it does not improve the complexity. The reason for this is because the constant time function MK is called just as many times.

4.2 Improvement of algorithm

The improved algorithm is actually a modification of the original MK and BUILD algorithms. The essence of the algorithm is not to go through all 2^n combinations of variables but rather use the rva properties explore possible combinations. For example in the Expression 2 mentioned above we know either $x = 0$ or $x = 1$ can be true, the same can be said for $y = 0$, $y = 1$ and $y = 34$. Therefor there are only $2 * 3$ possibilities that have to be explored, as shown by table 1.

x=0	x=1	y=0	y=1	y=34
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0

Table 1: Possible variable assignments

To implement this we modify the BUILD algorithm such that for any variable assignment that is set to true, we set all succeeding variable assignments with the same variable name to false. The next build call can iterate its variable pointer to the next variable name. We also make sure that at least one of each variable name is assigned to true. Using this method the number of MK function calls is reduced to the number of worlds in the probabilistic database as mentioned in section 2.1.

The implementation can be seen in Algorithm 1 and Algorithm 2 where the MK algorithm is mostly kept the same. Next to the methods and global variables used in the original algorithm, an additional hash map is store of the form:

$$rva_{o}rder = \{var\_name => [first\_occurrence, cardinality]\} \tag{3}$$

This hash map is used in the function $get\_last\_rva\_pos(var\_name)$ to efficiently find the variables that must be assigned to false in the high branch.

The extra input variable *last_assigned_var* stores the variable name of the variable that was assigned to true, this is used to make sure each variable has at least one assignment evaluated to true.

Because the MK function is constant time with regards to the number of variables, the expectation is that the time complexity scales with the number of MK calls. Since the number MK calls is reduced significantly compared to the previous implementations, it is assumed that the performance increase maps relates to this improvement. However the improvement might not relate directly to the fraction of MK calls because of the redundant nature of the algorithm (MK calls with $l == h$).

Algorithm 1: MK

```
Result: If node is in tree return the index of the node,
        otherwise create new node at end of tree and
        return the index.
Input: variable $v$, low pointer $l$, high pointer $h$
if $l == h$ then return h ;
node ← lookup(v, l, h);
if $node \geq -1$ then return node ;
node ← create_node(v, l, h);
insert_hash(v, l, h, node);
return node;
```

Algorithm 2: BUILD_PDROBDD

```
Result: Recursive function to build ROBDD
Input: boolean expression $expr$, pointer $i$
if $i \leq n$ then
    if $expr$ then
        return 1;
    else
        return 0;
    end
end
var ← order[i];
l_expr ← expr[var → 0];
l ← BUILD_ROBDD(l_expr, i + 1);
h_expr ← expr[var → 1];
for $v \in order$ do
    if $name(v) = name(var)$ then
        h_expr ← expr[v → 1];
    end
end
next ← get_next_rva_pos(var) h ←
  BUILD_ROBDD(h_expr, next)
return MK(var, l, h)
```

5. EXPERIMENTAL RESULTS

5.1 method

To facilitate easy experimentation and testing all versions of the algorithms as explained above have been implemented in Python version 3.7.1, the two probabilistic implementations are a subclass of the original implementation and thus share the majority of functions. This does have the consequence that the execution times will be much higher than if it were to be implemented in a low-level programming language.

To compare the algorithms, expressions are synthetically generated. First in a rather extreme case, increasing the number of variables and their cardinality, and secondly an example based on a real world duplicate detection.

The python timeit library is used to measure execution times, each expression is run 10 times on a Windows machine and the average result is taken. Furthermore the number of time the MK function is called is recorded as well as the CHECK algorithm in Cincotta's implementation. The resulting BDDs will be analyzed in the number of nodes which should give an idea about the average depth of the tree.

5.2 Synthetic data generation

The expressions are generated by two variables, the number of distinct variable names and the number of variables they can be assigned to. The structure of the expression is as follows.

$$(a1 \wedge b1 \wedge c1 \wedge \dots) \vee (a2 \wedge b2 \wedge c2 \wedge \dots) \vee \dots \qquad (4)$$

The results of three experiments are shown in Table 2. The measured MK calls using Cincotta's algorithm is the same as in the original. We notice that in all experiments the time complexity of both the original and Cincotta's algorithm is indeed 2^n in the number of variable assignments. Sadly, Cincotta's algorithm ran into exceptions during test B. This is marked by the abbreviation EXC, this does not mean that the original algorithm is incorrect but rather that there is a problem in our implementation of the algorithm. Tests that ran over a minute were halted, this is marked by DNF (did not finish).

Experiment A varies the number of distinct variables while keeping the cardinality at 1, this shows the strength of the new algorithm. We see big improvements in the conjunction scenario, this is because each variable must be assigned to the one possible value. to this single value. Although the execution times in this scenario are too small to generate reliable results, this scenario will have a complexity of $O(n)$ in the number of distinct variables.

Experiment B also varies the number of distinct variables, this time having two possible values for each variable. With regards to the total number of variable assignments this is essentially a worst case scenario as it creates the highest number of possible worlds. Again the execution time roughly follows the number of MK calls.

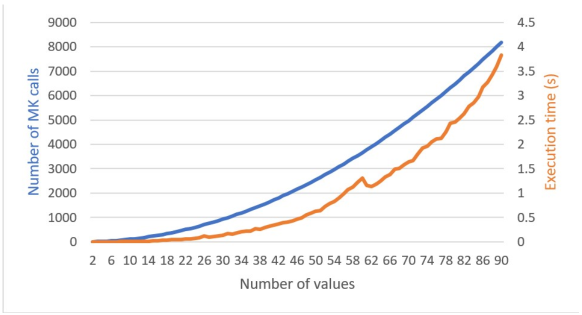


Figure 7: Examples of binary decision diagrams representing boolean formula.

In experiment C the number of variables is kept at two and their cardinality is increased, this is similar to what a duplicate detection use case would look like. This time the improved algorithm outperforms the original algorithm by a large margin. Not only did the execution times drop considerably, the size of the resulting BDD is just a fraction compared to the original diagram. Note in the time that the previous algorithm computes the expression with 10 values, the improved algorithm has computed a scenario with 20x the amount of variables. If by some miracle the

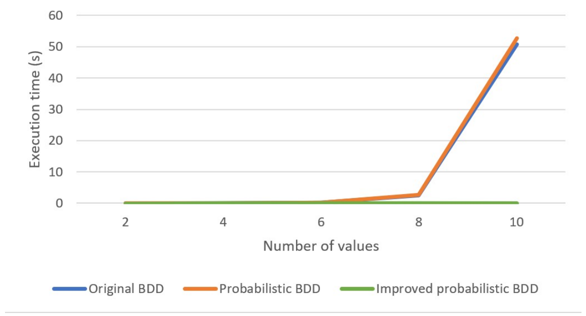


Figure 8: Examples of binary decision diagrams representing boolean formula.

original algorithm was able to compute the appropriate BDD, it would have more than a Novemdecillion [9] nodes.

The graph in Figure 7 confirms our hypothesis of the execution time being proportional to the number of MK calls in this experiment. In fact, this holds for all performed experiments. The reason the execution time grows slightly steeper than the number of MK calls is most likely caused by the increase in depth of the of the recursive BUILD function, which increases memory usage. Furthermore, the complexity of the shannon expansion method used in the BUILD function is $O(n)$ in the length of the expression.

The graph in Figure 8 shows just how significant the improvement of the new algorithm is. There is no real point in showing the same graph for the other experiments as the improved algorithm will always show as a near straight line along the x axis, or the original algorithms along the y axis in much the same manner.

6. DISCUSSION AND FUTURE WORK

Although this research has led to a significant increase in the scalability of probabilistic binary decision diagrams, it still has its limitations. For a start it only focusses on algorithms for building PDROBDDs, the apply algorithms (ADD/OR operation between two BDDs) has barely been studied. It is very much possible that these can be improved following the extra restraints imposed by PDROBDDs.

Furthermore it is likely that adding an additional restraint, namely at least one assignment of a variable must be true, will make PDROBDDs canonical in their variable ordering. However I have not had the time to show that this is true.

The experiments performed with the new algorithm have only been performed with synthetic data. Although the generated data attempts to model real data, it is still important to test more elaborately with real world data in a variety of scenarios. It is also important to analyze the resulting BDDs further, for example analyzing the maximum, minimum and average depth of each path either by estimation or traversal. When implemented in a database management system it can be compared to existing systems such as MayBMS.

7. CONCLUSION

Binary Decision Diagrams are a very promising concept in the field of Probabilistic Databases, the results of this research help to find a scalable approach to creating these diagrams in a probabilistic context. The improved algorithm allows for a major improvement in the scalability of PDROBDD construction whilst maintaining and in many cases improving the size of the resulting tree. The complexity has gone from being exponential $O(2^n)$, where n is the total number variables in the expression, to *at most* polynomial $O(c^n)$ where n denotes the number of *unique* variables and c the maximum cardinality. The improvement is especially noticeable in applications with a low number of unique variables with a high cardinality. Further research of PDROBDDs is required as well as an implemention into a PDBMS to test the improvements in real world scenarios.

.

8. REFERENCES

[1] C. Aggarwal. *MayBMS A System for Managing Large Probabilistic Databases*, pages 1–34. 08 2010.

[2] H. R. Andersen. An introduction to binary decision diagrams. *Lecture notes, available online, IT University of Copenhagen*, page 5, 1997.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, Aug. 1986.

[4] R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

[5] V. Cincotta. *Design and Implementation of a Scalable Probabilistic Database System*. PhD thesis, 2019.

[6] N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: Diamonds in the dirt. *Communications of the ACM*, 52(7):86–94, 2009.

[7] M. V. Keulen. *Probabilistic Data Integration*, pages 1–9. Springer International Publishing, Cham, 2018.

[8] A. Rauzy. A brief introduction to Binary Decision Diagrams. *Journal Europeen des Systemes Automatises*, 30(8):1033–1050, 1996.

[9] Wikipedia. Names of large numbers — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Names%20of%20large%20numbers&oldid=964718443`, 2020. [Online; accessed 28-June-2020].

			Original BDD			Probabilistic BDD			Improved probabilistic BDD		
	#variables	#values	Time (s)	Size	#MK	Time (s)	Size	#CHECK	Time (s)	Size	#MK
A	5	1	0.0006289	7	31	0.000965	7	0	0.0004055	2	5
	10	1	0.0252015	12	1023	0.024902	12	0	0.0003764	2	10
	15	1	0.9246912	17	32767	1.082633	17	0	0.0010316	2	15
	20	1	39.329198	22	1048575	37.5795	22	0	0.0005609	2	20
	40	1	DNF	42	1.10E+12	DNF	42	0	0.000612	2	40
B	2	2	0.0002961	8	15	0.000373	6	1	0.0006359	5	6
	4	2	0.0099531	16	255	EXC	EXC	EXC	0.0015981	9	30
	6	2	0.1148494	24	4095	EXC	EXC	EXC	0.0057161	13	126
	8	2	2.0643986	32	65535	EXC	EXC	EXC	0.0136911	17	510
	10	2	39.795957	40	1048575	EXC	EXC	EXC	0.0710281	21	2046
	12	2	DNF	48	1.68E+07	EXC	EXC	EXC	0.3492078	25	8190
	14	2	DNF	52	2.68E+08	EXC	EXC	EXC	1.289364	29	32766
	16	2	DNF	56	4.29E+09	EXC	EXC	EXC	5.2376908	33	131070
C	2	2	0.0002961	8	15	0.000373	6	1	0.0006359	5	6
	2	4	0.0083945	32	255	0.015843	23	7	0.0007998	14	20
	2	6	0.1471312	128	4095	0.151929	77	36	0.0033422	27	42
	2	8	2.5496329	512	65535	2.751452	270	169	0.0055764	44	72
	2	10	50.74679	2048	1048575	52.7689	1007	730	0.0065106	65	110
	2	25	DNF	67108864	1.13E+15	DNF	DNF	DNF	0.0975428	350	650
	2	50	DNF	2.25E+15	1.27E+30	DNF	DNF	DNF	0.7933124	1325	2550
	2	100	DNF	2.54E+30	1.61E+60	DNF	DNF	DNF	5.8028299	5150	10100
	2	200	DNF	3.21E+60	2.58E+120	DNF	DNF	DNF	47.659097	20300	40200

Table 2: Test results corresponding to the form $x = 0 \wedge y = 0 \vee \dots$