Exploiting Asymmetry in Logic Puzzles

Using ZDDs for Symbolic Model Checking Dynamic Epistemic Logic

Miedema, D.; Gattinger, M.

DOI
10.4204/EPTCS.379.32

Publication date
2023

Document Version
Final published version

Published in
Electronic Proceedings in Theoretical Computer Science

Exploiting Asymmetry in Logic Puzzles: Using ZDDs for Symbolic Model Checking Dynamic Epistemic Logic

Daniel Miedema
Bernoulli Institute
University of Groningen
The Netherlands
daniel2Miedema@gmail.com

Malvin Gattinger
ILLC
University of Amsterdam
The Netherlands
malvin@w4eg.eu

Binary decision diagrams (BDDs) are widely used to mitigate the state-explosion problem in model checking. A variation of BDDs are Zero-suppressed Decision Diagrams (ZDDs) which omit variables that must be false, instead of omitting variables that do not matter.

We use ZDDs to symbolically encode Kripke models used in Dynamic Epistemic Logic, a framework to reason about knowledge and information dynamics in multi-agent systems. We compare the memory usage of different ZDD variants for three well-known examples from the literature: the Muddy Children, the Sum and Product puzzle and the Dining Cryptographers. Our implementation is based on the existing model checker SMCDEL and the CUDD library.

Our results show that replacing BDDs with the right variant of ZDDs can significantly reduce memory usage. This suggests that ZDDs are a useful tool for model checking multi-agent systems.

1 Introduction

There are several formal frameworks for reasoning about knowledge in multi-agent systems, and many are implemented in the form of epistemic model checkers. Here we are concerned with the *data structures* used in automated epistemic reasoning. This is a non-issue in theoretical work, where Kripke models are an elegant mathematical tools. But they are not very efficient: models where agents know little tend to be the largest. More efficient representations are often based on Binary Decision Diagrams (BDDs), which use the idea that a representation of a function not depending on p can simply ignore that variable p. This fits nicely to the models encountered in epistemic scenarios, such as the famous example of the Muddy Children: If child 2 does not observe whether it is muddy, i.e. whether p_2 is true or false, then we can save memory by omitting p_2 in the encoding of the knowledge of child 2. However, which variables matter may change, and in many examples the claim that "many variables do not matter" only holds in the initial model. This motivates us to look at Zero-suppressed Decision Diagrams (ZDDs) which use an asymmetric reduction rule to omit variables that *must* be *false*, instead of the symmetric reduction rule targeting variables that *do not matter*.

Our informal research question is thus: Is it more memory efficient to have a default assumption that "anything we do not mention does not matter" or, for example "anything we do not mention must be false"? Obviously, the answer will depend on many aspects. Here we make the question precise for the case of Dynamic Epistemic Logic, and consider three well-known examples from the literature.

The article is structured as follows. We discuss related work in the rest of this section, then we provide the relevant background in Sections 2 and 3. Section 4 describes our experiment design and the formal models used. We present our results in Section 5 and conclude in Section 6.

R. Verbrugge (Ed.): Theoretical Aspects of
Rationality and Knowledge 2023 (TARK 2023)
EPTCS 379, 2023, pp. 407–420, doi:10.4204/EPTCS.379.32

Related work Model checking aims to verify properties of formally specified systems. Standard model checking methods search through a whole state transition graph and thus suffer from the state explosion problem: the number of states grows exponentially with the number of components or agents. To tackle this problem *symbolic* methods were developed [4]. These reduce the amount of resources needed, by reasoning about sets instead of individual states. Starting with SMV from [16], most approaches use Binary Decision Diagrams (BDDs) [2] to encode Boolean functions. Zero-suppressed Decision Diagrams (ZDDs) are an adaption of BDDs, introduced by Minato [18]. ZDDs naturally fit combinatorial problems and many comparisons between BDDs and ZDDs have been done. For both an elegant introduction into the topic of BDDs and many more references we refer to [13]. Symbolic model checking using ZDDs has not been studied much, partly due to underdeveloped construction methods [19].

Most existing symbolic model checkers use temporal logics such as LTL or CTL. Yet problems come in many forms and for examples typically described using epistemic operators (e.g. in multi-agent systems), Dynamic Epistemic Logic (DEL) is an established framework [8]. Also DEL model checking can be done symbolically [1], by encoding Kripke models as so-called knowledge structures. This lead to its implementation, SMCDEL, which is extended in this work. Another encoding, sometimes also called "symbolic models", is based on mental programs [6]. In concrete applications such as "Hintikka's World" these also get encoded as BDDs [5]. To our knowledge no previous work used ZDDs or other BDD variants for DEL model checking, with the exception of [12] where Algebraic Decision Diagrams (ADDs) are used for probabilistic DEL.

Here our main research questions is: Can ZDDs be more compact than BDDs when encoding the Kripke models for classical logic puzzles? We answer this question by adding ZDD functionality to SMCDEL and then comparing the sizes for three well-known examples from the literature.

2 Theory: Decision Diagrams

Symbolic model checkers, including SMCDEL, rely on efficient representations of Boolean functions. The most widely used data structure for this are Binary Decision Diagrams (BDDs). In this section we recall their definition and explain the difference between standard BDDs and ZDDs. How Boolean functions are then used for model checking DEL will be explained in the next section. Before we get to decision diagrams we define Boolean formulas and functions.

Definition 1. *The* Boolean formulas *over a set of variables P (also called* vocabulary*) are given by $\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi$ where $p \in P$. We define $\bot := \neg\top$, $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$ and $\varphi \to \psi := \neg(\varphi \wedge \neg\psi)$.*

We write $\vDash$ for the usual Boolean semantics using assignments of type $P \to \{0,1\}$. When P is given we identify an assignment (also called state*) with the set of variables it maps to* 1*. A* Boolean function *is any $f\colon \mathscr{P}(P) \to \{0,1\}$. For any φ we define the Boolean function $f_\varphi(s) := \{$if $s \vDash \varphi$ then 1 else 0$\}$.*

For example, if our vocabulary is $P = \{p,q,r\}$ and $s(p) = 0$, $s(q) = 1$ and $s(r) = 0$ then we identify s with $\{q\}$ and we have $s \vDash (\neg p \wedge q) \vee r$. In the following we will also just write φ for f_φ. Notably, two different formulas can correspond to the same Boolean function, but not vice versa.

Definition 2. *For any φ, ψ, and p, let $\varphi(\frac{p}{\psi})$ be the result of replacing every occurrence of p in φ by ψ. For any $A = \{p_1, \ldots, p_n\}$, let $\varphi(\frac{A}{\psi}) := \psi(\frac{p_1}{\psi})(\frac{p_2}{\psi})\ldots(\frac{p_n}{\psi})$. We use $\forall p\varphi$ to denote $\varphi\left(\frac{p}{\top}\right) \wedge \varphi\left(\frac{p}{\bot}\right)$. For any $A = \{p_1, \ldots, p_n\}$, let $\forall A\varphi := \forall p_1 \forall p_2 \ldots \forall p_n \varphi$.*

Decision Diagrams A decision diagram is a rooted directed acyclic graph, used to encode a Boolean function. Any terminal node (i.e. leaf) is labelled with 0 or 1, corresponding to the result of the function.

Any internal node n is labelled with a variable and has two outgoing edges to successors denoted by $\text{THEN}(n)$ and $\text{ELSE}(n)$ — each representing a possible value for the variable. A path from the root to a leaf in a decision diagram corresponds to an evaluation of the encoded function. A decision diagram is called *ordered* if the variables are encountered in the same order on all its paths.

Example 3. *The first (left-most) decision diagram in Figure 1 is a full decision tree for $q \wedge \neg r$. To evaluate it at state $\{p,q\}$ we start at the root and then go along the solid* THEN*-edge because p is true, then again along a* THEN*-edge as q is true and then along the dashed* ELSE*-edge as r is false. We get* 1 *as a result, reflecting the fact that $\{p,q\} \models q \wedge \neg r$. Similarly we can use the second and third diagram.*

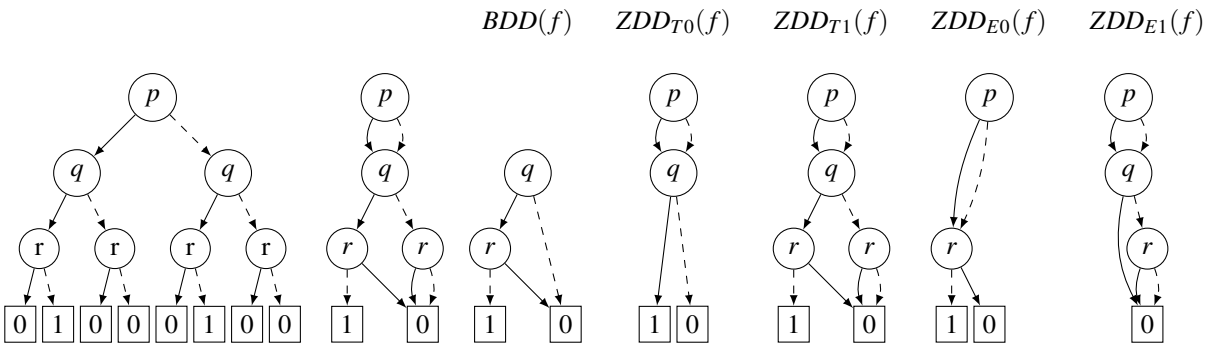


Figure 1: Seven decision diagrams for $f := q \wedge \neg r$, assuming vocabulary $\{p,q,r\}$.

Binary Decision Diagrams (BDDs) were introduced by [2] and are particularly compact decision diagrams, obtained using two reduction rules. The first rule identifies isomorphic subgraphs, i.e. we merge nodes that have the same label and the same children. In Figure 1 we get from the first to the second diagram. The second rule eliminates redundant nodes. A node is considered redundant if both its THEN- and ELSE-edge go to the same child. In Figure 1 this gets us from the second to the third diagram.

Zero-suppressed Decision Diagrams (ZDDs) were introduced by [18] and use a different second rule than BDDs. While in BDDs a node n is eliminated when $\text{THEN}(n) = \text{ELSE}(n)$, in ZDDs a node is eliminated when $\text{THEN}(n) = 0$. In Figure 1 this rule gets us from the second to the fourth diagram called $ZDD_{T0}(f)$. The idea is to not ignore the variables that "do not matter" (as p in $q \wedge \neg r$), but to remove the nodes of variables that must be false (as r in $q \wedge \neg r$). To evaluate $ZDD_{T0}(f)$ at state $\{p,q\}$ we again start at the root and twice follow a solid edge because p and q are true, but then we notice that the solid edge goes from q to 1, without asking for the remaining variable r. When evaluating a ZDD_{T0} such a transition demands that the variable we "jump over" must be false — hence the name "zero-suppressed". Indeed r is false in our state, so we do reach 1. If r would have been true, the result would have been 0.

Generalizing Elimination Rules The elimination rule "remove nodes that have a THEN-edge leading to 0" can be modified in two obvious ways: instead of THEN- we could consider ELSE-edges, and instead of 0 we could consider 1. This leads us to three additional elimination rules.

Definition 4. *We denote five different node elimination rules as follows. A node n with pairs of children $(\text{THEN}(n), \text{ELSE}(n))$ is eliminated if it matches the left side of the rule, and any edges leading to n are diverted to the successor s on the right side of the rule.*

$$EQ: \ (s,s) \Rightarrow s \qquad T0: \ (0,s) \Rightarrow s \qquad T1: \ (1,s) \Rightarrow s \qquad E0: \ (s,0) \Rightarrow s \qquad E1: \ (s,1) \Rightarrow s$$

Here EQ is the rule for BDDs, while $T0$ (for "Then 0") is the traditional ZDD rule. The remaining three are variations. For example, $E0$ says that any node with an ELSE-edge to 0 is removed, and any edge that led to the removed node should be diverted to where the THEN-edge of the removed node led.

In Figure 1 the $E0$ rule gets us from the second to the sixth diagram $ZDD_{E0}(f)$. Note that we used the rule twice: After deleting an r node the q node has an ELSE-branch to 0, so it is also eliminated. All diagrams encode the same function f, but when evaluating them we must interpret "jumps" differently.

A crucial feature of BDDs and ZDDs is that they are *canonical* representations: given a fixed variable order there is a unique BDD and a unique ZDD for each variant. It also becomes clear that for different Boolean functions a different kind of diagram can be more or less compact.

Definition 5. *For any Boolean function f, recall that $\neg f$ denotes its complement. Let $\lrcorner f$ denote the result of complementing all atomic propositions inside f. (For example, $\lrcorner(q \wedge \neg r) = \neg q \wedge r$.) For any decision diagram d, let* $\mathsf{flipLeaf}(d)$ *be the result of changing the labels of all leaves from 0 to 1 and vice versa; and let* $\mathsf{flipEdge}(d)$ *be the result of changing the labels of all edges from* THEN *to* ELSE *and vice versa.*

There is a correspondence between $\neg$ and flipLeaf, and between $\lrcorner$ and flipEdge. Moreover, we can use these operations to relate the four different variants of ZDDs as follows.

Fact 6. *For any Boolean function f we have:*

$$\begin{array}{lcl} DD_{T1}(f) & = & \mathsf{flipLeaf}\, DD_{T0}(\neg f) \\ DD_{E0}(f) & = & \mathsf{flipEdge}\, DD_{T0}(\lrcorner f) \\ DD_{E1}(f) & = & \mathsf{flipEdge}\, \mathsf{flipLeaf}\, DD_{T0}(\neg\lrcorner f) \end{array}$$

Example 7. *We illustrate Fact 6 using our running example $f := q \wedge \neg r$ with vocabulary $\{p,q,r\}$. Figure 2 shows the $T0$ decision diagrams mentioned in Fact 6. We see that for example $DD_{T1}(f)$ shown in Figure 1 is the same graph as $DD_{T0}(\neg f)$ with only the labels of the leaf nodes exchanged. Similarly, $DD_{E1}(f)$ in Figure 1 is the same graph as $DD_{T0}(\neg\lrcorner f)$ with flipped edges and leaves.*

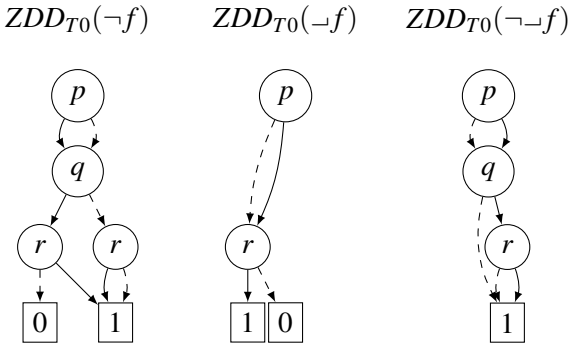


Figure 2: ZDDs with the same shape as the variants for $f := p \wedge \neg q$.

Fact 6 is crucial for our implementation, because the CUDD library we use does not support $T1$, $E0$ and $E1$ explicitly. Hence instead we always work with $T0$ diagrams of the negated or flipped functions.

3 Theory: Symbolic Model Checking DEL

Kripke Models We recap the standard syntax and semantics of Public Announcement Logic (PAL), the most basic version of Dynamic Epistemic Logic (DEL).

Definition 8. *Fix a vocabulary V and a finite set of agents I. The DEL language $\mathscr{L}(V)$ is given by $\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi \mid [\varphi]\varphi$ where $p \in V$, $i \in I$.*

As usual, $K_i\varphi$ is read as *"agent i knows that φ"*. The formula $[\psi]\varphi$ says that after a *public announcement* of ψ, φ holds. The standard semantics for $\mathscr{L}(V)$ on Kripke models are as follows.

Definition 9. *A* Kripke model *for a set of agents $I = \{1,\dots,n\}$ is a tuple $\mathscr{M} = (W, \pi, \mathscr{K}_1, \dots, \mathscr{K}_n)$, where W is a set of* worlds*, π associates with each world a state $\pi(w)$, and $\mathscr{K}_1, \dots, \mathscr{K}_n$ are equivalence relations on W. A* pointed Kripke model *is a pair $(\mathscr{M}, w)$ consisting of a model and a world $w \in W$.*

Definition 10. *Semantics for $\mathscr{L}(V)$ on pointed Kripke models are given inductively as follows.*

- $(\mathscr{M}, w) \vDash p$ *iff* $\pi^M(w)(p) = \top$.
- $(\mathscr{M}, w) \vDash \neg\varphi$ *iff not* $(\mathscr{M}, w) \vDash \varphi$
- $(\mathscr{M}, w) \vDash \varphi \wedge \psi$ *iff* $(\mathscr{M}, w) \vDash \varphi$ *and* $(\mathscr{M}, w) \vDash \psi$
- $(\mathscr{M}, w) \vDash K_i\varphi$ *iff for all* $w' \in W$, *if* $w \mathscr{K}_i^M w'$, *then* $(\mathscr{M}, w') \vDash \varphi$.
- $(\mathscr{M}, w) \vDash [\psi]\varphi$ *iff* $(\mathscr{M}, w) \vDash \psi$ *implies* $(\mathscr{M}^\psi, w) \vDash \varphi$ *where $\mathscr{M}^\psi$ is a new model based on the set $W^{\mathscr{M}^\psi} := \{w \in W^{\mathscr{M}} \mid (\mathscr{M}, w) \vDash \psi\}$ and appropriate restrictions of $\mathscr{K}_i$ and π to $W^{\mathscr{M}^\psi}$.*

More expressive versions of DEL also include common knowledge and complex epistemic or ontic actions such as private communication, interception, spying and factual change. Moreover, DEL can work both with S5 models and with arbitrary Kripke models. All of this is compatible with the symbolic semantics we recall in the next section, but for our purposes in this article the restricted language above is sufficient, and we only consider S5 models.

Knowledge Structures While the semantics described above is standard, it has the disadvantage that models are represented explicitly, i.e. the number of worlds also determines the amount of memory needed to represent a model. To combat this well-known state-explosion problem we can replace Kripke models with symbolic knowledge structures. Their main advantage is that knowledge and results of announcements can be computed via purely Boolean operations, as shown in [1].

Definition 11. *Suppose we have n agents. A* knowledge structure *is a tuple $\mathscr{F} = (V, \theta, O_1, \dots, O_n)$ where V is a finite set of atomic variables, θ is a Boolean formula over V and for each agent i, $O_i \subseteq V$. The set V is the* vocabulary *and the formula θ is the* state law *of $\mathscr{F}$. The O_i are called* observational variables*. An assignment over V that satisfies θ is a* state *of $\mathscr{F}$. A* scene *is a pair $(\mathscr{F}, s)$ where s is a state of $\mathscr{F}$.*

Example 12. *Consider the knowledge structure $\mathscr{F} := (V = \{p,q\}, \theta = p \to q, O_1 = \{p\}, O_2 = \{q\})$. The states of $\mathscr{F}$ are the three assignments $\varnothing$, $\{q\}$ and $\{p,q\}$. Moreover, $\mathscr{F}$ has two agents who each observe one of the propositions: agent 1 knows whether p is true and agent 2 knows whether q is true.*

We now give semantics for $\mathscr{L}(V)$ on knowledge structures.

Definition 13. *Semantics for $\mathscr{L}(V)$ on scenes are defined as follows.*

- $(\mathscr{F}, s) \vDash p$ *iff* $s \vDash p$.
- $(\mathscr{F}, s) \vDash \neg\varphi$ *iff not* $(\mathscr{F}, s) \vDash \varphi$
- $(\mathscr{F}, s) \vDash \varphi \wedge \psi$ *iff* $(\mathscr{F}, s) \vDash \varphi$ *and* $(\mathscr{F}, s) \vDash \psi$
- $(\mathscr{F}, s) \vDash K_i\varphi$ *iff for all t of $\mathscr{F}$, if $s \cap O_i = t \cap O_i$, then* $(\mathscr{F}, t) \vDash \varphi$.
- $(\mathscr{F}, s) \vDash [\psi]\varphi$ *iff* $(\mathscr{F}, s) \vDash \psi$ *implies* $(\mathscr{F}^\psi, s) \vDash \varphi$ *where* $\mathscr{F}^\psi := (V, \theta \wedge \|\psi\|_{\mathscr{F}}, O_1, \dots, O_n)$.

where $\|\cdot\|_{\mathscr{F}}$ is defined in parallel in the following definition.

Definition 14. *For any knowledge structure $\mathscr{F} = (V, \theta, O_1, \ldots, O_n)$ and any formula φ we define its* local Boolean translation *$\|\varphi\|_{\mathscr{F}}$ as follows.*

$$\begin{array}{lcllcl} \|p\|_{\mathscr{F}} & := & p & \|K_i\psi\|_{\mathscr{F}} & := & \forall(V \setminus O_i)(\theta \to \|\psi\|_{\mathscr{F}}) \\ \|\neg\psi\|_{\mathscr{F}} & := & \neg\|\psi\|_{\mathscr{F}} & \|[\psi]\xi\|_{\mathscr{F}} & := & \|\psi\|_{\mathscr{F}} \to \|\xi\|_{\mathscr{F}^\psi} \\ \|\psi_1 \wedge \psi_2\|_{\mathscr{F}} & := & \|\psi_1\|_{\mathscr{F}} \wedge \|\psi_2\|_{\mathscr{F}} & & & \end{array}$$

where the case for $K_i\psi$ quantifies over the variables not observed by agent i, using Boolean quantification as defined in Definition 2 above.

A main result from [1] based on [21] is that for any finite Kripke model there is an equivalent knowledge structure and vice versa. This means we can see knowledge structures as just another, hopefully more memory-efficient, data structure to store a Kripke model. An additional twist is that we usually store the state law θ not as a formula but only the corresponding Boolean function — which can be represented using a decision diagram as discussed in Section 2.

4 Methods: Logic Puzzles as Benchmarks

Our leading question is whether ZDDs provide a more compact encoding than BDDs for models encountered in epistemic model checking. To answer it we will work with three logic puzzles from the literature. All examples start with an initial model which we encode as a knowledge structure with the state law as a decision diagram. Then we make updates in the form of public announcements, changing the state law. We record the size of the decision diagrams for each update step.

As a basis for our implementation and experiments we use *SMCDEL*, the symbolic model checker for DEL from [1]. SMCDEL normally uses the BDD library CacBDD [15] which does not support ZDDs. Hence we also use the library CUDD [20] which does support ZDDs. However, also CUDD does not support the generalized elimination rules from Definition 4. Therefore we use Fact 6 to simulate the $T1$, $E0$ and $E1$ variants. Our new code — now merged into SMCDEL — provides easy ways to create and update knowledge structures where the state law is represented using any of the four ZDD variants.

An additional detail is that CUDD always uses so-called complement edges to optimize BDDs, but not for ZDDs. To compare the sizes of ZDDs to BDDs without complement edges we still use CacBDD. Altogether in our data set we thus record the sizes of six decision diagrams for each state law: the EQ rule with and without complement edges (called BDD and BDDc) and the four ZDD variants from Definition 4. We stress that by size of a diagram we mean the node count and not memory in bytes, because the former is independent of what libraries are used, whereas the latter depends on additional optimisations.

It now remains to choose examples. We picked three well-known logic puzzles from the literature with different kinds of state laws, such that we also expect the advantage of ZDDs to vary between them.

Muddy Children The Muddy Children are probably the best-known example in epistemic reasoning, hence we skip the explanation here and refer to the literature starting with [14]. A formalisation of the puzzle can be found in [8, Section 4.10] and the symbolic encoding in [1, Section 4].

Dining Cryptographers This problem and the protocol to solve it was first presented by [7]:

> "Three cryptographers gather around a table for dinner. The waiter informs them that the meal has been paid for by someone, who could be one of the cryptographers or the National Security Agency (NSA). The cryptographers respect each other's right to make an anonymous payment, but want to find out whether the NSA paid."

The solution uses random coin flips under the table, each observed by two neighbouring cryptographers but not visible to the third one. A formalisation and solution using Kripke models can be found in [11]. To encode the problem in a knowledge structure we let p_0 mean that the NSA paid, p_i for $i \in \{1,2,3\}$ that i paid. Moreover, p_k for $k \in \{4,5,6\}$ represents a coin. The initial scenario is then $(V = \{p_0,\dots,p_6\}, \theta = \otimes_1\{p_0,p_1,p_2,p_3\}, O_1 = \{p_1,p_4,p_5\}, O_2 = \{p_2,p_4,p_6\}, O_3 = \{p_3,p_5,p_6\})$ where the state law θ says that exactly one cryptographer or the NSA must have paid. In the solution then each cryptographer announces the XOR ($\otimes$) of all bits they observe, with the exception that the payer should invert their publicly announced bit. Formally, we get a sequence of three public announcements $[?!(\otimes p_1,p_4,p_5)][?!(\otimes p_2,p_4,p_6)][?!(\otimes p_3,p_5,p_6)]$ where $[?!\psi]\varphi := [!\psi]\varphi \wedge [\neg!\psi]\varphi$ abbreviates announcing whether. The protocol can be generalised to any odd number n instead of three participants.

Sum and Product The following puzzle was originally introduced in 1969 by H. Freudenthal. The translation is from [9] where the puzzle is also formalised in DEL:

> A says to S and P: I have chosen two integers x, y such that $1 < x < y$ and $x + y \leq 100$. In a moment, I will inform S only of $s = x + y$, and P only of $p = xy$. These announcements remain private. You are required to determine the pair (x, y). He acts as said. The following conversation now takes place: P says: "I do not know it." — S says: "I knew you didn't." — P says: "I now know it." — S says: "I now also know it." — Determine the pair (x, y).

Solving the puzzle using explicit model checking is discussed in [10]. To represent the four variables and their values in propositional logic we need a binary encoding, using $\lceil \log_2 N \rceil$ propositions for each variable that take values up to N. For example, to represent $x \leq 100$ we use $p_1,\dots,p_7$ and encode the statement $x = 5$ as $p_1 \wedge p_2 \wedge p_3 \wedge p_4 \wedge \neg p_5 \wedge p_6 \wedge \neg p_7$, corresponding to the bit-string 0000101 for 5.

The initial state law for Sum and Product is a big disjunction over all possible pairs of x and y with the given restrictions, and the observational variables ensure that agents S and P know the values of s and p respectively. For a detailed definition of the knowledge structure, see [1, Section 5].

The announcements in the dialogue are formalised as follows, combining the first two into one: First S says $K_S\neg\bigvee_{i+j\leq 100} K_P(x = i \wedge y = j)$, then P says $\bigvee_{i+j\leq 100} K_P(x = i \wedge y = j)$ and finally S says $\bigvee_{i+j\leq 100} K_S(x = i \wedge y = j)$. Solutions to the puzzle are states where these three formulas can be truthfully announced after each other. A common variation on the problem is to change the upper bound for $x + y$. We use this to turn obtain a scalable benchmark, starting with 65 to ensure there exists at least one answer.

It is well known that ZDDs perform better on sparse sets [3]. In our case, sparsity is the number of states in the model divided by the total number of possible states for the given vocabulary. Our three examples vary a lot in their sparsity: Muddy Children's sparsity is 0.5 on average (going from 0.875 to 0.125, for 3 agents), Dining Cryptographers is fairly sparse from start to finish (0.25 to 0.0625, for 3 agents), and Sum and Product is extremely sparse (e.g. starting with $< 1.369 \cdot 10^{-7}$ for $x + y \leq 100$).

5 Results

For each example we present a selection of results we deem most interesting, showing differences between BDD and ZDD sizes. The full data set for two examples can be found in the appendix where we also

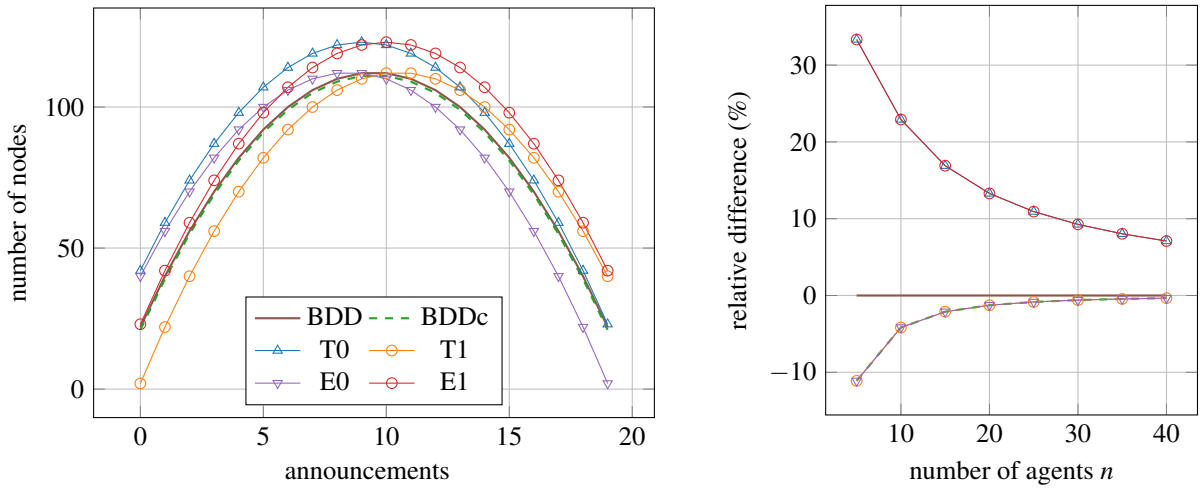
include instructions how all of the results can be reproduced.

Muddy children We vary the number of children n from 5 to 40, in steps of 5. We can also vary the number of muddy children $m \leq n$, but mostly report results here where $m = n$. Given any number of children, we record the size of the decision diagrams of the state law after the kth announcement, where k ranges from 0 (no announcements made yet) to $m-1$ (after which all children know their own state).

As an example, let us fix $n = m = 20$. Figure 3a shows the size of the decision diagrams after each announcement. The lines all follow a similar curve, with the largest relative differences in the initial and final states. Initially the most compact variant is T1 whereas at the end E0 is the most compact. This matches the asymmetry in the Muddy Children story: at the start the state law is $p_1 \vee \ldots \vee p_n$, hence all THEN edges lead to 1 and $T1$ removes all nodes. In contrast, at the end the state law is $p_1 \wedge \ldots \wedge p_n$ which means that all ELSE edges lead to 0 and thus $E0$ eliminates all nodes.

Hence at different stages different variants are more compact. But we want a representation that is compact throughout the whole process. We thus consider the average size over all announcements, varying n from 5 to 40. Figure 3b shows the relative size differences, with standard BDDs as 100%. The $T0$/$E1$ and the BDDc/$E0$/$T1$ lines overlap. We see that $T1$ and $E0$ are more compact for small models, but not better than BDDs with complement edges and this advantage shrinks with a larger number of agents.

We also computed sizes for $m < n$, i.e. not all children being muddy. In this case the sizes for each update step stay the same but there are fewer update steps because the last truthful announcement is in round $m-1$. As expected this is in favour of the $T1$ variant.

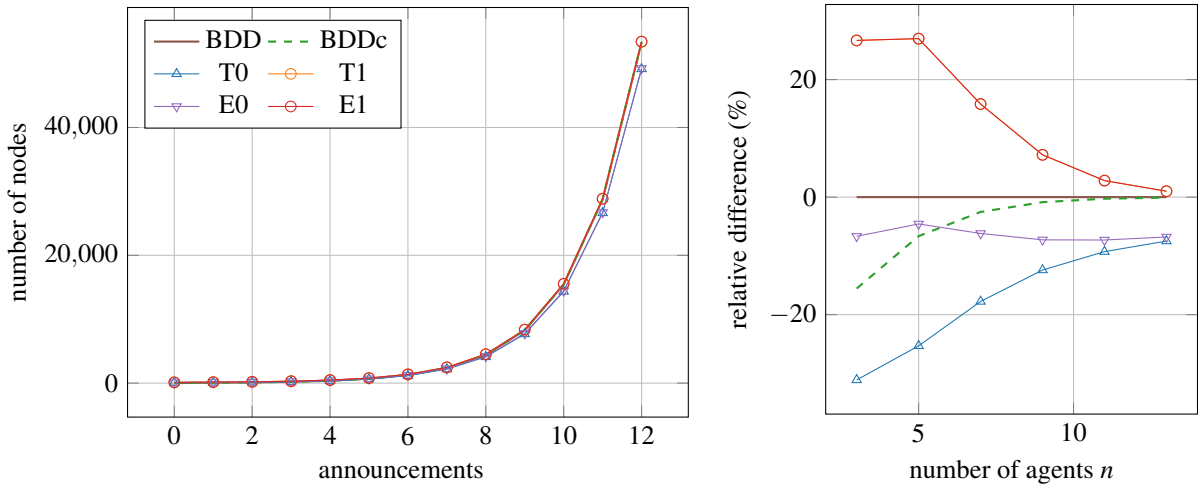


(a) Absolute sizes per announcement, for $n = 20$.

(b) Relative average sizes.

Figure 3: Results for Muddy Children.

Dining cryptographers For 13 agents we show the sizes after each announcement in Figure 4a. It becomes clear that there is little difference between the variants, which can be explained by the sparsity of the model throughout the whole story. Still, the $T0$/$E0$ variants slightly outperform the BDD(c) and the $T1$/$E1$ variants. This makes sense as most variables saying that agent i paid will be false. For lower numbers of agents the difference is larger, as visible in Figure 4b where we vary the number of agents from 3 to 13. Note that $T1$ and $E1$ overlap here, and $T0$ provides the best advantage.

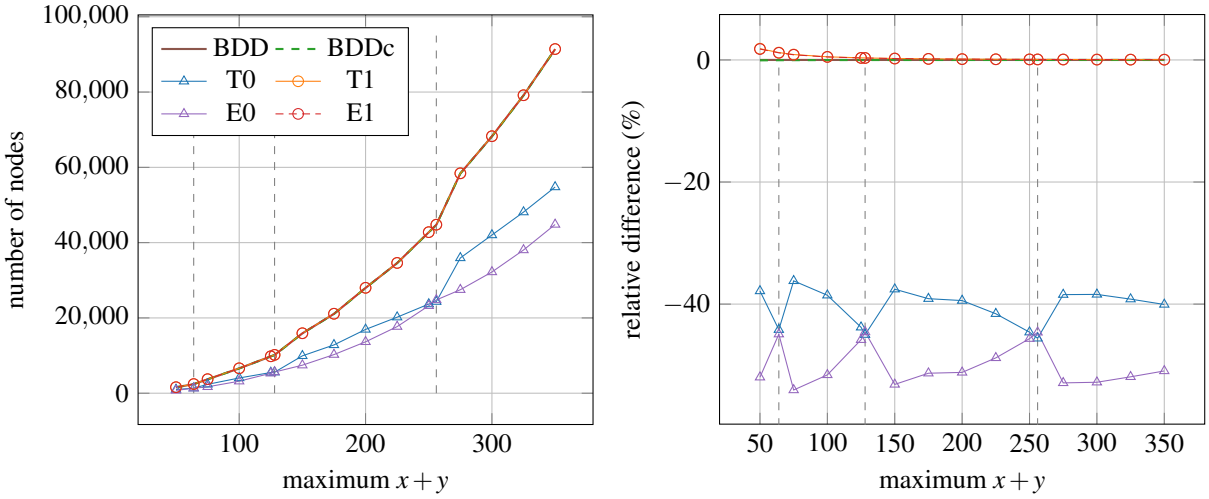


(a) Absolute sizes per announcement, for $n = 13$.

(b) Relative average sizes.

Figure 4: Results for Dining Cryptographers.

Sum and Product In this last example we can vary the upper bound of $x+y$ from 50 to 350, but not the number of agents and announcements. Figure 5a shows the sizes averaged over all four stages. We note that the BDD(c), $T1$ and $E1$ lines all overlap (with insignificant differences), and that T0 and E0 perform the best here. In contrast to the first two examples, this advantage does not disappear for larger instances of the puzzle, as can be seen in Figure 5b where we show the relative differences. Interestingly, we see that $T0$ and $E0$ meet up and diverge again wherever the bound for $x+y$ is a power of 2 (i.e. 64, 128 or 256) which we mark by vertical dashed lines. This is due to the bit-wise encoding where just above powers of two an additional bit is needed, but it must be false for almost all values.



(a) Average sizes per maximum $x+y$.

(b) Relative sizes per maximum $x+y$.

Figure 5: Results for Sum and Product.

6 Conclusion

In all experiments we find a ZDD elimination rule that can reduce the number of nodes compared to BDDs, with the exception that in the Muddy Children example complement edges provide the same advantage. This leads us to conclude that ZDDs are a promising tool for DEL model checking. Specifically, if domain knowledge about the particular model allows one to predict which ZDD variant will be more compact, ZDDs can outcompete BDDs.

The BDD elimination rule treats true and false atomic propositions symmetrically, whereas ZDD rules are asymmetric. This means their success depends on asymmetry in the model.

When translating an example from natural language to a formal models we usually try to avoid redundant variables, which already reduces the number of BDD-eliminable nodes. This is likely the reason why using ZDDs provides an advantage or, for examples with a sparsity around 0.5 like the Muddy Children, at least the same performance as BDDs with complement edges.

Specifically for logic puzzles, usually all variables are needed, and models become asymmetric and sparse as information is revealed and possible answers are ruled out. Our results confirm that sparsity and the kind of asymmetry prevalent in the model can predict which ZDD variant is most beneficial.

In this article we only considered S5. SMCDEL also provides modules for K and in further experiments we compared the sizes of ZDDs and BDDs of the state law of *belief* structures. As an example we used the famous Sally-Anne false belief task. The results were similar to those here and can be found in [17].

Future work An obvious limitation is that we only compared memory and not computation time. The size of a decision diagram correlates with the computation time needed to build it. But the step-wise construction techniques in SMCDEL are slower for ZDDs than for BDDs. For example, to compute the Sum and Product result we rather convert each state law BDD to ZDDs instead of computing ZDDs directly. Before a meaningful comparison of computation time can be done, the construction methods for ZDDs need to be further optimized.

We found some indicators which elimination rule is most compact in which case, but a more general approach to formalise domain knowledge and use it to make a correct prediction would be a powerful tool.

Acknowledgements This work is based on the master thesis [17] by the first author, written at the University of Groningen and co-supervised by Rineke Verbrugge and the second author.

We thank the TARK reviewers for their careful reading and helpful comments on this article.

References

[1] Johan van Benthem, Jan van Eijck, Malvin Gattinger & Kaile Su (2018): *Symbolic Model Checking for Dynamic Epistemic Logic — S5 and Beyond*. *Logic and Computation* 28(2), pp. 367—402, doi:10.1093/logcom/exx038.

[2] Randal E Bryant (1986): *Graph-based algorithms for boolean function manipulation*. *IEEE Transactions on Computers* 100(8), pp. 677–691, doi:10.1109/TC.1986.1676819.

[3] Randal E. Bryant (2018): *Binary Decision Diagrams*. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith & Roderick Bloem, editors: *Handbook of Model Checking*, Springer, pp. 191–217, doi:10.1007/978-3-319-10575-8_7.

[4] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill & Lain-Jinn Hwang (1992): *Symbolic model checking:* 10^{20} *states and beyond*. *Information and computation* 98(2), pp. 142–170, doi:10.1016/0890-5401(92)90017-a.

[5] Tristan Charrier, Sébastien Gamblin, Alexandre Niveau & François Schwarzentruber (2019): *Hintikka's World: Scalable Higher-order Knowledge*. In: *IJCAI 2019*, pp. 6494–6496, doi:10.24963/ijcai.2019/934.

[6] Tristan Charrier, Sophie Pinchinat & François Schwarzentruber (2019): *Symbolic model checking of public announcement protocols*. *Logic and Computation* 29(8), pp. 1211–1249, doi:10.1093/logcom/exz023.

[7] David Chaum (1988): *The dining cryptographers problem: Unconditional sender and recipient untraceability*. *Journal of cryptology* 1(1), pp. 65–75, doi:10.1007/BF00206326.

[8] Hans van Ditmarsch, Wiebe van Der Hoek & Barteld Kooi (2007): *Dynamic Epistemic Logic*. Springer, doi:10.1007/978-1-4020-5839-4.

[9] Hans van Ditmarsch, Jan van Eijck & Rineke Verbrugge (2009): *Publieke werken—freudenthal's somen-productraadsel*. *Nieuw Archief voor Wiskunde* 10(2), pp. 126–131. Available at `https://www.nieuwarchief.nl/serie5/pdf/naw5-2009-10-2-126.pdf`.

[10] Hans van Ditmarsch, Ji Ruan & Rineke Verbrugge (2007): *Sum and Product in Dynamic Epistemic Logic*. *Logic and Computation* 18(4), pp. 563–588, doi:10.1093/logcom/exm081.

[11] Jan van Eijck & Simona Orzan (2007): *Epistemic verification of anonymity*. *Electronic Notes in Theoretical Computer Science* 168, pp. 159–174, doi:10.1016/j.entcs.2006.08.026.

[12] Sébastien Gamblin, Alexandre Niveau & Maroua Bouzid (2022): *A Symbolic Representation for Probabilistic Dynamic Epistemic Logic*. In: *AAMAS 2022*, pp. 445–453. Available at `https://dl.acm.org/doi/abs/10.5555/3535850.3535901`.

[13] Donald E. Knuth (2011): *The Art of Computer Programming, volume 4A: Combinatorial Algorithms, Part 1*. Addison-Wesley.

[14] John E Littlewood (1953): *A Mathematician's Miscellany*. Methuen and Company Limited.

[15] Guanfeng Lv, Kaile Su & Yanyan Xu (2013): *CacBDD: A BDD package with dynamic cache management*. In: *Computer Aided Verification*, Springer, pp. 229–234, doi:10.1007/978-3-642-39799-8_15.

[16] Kenneth L McMillan (1993): *Symbolic model checking*. Springer, doi:10.1007/978-1-4615-3190-6.

[17] Daniel Miedema (2022): *Zero-suppression Decision Diagrams versus Binary Decision Diagrams on Dynamic Epistemic Logic Model Checking*. Master's thesis, University of Groningen. Available at `https://fse.studenttheses.ub.rug.nl/27287/`.

[18] Shin-ichi Minato (1993): *Zero-suppressed BDDs for set manipulation in combinatorial problems*. In: *Proceedings of the 30th international Design Automation Conference*, pp. 272–277, doi:10.1145/157485.164890.

[19] Shin-ichi Minato (2001): *Zero-suppressed BDDs and their applications*. *International Journal on Software Tools for Technology Transfer* 3(2), pp. 156–170, doi:10.1007/s100090100038.

[20] Fabio Somenzi (2012): *CUDD: CU decision diagram package*. Available at `http://vlsi.colorado.edu/~fabio/CUDD/`. Version 2.5.0.

[21] K. Su, A. Sattar, G. Lv & Y. Zhang (2009): *Variable Forgetting in Reasoning about Knowledge*. *Journal of Artificial Intelligence Research* 35, pp. 677–716, doi:10.1613/jair.2750.

Appendix

The ZDD encoding of knowledge structures has been integrated into SMCDEL itself. All our results can be reproduced using the Haskell Tool *Stack* from `https://haskellstack.org` as follows.

```
git clone https://github.com/jrclogic/SMCDEL
cd SMCDEL
git checkout zdd-experiments
stack bench --no-run-benchmarks # build but do not run yet
stack bench smcdel:bench:sizes-muddychildren
```

```
stack bench smcdel:bench:sizes-diningcryptographers
stack bench smcdel:bench:sizes-sumandproduct
```

The last three commands will create `.dat` files containing the results. On a system with a 4.8 GHz CPU the last three commands above take approximately 10 seconds, one minute and three hours.

We include the results for Dining Crytographers and Sum and Product here, but omit the (several pages long) results for the Muddy Children.

Results for Dining Cryptographers

```
# Note: round -1 indicates the average.
```

n	m	round	BDD	BDDc	T0	T1	E0	E1
3	1	0	9	7	9	13	11	13
3	1	1	15	12	10	19	14	19
3	1	2	21	19	12	25	17	25
3	1	-1	11.25	9.5	7.75	14.25	10.5	14.25
5	1	0	13	11	18	26	22	26
5	1	1	25	20	21	38	29	38
5	1	2	41	37	29	54	40	54
5	1	3	64	61	44	77	57	77
5	1	4	98	96	68	111	82	111
5	1	-1	60.25	56.25	45.0	76.5	57.5	76.5
7	1	0	17	15	31	43	37	43
7	1	1	35	28	36	61	48	61
7	1	2	61	55	50	87	67	87
7	1	3	102	97	79	128	100	128
7	1	4	170	166	133	196	157	196
7	1	5	285	282	228	311	254	311
7	1	6	479	477	388	505	415	505
7	1	-1	287.25	280.0	236.25	332.75	269.5	332.75
9	1	0	21	19	48	64	56	64
9	1	1	45	36	55	88	71	88
9	1	2	81	73	75	124	98	124
9	1	3	140	133	118	183	147	183
9	1	4	242	236	202	285	236	285
9	1	5	423	418	359	466	397	466
9	1	6	747	743	645	790	686	790
9	1	7	1326	1323	1156	1369	1199	1369
9	1	8	2352	2350	2052	2395	2096	2395
9	1	-1	1344.25	1332.75	1177.5	1441.0	1246.5	1441.0
11	1	0	25	23	69	89	79	89
11	1	1	55	44	78	119	98	119
11	1	2	101	91	104	165	133	165
11	1	3	178	169	161	242	198	242
11	1	4	314	306	275	378	319	378
11	1	5	561	554	494	625	544	625
11	1	6	1015	1009	906	1079	961	1079
11	1	7	1852	1847	1671	1916	1730	1916
11	1	8	3392	3388	3077	3456	3139	3456
11	1	9	6211	6208	5636	6275	5700	6275
11	1	10	11333	11331	10244	11397	10309	11397
11	1	-1	6259.25	6242.5	5678.75	6435.25	5802.5	6435.25
13	1	0	29	27	94	118	106	118
13	1	1	65	52	105	154	129	154
13	1	2	121	109	137	210	172	210
13	1	3	216	205	208	305	253	305
13	1	4	386	376	352	475	406	475
13	1	5	699	690	633	788	695	788
13	1	6	1283	1275	1171	1372	1240	1372
13	1	7	2378	2371	2190	2467	2265	2467
13	1	8	4432	4426	4106	4521	4186	4521
13	1	9	8277	8272	7687	8366	7771	8366
13	1	10	15449	15445	14341	15538	14428	15538

13	1	11	28764	28761	26628	28853	26717	28853
13	1	12	53342	53340	49156	53431	49246	53431
13	1	-1	28860.25	28837.25	26702.0	29149.5	26903.5	29149.5

Results for Sum and Product

```
# Note: round -1 indicates the average.
```

n	round	BDD	BDDc	T0	T1	E0	E1
50	0	5032	5030	3082	5060	2505	5060
50	1	697	696	458	726	288	724
50	2	547	546	361	576	221	574
50	3	1	1	0	31	0	31
50	-1	1569.25	1568.25	975.25	1598.25	753.5	1597.25
64	0	7916	7914	4247	7944	4602	7944
64	1	930	929	614	959	381	957
64	2	760	759	501	789	308	787
64	3	1	1	0	31	0	31
64	-1	2401.75	2400.75	1340.5	2430.75	1322.75	2429.75
75	0	12534	12532	7847	12566	5986	12566
75	1	1274	1273	900	1307	456	1305
75	2	939	938	658	972	335	970
75	3	35	34	27	68	12	66
75	-1	3695.5	3694.25	2358.0	3728.25	1697.25	3726.75
100	0	22438	22436	13514	22471	11279	22471
100	1	2289	2288	1567	2323	870	2321
100	2	1594	1593	1087	1628	596	1626
100	3	36	35	28	70	12	68
100	-1	6589.25	6588.0	4049.0	6623.0	3189.25	6621.5
125	0	33826	33824	18476	33859	19074	33859
125	1	3149	3148	2095	3183	1262	3181
125	2	2101	2100	1383	2135	835	2133
125	3	36	35	28	70	12	68
125	-1	9778.0	9776.75	5495.5	9811.75	5295.75	9810.25
128	0	35315	35313	18823	35348	20401	35348
128	1	3149	3148	2095	3183	1262	3181
128	2	2101	2100	1383	2135	835	2133
128	3	36	35	28	70	12	68
128	-1	10150.25	10149.0	5582.25	10184.0	5627.5	10182.5
150	0	55028	55026	33874	55065	26559	55065
150	1	5147	5146	3526	5185	1938	5183
150	2	3354	3353	2261	3392	1267	3390
150	3	40	39	32	78	12	76
150	-1	15892.25	15891.0	9923.25	15930.0	7444.0	15928.5
175	0	73233	73231	43763	73270	36869	73270
175	1	6753	6752	4635	6791	2549	6789
175	2	4265	4264	2893	4303	1599	4301
175	3	40	39	32	78	12	76
175	-1	21072.75	21071.5	12830.75	21110.5	10257.25	21109.0
200	0	98044	98042	58134	98082	49615	98082
200	1	8498	8497	5929	8537	3096	8535
200	2	5275	5274	3666	5314	1877	5312
200	3	41	40	33	80	12	78
200	-1	27964.5	27963.25	16940.5	28003.25	13650.0	28001.75
225	0	121863	121861	69555	121901	64632	121901
225	1	10103	10102	6910	10142	3827	10140
225	2	6284	6283	4289	6323	2317	6321
225	3	41	40	33	80	12	78
225	-1	34572.75	34571.5	20196.75	34611.5	17697.0	34610.0
250	0	148149	148147	80231	148187	83173	148187
250	1	14131	14130	8991	14170	6071	14168
250	2	8664	8663	5470	8703	3659	8701
250	3	41	40	33	80	12	78
250	-1	42746.25	42745.0	23681.25	42785.0	23228.75	42783.5
256	0	154815	154813	81895	154853	88925	154853

256	1	14992	14991	9616	15031	6371	15029
256	2	9084	9083	5787	9123	3786	9121
256	3	41	40	33	80	12	78
256	-1	44733.0	44731.75	24332.75	44771.75	24773.5	44770.25
275	0	203227	203225	123011	203269	98715	203269
275	1	18794	18793	12876	18837	7046	18835
275	2	11435	11434	7808	11478	4189	11476
275	3	45	44	37	88	12	86
275	-1	58375.25	58374.0	35933.0	58418.0	27490.5	58416.5
300	0	238864	238862	144850	238906	116069	238906
300	1	21339	21338	14568	21382	8066	21380
300	2	12671	12670	8634	12714	4662	12712
300	3	45	44	37	88	12	86
300	-1	68229.75	68228.5	42022.25	68272.5	32202.25	68271.0
325	0	277256	277254	165770	277298	137410	277298
325	1	24822	24821	16902	24865	9451	24863
325	2	14341	14340	9749	14384	5305	14382
325	3	45	44	37	88	12	86
325	-1	79116.0	79114.75	48114.5	79158.75	38044.5	79157.25
350	0	318340	318338	187632	318382	160813	318382
350	1	29838	29837	19932	29881	11776	29879
350	2	17313	17312	11500	17356	6686	17354
350	3	45	44	37	88	12	86
350	-1	91384.0	91382.75	54775.25	91426.75	44821.75	91425.25