Decision diagrams as plans: Answering observation-grounded queries

Dylan A. Shell Jason M. O'Kane

***Abstract*— We consider a robot that answers questions about its environment by traveling to appropriate places and then sensing. Questions are posed as structured queries and may involve conditional or contingent relationships between observable properties. After formulating this problem, and emphasizing the advantages of exploiting deducible information, we describe how non-trivial knowledge of the world and queries can be given a convenient, concise, unified representation via reduced ordered binary decision diagrams (BDDs). To use these data structures directly for inference and planning, we introduce a new product operation, and generalize the classic dynamic variable reordering techniques to solve planning problems. Also, finally, we evaluate optimizations that exploit locality.**

I. Introduction

This paper explores the implications of asking robots questions, rather than telling them what to do. This model of interaction is appropriate when robots are being used to retrieve information about their world. Existing techniques for planning in robot data acquisition settings, such as informative path planning [22] and information gathering [9], are effective at collecting large quantities of desired data. But, while the datasets they produce tend to be useful for aggregated statistical analysis, when one is concerned with finer concepts involving richer semantic relationships — particularly when efficient execution is a concern— such methods may be too indiscriminate for those needs.

We formulate a class of useful and interesting problems based on directly posing specific queries to the robot: the intention is that with an exact statement of what is desired, this can be turned into opportunities for efficiency. There is the potential for the robot to exploit structure in the world to draw inferences that can save work. We show how such inference is possible (and useful for question-answering), by combining knowledge of regularity in the world with structured queries, both expressed in declarative form.

To that end, this paper formulates a new kind of planning problem in which the goal is to make a set of measurements that suffice to answer a specific YES/NO query. The paper also describes our initial experimentation with a new declarative language called Structured Robot Query Language (SRQL, meant to be pronounced like 'circle') that is rich enough to express both the regularity that structures the robot's world and the non-trivial queries that robots can be tasked to resolve within that world. Throughout the paper, fragments of SRQL code appear in green.

The authors are with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX, USA. {dshell, jokane}@cs.tamu.edu This material is based upon work supported by the NSF under Grants 1849249 & 1849291.

We also describe a method that forms plans for robots to answer SRQL queries. The method works by representing both the query and the world knowledge as binary decision diagrams, performing a novel conditioning operation on these diagrams, and finally optimizing the resulting plan via a generalization of Rudell's dynamic variable ordering algorithm. In particular, we introduce a block-oriented sifting optimization to Rudell's algorithm that exploits the specific local structure in these problems. Our results show that this optimization reduces the computational cost by two orders of magnitude.

The setting we consider, in which the robot has only partial information about the status of the world, is a variation upon a common refrain in robotics research [10], [13]. The important feature here is that, though the system's state is only partially observable, that partial observability takes a particular projective form, with certain elements fully known, and other elements that are observable only within certain locations. In that sense, the present setting dovetails closely with the recently-proposed locally-observable Markov decision process (LOMDP) model, in which uncertainty arises from limits on sensor range [11].

Finally, at the heart of our problem is the notion that a robot must choose where, and in what sequence, observations should be made to capture data satisfying certain specifications. Thus, strong parallels exist between our work and existing methods for active sensing [14], [18], [19], sensor selection [8], [15], [20], [21], [23], [26], and triage of captured data [7]. In that vein, a family of close cousins to the present work consider embodied question answering (EQA) [3], [25], including in manipulation contexts [5]. Our approach complements that line of work —which is strongly focused on perceptual challenges— by considering how knowledge about the structure of the world can contribute to efficient query-resolving plans. Although our syntax will resemble first-order logic [17], the focus is on generating branching trees of observations. This is distinct, also, from trees used predominantly for action, e.g., [12].

II. Background: Binary decision diagrams

A Binary Decision Diagram (BDD) is a directed acyclic graph (DAG) data structure that encodes a boolean function on a collection of boolean variables [1], [2]. Variables are represented in the graph as vertices, with a pair of edges departing every non-leaf vertex, each edge being associated with a truth value for the vertex's variable. There are two leaf vertices, labeled TRUE and FALSE, and all paths traced via edges reach one of these leaves. An encoding of a function has a root vertex and, for any particular assignment of truth

values to the variables, one evaluates the function by starting from the root, selecting the outgoing edge on the basis of the variable's value, and proceeding to the next vertex. One continues this way until a leaf is reached. That leaf's label is the value of the function for the given variable assignments.

Being a DAG, any BDD can be ordered topologically in what are usually termed levels. An *ordered BDD* imposes the requirement that any variable appear in most one level. A *reduced ordered BDD* is an ordered BDD with the added property that isomorphic subgraphs —essentially identical expressions— are collapsed into one copy, with vertices having both edges arriving at the same vertex being elided. The ordering helps simplify the process of matching subgraphs, so is a practical way to realize the opportunities for compression. An important consequence of these requirements on reduced ordered BDDs is that, when the variable ordering is fixed, there is precisely one representation of a given boolean function—for a given variable ordering, the reduced ordered BDD representation is canonical. However, the complexity of the representation, in terms of the number of vertices in the BDD, can depend heavily on the specific order selected.

To avoid this variance, the sifting algorithm of Rudell [16] aims to minimize reduced ordered BDD size by re-ordering the variables. It does this by picking a variable and then sweeping it backwards and forwards to find where the resulting BDD is smallest. A different variable is then picked, sifting repeated, and so on. At the core of the algorithm is a variable swapping primitive, because exchanging two variables in adjacent layers is a local operation on BDDs. For additional detail on BDDs, see Drechsler and Becker [6].

III. The problem: Planning for observation-grounded queries

A. Locations, transitions, properties

A robot operates in an environment characterized by a finite collection L of *locations*, one of which is designated as the initial location. The environment is modeled as a weighted directed graph, in which each vertex l corresponds to one of the locations, each edge $l \to m$ indicates direct traversability between a pair of locations, and the non-negative edge weights $c_{\text{trav}}(l,m)$ describe additive cost (in terms of time, energy, or other quantity of interest) for the robot to travel directly from l to m. We write $c^*_{\text{trav}}(l,m)$ to refer to the total cost of the shortest path along the graph from l to m, possibly including intermediate vertices. The locations, edges, and weights are assumed to be static and known to the robot.

Though this physical layout of the environment is assumed to be fixed and known to the robot, we are interested in scenarios where additional details about the environment and the objects therein are unknown. These details about the status of the environment are characterized by a collection P of boolean *properties*, each of which may be present or absent at each location. A given location may exhibit zero, one, or more properties.

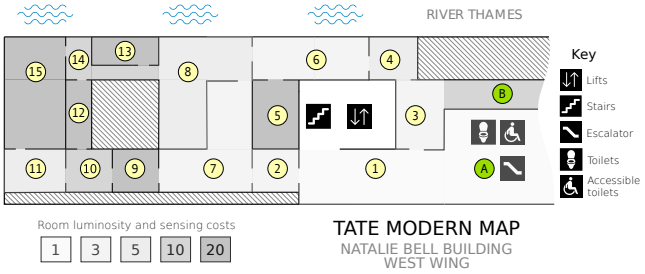


Fig. 1: Case study scenario: a mobile robot operates in part of the Tate Modern's 2nd floor to answer queries put to it about art. It does this by constructing plans, informed by the background knowledge available to it, that provide answers when executed.

Example 1. *Figure 1 is part of the floor plan of one of London's most vibrant art galleries: the Tate Modern. The mobile robot we envision, following a proud line of such robots [24], can roam this space autonomously. It is equipped with a camera and uses detectors —learned from data, hand-crafted, or both— capable of identify different aspects of art (e.g., whether an item is a sculpture or a painting, or perhaps a particularly famous piece, etc.) as well as properties of the room it occupies (size and shape). The two spots marked with the green nodes are the Atrium and the Biographic information display for the artist-in-residence; the other labeled locations are places from which art is visible. The robot moves from room to adjoining room, with traversal costs (c_{trav}) having been estimated from the distances between regions on the map.*

B. Situations, worlds, queries

A *situation* $s : L \times P \to \{\text{YES}, \text{NO}\}$ is defined as a function that maps location-property pairs to the tokens YES and NO, indicating that the property is present or absent in situation s. The space of all such situations is called the situation space $\mathcal{S}$. In most cases, structural knowledge about the environment will indicate that only certain types of situations may occur; a *world model* $\mathcal{W} \subseteq \mathcal{S}$, known to the robot, captures this knowledge. In SRQL, this world model is expressed indirectly as a collection of logical formulas that hold for precisely those situations in $\mathcal{W}$.

Example 2. *For the scenario of Example 1, we can express structure known about the world of the Tate. This includes geometric features: for instance, we declare that location* 11 *has the property of being a corner as* corner(loc11)*, and that* 12 *is not one with* ¬corner(loc12)*, etc. Similarly, we declare locations* 7 *and* 8 *to be the* L*-shaped ones,* ell-shaped(loc07) *and* ell-shaped(loc08)*. To state that both categories of visual form never appear in the same location and that some form will appear at each location, we write:*

$$\forall x\text{: } (painting(x) \text{ AND } \neg sculpture(x)) \text{ OR } (sculpture(x) \text{ AND } \neg painting(x)),$$

and the expression $\exists z$: sculpture(z)*, indicates, additionally, that there is at least one sculpture.*

One of the most famous paintings in the gallery is Andy Warhol's Marilyn Diptych*. Since it is a popular attraction, the curators must place it, not only prominently, but in*

sufficiently capacious room. Suitable rooms are locations 6*,* 7*,* 8*, and* 15*, expressed as:*

marilyn-diptych(loc06) OR *marilyn-diptych(loc07)* OR *marilyn-diptych(loc08)* OR *marilyn-diptych(loc15).*

Then, we express that this piece is a painting via the statement $\forall$ *x*: *marilyn-diptych(x)* $\Rightarrow$ *painting(x). Finally, we point out that the piece is unique in the following:*

$\forall$ *u,v,* $u \neq v$: *marilyn-diptych(u)* $\Rightarrow \neg$*marilyn-diptych(v). The set of situations that satisfy these statements taken in conjunction forms the world model* $\mathcal{W}$*.*

The robot, operating in a specific static but unknown situation $s \in \mathcal{W}$, seeks to answer a boolean *query*, seen as $\mathcal{Q} \subseteq \mathcal{S}$, about the situation. That is, the robot seeks to determine, based on some combination of the world model $\mathcal{W}$ and its own observations of the world, whether or not $s \in \mathcal{Q}$.

Example 3. *With the facts about the world in place (recall Example 2), it is possible to begin to ask questions about the world that the robot inhabits. Our first question is 'Are there any paintings to see?' It can be expressed directly in SRQL code, thus:*

Query: $\exists$ *x*: *painting(x)* ..... *(0)*

Example 4. *Or suppose we wish to see the* Marilyn*. We might ask whether it is in location* 15*, as follows:*

Query: marilyn-diptych(loc15) ..... *(1)*

Example 5. *A more interesting query might ask whether the* Marilyn *is in an* L*-shaped room. This query is easy to pose.*

Query: $\forall$ *x*: *marilyn-diptych(x)* $\Rightarrow \neg$*ell-shaped(x)* ..... *(2)*

To accomplish the goal of answering the given query, we assume that the robot is equipped with suitable sensors to observe which properties are extant at its current location. However, the robot must intentionally observe a location, incurring an *observation cost* $c_{\text{obs}}(l)$ to do so. Note that observation costs may differ according to the robot's location.

Example 6. *For our robot in the Tate Modern, the observation cost is a function of the room's lighting level (which influences duration that the robot must wait for an in-focus image, operate its shutter, and post-process raw camera images).*

Taken together, we refer to the structure of locations, transitions, travel cost function, and observation cost function as the *location graph*, denoted G.

At each time step, the robot, from its current location l, may choose either (i) to *travel* along some outgoing edge $l \to m$, deterministically reaching location m whilst incurring cost $c_{\text{trav}}(l,m)$, or (ii) to *observe* location l at cost $c_{\text{obs}}(l)$, thereby obtaining the set $\{p : P \mid s(l,p) = \text{YES}\}$, a complete and accurate reporting of the properties present at l in the prevailing situation s.

C. Plans

In this setting, a plan may be expressed as a BDD, in which each internal vertex asks about one specific location l and specific property p, leading to two children corresponding to $s(l,p) = \text{YES}$ and $s(l,p) = \text{NO}$ answers to that single-property question. Each leaf is labeled with an answer to the overall query, either YES or NO.

An essential question is whether a plan of this type will guide the robot to a correct decision about its query $Q \in \mathcal{Q}$. The next definitions formulate this concept precisely.

Definition 1. The *outcome* of a plan π in situation s is the label—either YES or NO—of the leaf reached by tracing from the root of p along the edges in accordance with the properties present in s.

Definition 2. A plan π *correctly resolves* a query $\mathcal{Q}$ in a world $\mathcal{W}$ if, for every situation $s \in \mathcal{W} \cap \mathcal{Q}$, the outcome of p in s is YES, and for every situation $s \in \mathcal{W} \cap (\mathcal{S} \setminus \mathcal{Q})$, the outcome of p in s is NO.

Notice that Definition 2 is silent about how the plan should behave in situations that are outside of the world model $\mathcal{W}$.

Beyond mere correctness, the robot should endeavor to find plans that resolve their queries in a cost-efficient way. For a given plan π, executing within a given location graph G, consider a simple path from the root to some leaf. Such a path can be segmented into blocks of one or more consecutive question nodes all at the same location, punctuated by travel from one location to the next. The cost of executing such a path is the sum of the observation costs $c_{\text{obs}}(l)$ for each block (determined by the location l shared by the nodes in that block), plus the sum of optimal travel costs $c^*_{\text{trav}}(l,l')$ from the initial state to the location of the first block, and between successive blocks. Intuitively, this represents the combined travel and execution costs for a robot carrying out the instructions encoded in the plan. Based on this concept of the cost of an execution, the extension to the cost of an entire plan is straightforward.

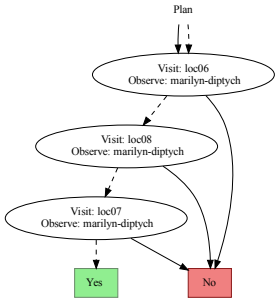
Definition 3. The *execution cost* of a plan π is the maximum (i.e. worst case) across the path execution costs for each simple path from root to leaf within π.

The main problem addressed in the remainder of this paper is as follows:

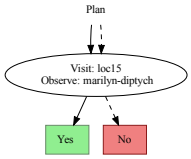
Given: A location graph G, a world model $\mathcal{W}$, and a query $\mathcal{Q}$.

Compute: A plan π that correctly resolves $\mathcal{Q}$ in $\mathcal{W}$, with minimal execution cost in G.

Example 7. *Figure 2 illustrates the idea of BDD-based plans by showing a pair of plans, each of which happens to be produced by algorithm introduced below in Section V. Both of these plans correctly resolve Query 1. Figure 2a shows a plan that differs from the obvious plan of going to location* 15 *and observing. Instead, this plan, when started from the Atrium as the initial location, has cost* 13.7 *% less than driving to location* 15*.*



(a) A plan to answer Query 1.



(b) Another plan to answer the same query, but optimized for when the robot starts at location 11 instead.

Fig. 2: Two plans generated in response to SRQL query marilyn-diptych(loc15); that have differing execution costs. Solid and dashed edges represent YES and NO respectively. When the robot starts in the Atrium, plan (a) has execution cost 208.25 units, and plan (b) cost 241.44 units. When it starts at location 11 (a) costs 277.74 units, and (b) 62.33 units.

We can also ask the same question, but now with the robot positioned initially at location 11. *(For future reference, we shall refer to this as Query 1b). In this case, the simpler plan shown in Figure 2b is now optimal, owing to the lower travel distance to location* 15.

Example 8. *What about Query 0? In that case, the execution cost of the optimal plan is 0, requiring no travel and no observations! Why? Notice that the knowledge about the world encoded in $\mathcal{W}$ tells us there is a painting, specifically Warhol's* Marilyn. *Thus, when asked if there are any paintings to be seen, the robot doesn't have to move anywhere to answer in the affirmative. The robot does not know at which location the painting might be, but that wasn't specifically what was asked. This example demonstrates, in an extreme case, the value of the world model $\mathcal{W}$: structure in the world can be exploited to avoid needless effort by the robot. The algorithm proposed below is designed to automatically detect and utilize this structure, even in milder forms.*

IV. PLANNING VIA THE CONDITIONED PRODUCT BDD

This section outlines an algorithm to solve the planning problem introduced in Section III. The key idea is to describe the world model $\mathcal{W}$ and query $\mathcal{Q}$ both as boolean functions, and to represent these as ordered reduced BDDs with variables for each location-property pair. The world BDD traces to TRUE for precisely the situations that are possible. The query BDD arriving at TRUE implies the answer to the query is in the affirmative; FALSE means the answer is negative.

Observe that though both $\mathcal{W}$ and $\mathcal{Q}$ can be expressed in the same data structure, their interpretations differ. In $\mathcal{W}$ we know that, to describe a situation, the tracing will arrive at TRUE. In other words, we use the BDD to characterize the set of inputs that make the overall boolean formula true. In contrast, for $\mathcal{Q}$, one expects that in most interesting cases both answers would possible. We need to combine the two BDDs, but *condition* on the truth of one of them.

As a result, the broad idea of the algorithm is to compute this combined BDD through a sort of product graph operation, and then condition the combined graph upon the $\mathcal{W}$ part tracing to TRUE. In detail, consider that we have a BDD and, like the world $\mathcal{W}$, we know that tracing must arrive at TRUE. Suppose we are tracing within it and we arrive at a vertex for variable v (for property p_v at location l_v). If one of the vertex's edges, say the YES edge, arrives at the FALSE leaf then, as we require $\mathcal{W}$ to be true, we deduce that v must be false (i.e., property p_v cannot hold at l_v). This conclusion does not require any measurement:— v's value can be known for free at this point in the BDD or, because the variables that precede it in the DAG have already discerned sufficient information to determine v, we say it can be 'inferred via conditioning.' Symmetrically, if it were the NO edge arriving at FALSE, we could conclude that property p_v does hold.

The general rule to condition on the truth of a BDD is as follows: when a variable's edge leads to a sub-DAG and all tracings forward, passing only through variables whose values can be inferred, always arrive at the FALSE leaf, then that variable's value can be inferred. And specifically, if the edge was a YES edge, the variable is false; a NO edge means the variable is true. One identifies this property computationally via a recursive procedure that checks the property on the two children. The previous paragraph expressed the base case, viz. an edge leading to FALSE. Note that some variable might have, from both of its edges, a sub-DAG that arrives at TRUE after passing through downstream variables whose values can be inferred. Such a variable fails to meet the criterion to be inferred itself: that variable's value —though irrelevant for evaluating the boolean function expressed in the sub-DAG— is unknown. This reasoning motivates the following construction.

Construction. The *output-conditioned product* of two ordered BDDs with compatible variable orderings is the ordered BDD constructed as follows:

1) First 'complete' each BDD, using the levels from the union of their variables: for any edge $v_p \longrightarrow v_s$, which skips levels, say v_q and v_r, ordered $v_p \prec v_q \prec v_r \prec v_s$, reintroduce those variables by joining $v_p \longrightarrow v_q$, and both $v_q \dashrightarrow v_r$ and $v_q \longrightarrow v_r$, and both $v_r \dashrightarrow v_s$ and $v_r \longrightarrow v_s$. (And similarly for $v_p \dashrightarrow v_s$.)

2) Form a graph product of the two completed BDDs in which each vertex corresponds to an ordered pair of vertices from the original BDDs, starting from a pair with both roots $(v_{\text{root}}, v'_{\text{root}})$, and then tracing edges forward, making pairs for the children so obtained. The result is no longer a binary DAG as, in general, there will be four out-edges from each internal node bearing the labels (YES, YES), (YES, NO), (NO, YES), and (NO, NO). As the variable ordering is compatible (and the prior step ensured both BBDs have all variables), the product retains its binary form without any restoration needed.

3) The resulting product is then conditioned using the method described in Section IV. Since it consists of vertices made up of variable pairs, we ask if the sub-BDD arrives, not at the FALSE leaf, but at some $(\text{FALSE}, \cdot)$ node. All nodes with values that can be inferred are pruned. If

inference indicates that the value must be x, then bypass the node by rewiring the incoming edge(s) directly to the child reached by the x-labeled edge.

4) At this point we have obtained an ordered BDD, but some structure may be repeated and it can contain superfluous queries. To remove this redundancy, the final step is to form a reduced ordered BDD from the result.

If the inputs are ordered BDDs encoding $\mathcal{W}$ and $\mathcal{Q}$, the preceding construction yields a reduced ordered BDD where the leaves (TRUE, TRUE) and (TRUE, FALSE) correspond, to the answers of YES and NO to the query, respectively. The product's construction omits nodes for variables with values that can be inferred on the basis of $\mathcal{W}$, but retains nodes for factors which affect the query's answer.

The construction's correctness can be established via an argument that takes any situation $s \in \mathcal{W}$, and traces, in conjunction, the world and query BDDs on one hand, and the output-conditioned product on the other. The resulting structure constitutes a plan π that correctly resolves $\mathcal{Q}$ in $\mathcal{W}$.

One final note: our implementation does not carry out the four steps stage-wise, directly one after the other, as described here. Instead, repeated traversals can be avoided by performing the steps simultaneously. Our implementation uses bookkeeping to be equivalent to step 1 without introducing nodes or actually modifying edges in the pruning of step 3, while incrementally doing a depth-first walk of both BDDs in concert, to build the reduced ordered BDD in postorder fashion. Our code is based on version 0.5.7 of the open-source `dd` package [4].

V. Finding better plans by sifting

Recall that the algorithm introduced in the previous section begins from ordered reduced BDDs for $\mathcal{W}$ and $\mathcal{Q}$, and that the size of a reduced ordered BDD is determined by the ordering of the variables. Even more relevant here is the fact that this ordering also has a strong impact on the execution cost of the resulting plan. As a result, we apply sifting methods, inspired by Rudell's algorithm, before the conditioned product step, in an effort to generate more efficient plans. In our case, we are primarily interested in plan execution cost; plan size is a (distant) secondary consideration.[1]

Specifically, we consider five different sifting algorithms. In each, for a certain order of variables, a plan's cost is evaluated by forming the output-conditioned product and computing the execution cost of Definition 3 on the result. Two are fairly straightforward:

1. Rudell's classic sifting algorithm [16], as a baseline. It re-orders individual variables to reduce the size of the BDD.
2. Rudell's classic sifting but with plan costs. This method operates on individual variables but constructs the product to evaluate whether a sifting move has improved worst-case cost of execution of the plan.

The remaining three variations exploit the specific structure of the execution cost that we seek to optimize.

Notice that two BDD variables that refer to properties at the same location incur no traversal cost to measure one after the other. Thus, one expects variables associated with the same location to appear in blocks in cost-minimal plans. This motivates the concept of *sifting-by-block*. The idea is treat variables in per-location blocks, and to sift whole blocks at a time. This still uses the same underlying variable swapping primitive, but only the pays the computational price of forming an output-conditioned product after complete block movements. We consider three approaches based on this idea.

3. Sifting-by-block, which moves whole blocks of variables (those tied to a single location) together before constructing the product to evaluate plan execution cost.
4. A version of sifting-by-block that, once plan execution costs have been reduced, then sifts variables (restricted to only within blocks) in order to reduce BDD size.
5. A version like the previous, but which first optimizes size (with sifting within blocks) and only then uses sifting-by-block to optimize plan execution cost.

Section VI presents a quantitative evaluation and analysis of these five variations.

VI. Case Study: Revisiting the Tate

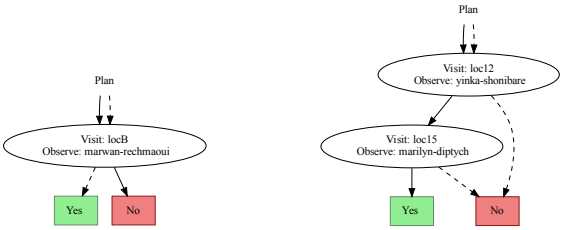
Examples 1–4 introduced a problem domain in which the algorithms of Sections IV and V can be exercised.

First, some illustrative examples based on variant 3 of the sifting algorithm. One illustration has appeared already: Figure 2 shows two plans generated by it for Query 1.

For Query 2, the planner tends to produce one of three plans, either visiting location 6 followed by location 15 (with a cost of 244.44) or location 8 followed by 7 (with a cost 205.25 units) or, the majority of the time (59 %, being 2.7× more likely to be generated than the next most frequently returned plan), location 7 followed by 8 (costing 170.02 units). Now, suppose the two central L-shaped rooms have particularly high foot traffic. This might cause the robot to spent a great deal of time avoiding visitors, incurring greater costs for observations in those places. This is easily captured by modifying their respective observation costs. Doing so and then asking the same question (dubbed Query 2b) causes the 6→15 route, with unchanged cost 244.44, to be the preferred choice, being found 65 % of the time (being 1.9× more likely than the next most frequently generated plan).

Perhaps, as you glance at the current exhibits at the Tate, you recall that two of your favorite living artists, Marwan Rechmaoui and Yinka Shonibare, may have art on display in the gallery. Rechmaoui's pieces (including *Beirut Caoutchouc*) as well as Shonibare's art (e.g., *Grain Weevil*) have been part of the collection in the past. Suppose you know that, if they are still available for viewing, that the former's sculpture would be at location 13, while latter's at location 12. You have conflicting reports that each is currently the artist-in-residence, meaning that if their art is

[1] A graph with fewer nodes means fewer observations, which does suggest so smaller size might correlate with lower execution cost: the discussion of empirical results in Section VI examines data regarding this question.



(a) An efficient plan for Query 3. **(b)** An efficient plan for Query 4.

Fig. 3: Plans for Query 3 and Query 4, the second being the logical conjunction of Queries 1 and 3.

on display, then they will be featured in the infographic at location B. These facts are expressed in SRQL as part of the world knowledge as follows:

$$\Big((\text{yinka-shonibare(locB)} \text{ AND } \text{yinka-shonibare(loc12)}) \text{ OR } (\neg\text{yinka-shonibare(locB)} \text{ AND } \neg\text{yinka-shonibare(loc12)})\Big) \text{ AND } \Big((\neg\text{marwan-rechmaoui(locB)} \text{ AND } \neg\text{marwan-rechmaoui(loc13)}) \text{ OR } (\text{marwan-rechmaoui(locB)} \text{ AND } \text{marwan-rechmaoui(loc13)})\Big),$$

and also, with just a single artist-in-residence at any time, only one will be featured:

$$\text{yinka-shonibare(locB)} \text{ XOR } \text{marwan-rechmaoui(locB)}.$$

Then a query of interest just asks the robot whether we'll be able to see Yinka Shonibare's art:

$$\text{Query:} \quad \exists\, x\colon \text{yinka-shonibare}(x) \qquad (3)$$

Using this information, the planner gives a simple strategy for the robot (see Figure 3a), with cost 87.37 units: drive to B and determine whose biographic information is featured. Going directly to location 12 would incur cost 217.00.

But then, being pushed for time, we want to know if we can see the *Marilyn* in location 15 as well as Shonibare's art. So we combine Queries 1 and 3 as follows:

$$\text{Query:} \quad \exists\, x\colon \text{yinka-shonibare}(x) \text{ AND } \text{marilyn-diptych(loc15)} \qquad (4)$$

The planner's solution for Query 4, with cost 280.63, is shown in Figure 3b. Noteworthy is that while Query 4 combines information needed to answer Queries 1 and 3, the optimal solution combines two plans that were dominated for either of those two.

While the previous discussion has showcased SRQL in some simple cases, it has not provided any results on comparative performance. We ran 5 different methods on 9 queries: namely Queries 0, 1, 1b, 2, 3b, 3, 4, along with two additions that produce more interesting plans:

$$\text{Query:} \quad \forall\, x\colon \text{sculpture}(x) \Rightarrow (\neg\text{ell-shaped}(x) \text{ OR } \text{corner}(x)) \qquad (5)$$

$$\text{Query:} \quad \exists\, x\colon \text{sculpture}(x) \text{ AND } \neg\text{ell-shaped}(x) \qquad (6)$$

We collected data on processing time, the resulting size of the BDD, and worst-case execution cost of plans for each of the methods. Each execution started with random initial conditions, with variables placed within location-oriented blocks and blocks randomly shuffled, but identical initial conditions tried on each of the methods. Each method was executed a total of 100 times. The processing times are summarized in Figure 4. Data on quality of the resulting plan

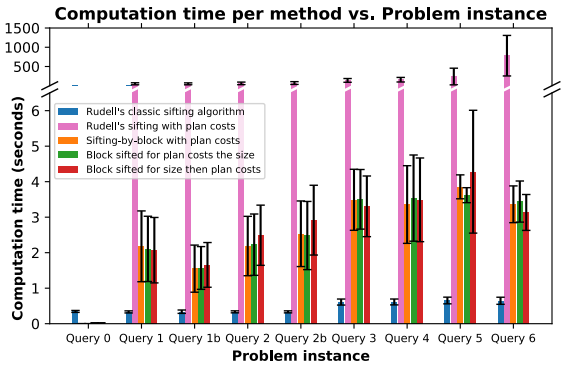


Fig. 4: Computation times for the sample queries in the Tate gallery.

Fig. 5: The size and quality of resulting plans for the sample queries, varying each method. Result size is measured in BDD vertices. Quality is measured by execution cost.

appears in Figure 5. (Recall that Query 0 is an exceptional case, where an empty plan suffices.)

Intuitively, Rudell's classic sifting algorithm might do well, because fewer variables does imply that less navigation would be required. But while it is efficient to compute (shown in Figure 4), its optimization of size, at best a surrogate to plan execution cost, is a poor solution. Indeed, Figure 5 shows that evaluating execution cost is important: The particular locations, not merely their cardinality, plays a decisive role. Classic sifting with plan costs produces high-quality results, but is slow. The last three methods incur no loss in quality, but with a two-order of magnitude reduction in computational time. We see that also compressing for size (before or after) makes only a marginal difference.

VII. Conclusion and Outlook

The present paper has advocated asking robots questions and having them combine what they know of the world, along with their ability to go out and probe it, as a way of being useful. This forms a sort of high-level abstraction, where the user is less concerned about *how* a robot acts than that it obtains the needed information. A core technical challenge is to find representations that are rich enough while not requiring impractical amounts of storage. Reduced ordered BDDs appear to be a promising candidate; future work will examine modifications (e.g., beyond Boolean to multi-valued expressions), optimizations (e.g., re-use for sets of queries), and specializations to better address their uses and abuses in planning.

REFERENCES

[1] Henrik R. Andersen. An introduction to binary decision diagrams. In *Lecture notes for "Efficient Algorithms and Programs"*. The IT University of Copenhagen, 1999.

[2] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–690, August 1986.

[3] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[4] The `dd` python package, available at https://github.com/tulip-control/dd.

[5] Yuhong Deng, Naifu Zhang, Di Guo, Huaping Liu, Fuchun Sun, Chen Pang, and Jing Pang. MQA: answering the question via robotic manipulation. In *Robotics: Science and Systems*, 2020.

[6] Rolf Drechsler and Bernd Becker. *Binary decision diagrams: theory and implementation*. Springer, 2013.

[7] Yogesh Girdhar and Gregory Dudek. Optimal online data sampling or how to hire the best secretaries. In *Canadian Conference on Computer and Robot Vision*, pages 292–298, 2009.

[8] A. Haji-Valizadeh and K. A. Loparo. Minimizing the cardinality of an events set for supervisors of discrete-event dynamical systems. *IEEE Transactions on Automatic Control*, 41(11):1579–1593, 1996.

[9] Geoffrey Hollinger and Gaurav Sukhatme. Sampling-based motion planning for robotic information gathering. In *Proceedings of Robotics: Science and Systems (RSS)*, Berlin, Germany, June 2013.

[10] Steven M. LaValle. Filtering and planning in information spaces. Technical report, Department of Computer Science, University of Illinois, 2009.

[11] Max Merlin, Neev Parikh, Eric Rosen, and George Konidaris. Locally observable Markov decision processes. In *ICRA 2020 Workshop on Perception, Action, Learning*, 2020.

[12] Petter Ögren and Christopher I Sprague. Behavior trees in robot control systems. *Annual Review of Control, Robotics, and Autonomous Systems*, 5:81–107, 2022.

[13] Robert Platt Jr, Russ Tedrake, Leslie Kaelbling, and Tomas Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Robotics: Science and Systems*, 2010.

[14] Hazhar Rahmani, Dylan A. Shell, and Jason M. O'Kane. Planning to

[23] Tae-Sic Yoo and S. Lafortune. NP-completeness of sensor selection problems arising in partially observed discrete-event systems. *IEEE Transactions on Automatic Control*, 47(9):1495–1499, 2002.

chronicle: Optimal policies for narrative observation of unpredictable events. *International Journal of Robotics Research*, 2022.

[15] Hazhar Rahmani, Dylan A Shell, and Jason M O'Kane. Sensor selection for detecting deviations from a planned itinerary. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and System*, pages 6511–6518, 2021.

[16] Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the IEEE International Conference on Computer Aided Design (ICCAD)*, pages 42–47, 1993.

[17] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach, Fourth Edition*. Pearson Education Limited, Harlow, United Kingdom, 2022.

[18] Allison Ryan and J Karl Hedrick. Particle filter based information-theoretic active sensing. *Robotics and Autonomous Systems*, 58(5):574–584, 2010.

[19] Paolo Salaris, Riccardo Spica, Paolo Robuffo Giordano, and Patrick Rives. Online optimal active sensing control. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 672–678, 2017.

[20] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.

[21] David Sears and Karen Rudie. Minimal sensor activation and minimal communication in discrete-event systems. *Discrete Event Dynamic Systems*, 26(2):295–349, June 2016.

[22] Amarjeet Singh, Andreas Krause, and William J Kaiser. Nonmyopic adaptive informative path planning for multiple robots. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1843–1850, 2009.

[24] S. Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. MINERVA: A Second-Generation Museum Tour-Guide Robot. In *ICRA*, volume 3, pages 1999–2005, 1999.

[25] Erik Wijmans, Samyak Datta, Oleksandr Maksymets, Abhishek Das, Georgia Gkioxari, Stefan Lee, Irfan Essa, Devi Parikh, and Dhruv Batra. Embodied question answering in photorealistic environments with point cloud perception. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[26] Xiang Yin and Stéphane Lafortune. A general approach for optimizing dynamic sensor activation for discrete event systems. *Automatica*, 105:376–383, 2019.