Task Value Calculus: Multi-objective Trade off Analysis using Multiple-Valued Decision Diagrams

Tyler Giallanza, Erik Gabrielsen, Michael A. Taylor, Eric C. Larson, and Mitchell A. Thornton
Darwin Deason Institute for Cybersecurity
Southern Methodist University, Dallas, Texas, USA
{tgiallanza, egabrielsen, taylorma, eclarson, mitch}@smu.edu

***Abstract*—Most multiple-objective optimization algorithms utilize continuous input variables. Given that many decision variables in common use-cases are discrete rather than continuous, we develop a multiple-objective optimization framework over discrete variables known as task value calculus (TVC). The underlying mathematical models in TVC utilize a multiple-valued algebraic framework where both the objective functions and the system or process structure models are represented as multiple-valued functions. TVC allows for fast multiple-objective optimization through the use of the multiple-valued decision diagram (MDD) data structure. The algorithms and structures internal to TVC are described and experimental results are provided. TVC is implemented with a simple graphical user interface making it suitable for use by both laypersons and domain experts.**

***Keywords*-multiple-valued logic; multiple-objective optimization; multiple-valued decision diagram**

I. Introduction

Teams in business and government often need to make decisions that satisfy multiple objectives, such as minimizing cost and maximizing probability of success. Since these objectives often conflict in complex ways, optimizing for multiple objectives is a difficult computational problem. Thus, a class of algorithms known as multiple-objective optimization algorithms exists to simplify the process of decision-making for the end-user. In this paper, we use Multiple-Valued Decision Diagrams (MDDs) as a basis in the implementation of a prototype optimization system to efficiently search for viable solutions in the presence of multiple objectives. A central goal of the prototype is to allow it to be used by laypersons with little knowledge of the underlying mathematical models and to produce visualizations of the trade-offs among differing solutions.

Multiple-objective optimization algorithms can be broadly classified into two categories: *a priori* and *a posteriori*. *A priori* algorithms assume a prior consideration of how important multiple objectives are relative to one another. The Weighted Sum Method, for example, assigns a specific weight to each objective function and adds the weighted objectives together to form a single objective function that can be optimized. In general, *a priori* methods map multiple-objective problems into a single-objective by assuming the relative importance of the objectives. These solutions present the user with a single solution rather than a variety of solutions to choose from.

The drawback of *a priori* methods is that they force the decision-maker to determine the importance of the objectives before being presented with the options. Especially for problems with many objectives, it can be difficult to exactly determine the relative importance of multiple objectives. Moreover, it is difficult to quantify how modifying the importance of one objective might affect the viable solutions.

Alternatively, *a posteriori* methods, do not assume that some objectives are more important than others. These algorithms, including NSGA-II [1], MOEA/D [2], PESA-II [3], and SPEA2 [4], try to present the user with a set of optimal solutions, known as the *Pareto-optimal set* or the *Pareto frontier*. Any solutions on the Pareto frontier represent optimal trade-offs between the various objective functions. Using *a posteriori* methods, the user is presented with many possible solutions and chooses the one best suited to their use-case.

To use an *a posteriori* algorithm, the user must first develop a mathematical function that relates the input variables to the various objectives. This process is trivial when known mathematical functions are good models of the problem. However, when approximations are required the process of developing a mathematical relationship between inputs and objectives can be a daunting task, especially for large systems or processes. Additionally, discrete variables, such as the number of employees to hire, often introduce nonlinearities in the objective function mapping process that complicate the ability of traditional optimization algorithms to search efficiently.

To address these issues, we introduce a system called *Task Value Calculus (TVC)*. TVC allows for fast optimization of problems that are discrete rather than continuous and does not require users to specify the exact mathematical modeling. TVC accomplishes these goals by reformulating the process of "objective function specification" into a simpler form—the user constructs a simple block diagram and the process structure function is automatically generated using MDDs. Rather than use models that relate inputs to objectives, TVC utilizes a table-based approach. The table-based approach is often simpler for the end-user to understand and better suited for discrete input variables. Internally, TVC makes use of the MDD data structure and associated graph-based algorithms to automatically specify the relationship between objective functions and choices for each node of the block diagram. An intuitive graphical user interface allows for the construction

of the block diagram representing the structure function and provides drop down table menus for specifying the objective functions. Thus, the system allows a non-technical person to use the TVC prototype with minimal training required.

II. Background Concepts

A. Multiple-Valued Decision Diagrams

Because TVC is designed for discrete inputs rather than continuous inputs, we are able to take advantage of the MDD data structure. MDDs enable solution space search algorithms to be efficiently implemented and they compress the storage space required for TVC. MDDs are data structures in the form of acyclic directed graphs that represent discrete-valued functions, $f : \mathbb{N}_r^n \rightarrow \mathbb{N}_r$, where $\mathbb{N}_r$ is the finite set of natural numbers, $\{0; 1; 2; 3; \cdots; r-2; r-1\}$. The support set of f is a set of independent discrete variables, $x_i \in \mathbb{N}_r$, with a cardinality of n. The discrete function f can be denoted as $f(x_1; x_2; \cdots; x_n)$.

MDDs are based upon the binary decision diagram (BDD) data structure originally proposed in [5] as first described in [6]. MDDs are implemented as software packages that support the graphical data structures as well as a set of operators for their manipulation. One of the first such software implementations is described in [7]. Our implementation was able to benefit from past research and results obtained from the exercise of implementing prior BDD packages, such as the "variable ordering" property and the "reduction rules" resulting in canonical representations [5]. However, the generalizations in MDDs as compared to BDDs required some new approaches for the internal data structures and their storage and manipulation. Therefore the implementation in [7] incorporated new algorithms for implementing the mathematical operations such as that described in [8] that support the *APPLY* function. The *APPLY* function is of the form $f_{new} = APPLY(f_1; f_2; <op>)$ where f_1 and f_2 represent operand MDDs, $<op>$ is a two-place operator, and f_{new} is a resulting MDD that models the function $f_{new} = f_1 <op> f_2$.

MDDs allow for a representative function f to be evaluated through a traversal of a single path from the initial vertex to a terminal vertex. The particular path traversed is governed by the valuation of each variable assignment as also labeled on the edges in the path. When a nonterminal vertex is encountered in a traversal, the outgoing edge annotated with that variable's valuation of interest is traversed until a terminal vertex is encountered. The constant value annotating the terminal vertex is then the value of the function.

In terms of mathematical manipulations of discrete functions represented by MDDs, it is generally the case that such operations are implemented as graph traversals, typically in a recursive fashion. From a theoretical point of view, a collection of operations resulting in a functionally complete set over an algebra is desirable. There are a variety of different algebras that have been formulated over discrete variables with a finite valuation set. One of the most common algebras is the Boolean or switching algebra, however, it is only functionally complete when the radix value $r = 2^k$ where k is some positive integer.

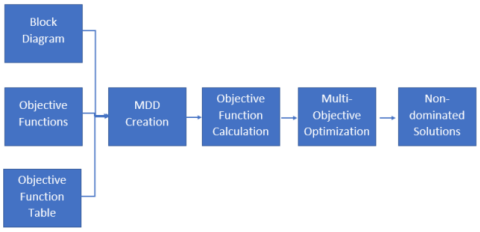

Fig. 1. Flowchart of the TVC process.

Therefore, MDDs are commonly used with a set of operators from an algebra other than the Boolean family of algebra since the radix is not, in general, a power of two. A general discussion of different finite discrete switching algebras and the theory of MDDs can be found in [9].

III. Task Value Calculus

Task Value Calculus utilizes the MDD data structures described above within a framework to quickly optimize multiple objectives in discrete systems. TVC utilizes a table-based approach rather than a mathematical modeling approach to define the inputs to the system. In the table-based approach used by TVC, three inputs are required: first, a block diagram that specifies the structure function; second, a listing of the objective functions used; and third, a table that relates the values of the input variables to local changes in the objective functions. These three inputs are used to form the underlying MDDs within TVC. Once the MDDs are created, they are traversed to find potential solutions that optimize the given objective functions. Figure 1 shows a flowchart of this process.

A. Block Diagram

A human consideration of a system or process as a collection of subsystems or subtasks that may have serial temporal dependencies is often easily represented as a block diagram. Through the creation of a block diagram by using a specialized graphical user interface (GUI), a user can specify the system or process of interest quickly and easily without requiring a detailed knowledge of the underlying mathematical models.

An example of a block diagram representing an extremely simple system or process of interest is shown in Figure 2. In this scenario, a company desires to deliver a package and can do so with either a truck "A" or truck "B." These two trucks have different requirements for loading the package at the initial location (the staging site) represented by "C" and different requirements for maintenance after the package is delivered that occurs at their refresh site denoted by "D." Furthermore, TVC always uses two blocks in the specification of a system or process block diagram structure labeled as "S" (start) and "E" (end). This ensures that the internal TVC processing has unique beginning and end points when it parses through the block diagrams and "S" and "E" are not part of the actual system or process being modeled. We note that a parallel structure is created to denote the choice of delivery trucks and the staging and refreshing sites are indicated as a

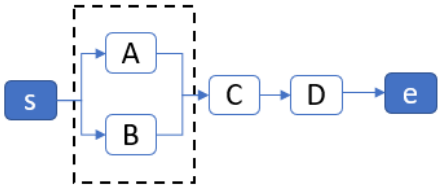

Fig. 2. Small Example Block Diagram

series structure. Furthermore, we note that the placement or order in which the parallel structure comprised of blocks "A" and "B" with respect to its location in the serial structure of "C" and "D" is not important. While logically, a user may choose to represent these in a time-dependent order, placing "C" first, then the parallel structure, followed by "D," the actual order is not relevant to the underlying calculations. This offers a degree of freedom that TVC can exploit in terms of ease of use in developing the structure diagram as well as with respect to reordering the variables in the MDD representing the structure function.

TVC extracts a representative system structure function from the block diagram that is represented internally as an MDD. One of the first uses of MDDs for this application was provided in [10] for cyber threat evaluation of system and since that time many other similar applications of MDDs for system and process analysis have been reported including [11]–[13]. This extraction process is enabled in TVC through the use of an underlying algebraic theory that allows the interconnections of a block diagram to be easily translated into a set of operators in a defining algebra. Specifically, TVC utilizes a discrete finite algebra based upon the theoretical work of E. Post [14] that is also used in [13]. The Post algebra can be considered to be a generalization of Boolean algebra since it results in a functionally complete algebra in the case of any positive radix $r > 3$. In Boolean algebra with radix $r = 2$, the functions AND, OR, and NOT form a functionally complete set with the constants $\{0; 1\}$. In the Post algebra underlying TVC, the corresponding functions MIN, MAX, and literal selection $\{J_i\}$ form a functionally complete set for any positive integer $r > 2$ and it becomes equivalent to the binary Boolean algebra when $r = 2$.

The MIN operator produces the minimum value among a set of operands and represents a multiplicative operation. Likewise, the MAX operator produces the maximum value among a set of operands and represents an additive operator. The literal selection operators comprise different unary operators denoted as $\{J_i\}$ where $i \in \{0; 1; : : : ; (r-1)\}$. Each $\{J_i(x)\}$ results in a value of $r-1$ when the $x = i$ and 0 when $x \neq i$. For example, in a radix-3 system, $J_2(x)$ results in a value of 2 when $x = 2$ and a value of 0 when $x = 0$ or $x = 1$.

TVC automatically translates the user-specified block diagram into an MDD structure function in a multi-step process. The block diagram is represented as a combination of series and parallel branches. These branches inherently represent MVL algebraic operators, where series connections

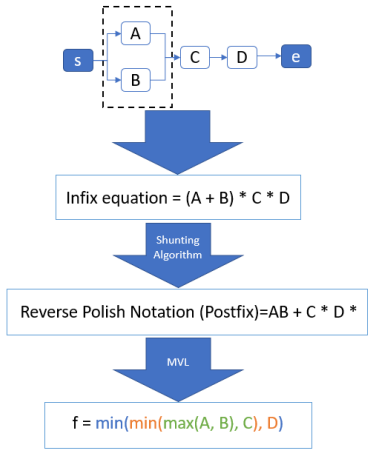

Fig. 3. Converting Block Diagram into MVL

are representative of a multiplicative MIN operator and parallel connections are representative of an additive MAX operator. Next, the block diagram is traversed and the series and parallel branches are recognized and converted into an MVL algebraic (infix) representation. The infix representation is converted to a postfix form using Dijkstra's shunting yard algorithm [15]. Finally, the postfix equation is converted into an MDD by processing the postfix representation and invoking the internal *APPLY* algorithm. The one-place literal select operators are useful when the block diagram has a selection block that allows one portion of the block diagram to be activated based upon a variable value as described in [13]. Figure 3 gives an example of converting a block diagram into MVL operators into postfix form.

The postfix form is represented internally in TVC using a LIFO stack of tokens representing variables and subexpressions. The stack content indicates the operands and operators used with a series of calls to the MDD *APPLY* algorithm. Initially, MDDs are created for each dependent variable in an expression. Next, the postfix expression is processed from left to right with operands being pushed to the stack. As soon as two operands are present in the stack, they are popped and an appropriate call to the APPLY algorithm is made using the operands and the appropriate algebraic operator indicated by the postfix representation. The result of *APPLY* algorithm is a pointer to the newly formed MDD and it is pushed onto the stack. This process continues until the entire postfix expression has been processed and the remaining entry in the stack is a single pointer to the MDD representing the entire postfix expression.

B. Objective Functions

Objective functions in TVC are designed to be simple to interface with the MDD structure representing the structure function and guide traversals of the MDD structure function such that they may be used to evaluate how the objective interacts with the system or process. We specify objective

functions as discrete-valued MVL functions in TVC that operate over the structure function initially represented by the entered block diagram. An objective function can operate on all of the variables in the block diagram or a subset thereof.

An optimization used in TVC is the pruning of variables that are not required in the evaluation of an objective as applied to the structure function. In the case where variables are connected in series, the objective function operates on all of the variables in the block diagram. When parallel connections are present in the block diagram, some of the variables are omitted from the computation such that only one variable from each parallel block is used. Variables are chosen to be omitted depending on whether the objective function is being minimized or maximized. Thus, an objective function that is maximized will use whichever variable in the parallel block that has the greatest value, and an objective function that is minimized will use whichever variable in the parallel block that has the least value.

In TVC we allow for two types of objective functions that are referred to as "multiplicative" or "additive." Multiplicative objective functions return the product of the variable valuations and additive objective functions return the sum of the variable valuations. Multiplicative and additive objective functions are observed to represent many different real-life objective functions, including but not limited to time, cost, profit, energy consumption, up-time, reliability, chance of success, and risk. For example, time is an additive type of objective since each component delay in a system is summed to compute the overall delay. Likewise, reliability is multiplicative since overall system reliability is product of the reliabilities of each serial component comprising the system structure.

C. Objective Function Table

The final input to the TVC system is a table relating the objective functions to the input variables. Each input variable requires r different values for each objective function for a given radix, r. An example of an objective function table for package delivery cost is given in Table I. In this example, we show the cost objective function for Trucks A and B, the staging site, and the refresh site for the simple case where $r = 3$. The choice of $r = 3$ is representative of the different available options for the package delivery task. Each system structure element is denoted as "State" in Table I and as an individual block in Fig. 2. Thus, TVC could be used to find the overall minimum cost in relation to which time of day each system element is employed. The use of $r = 3$ is used to denote objective function values for Trucks A and B representing one of three different time periods of day; the radix values for the staging site represent loading the truck on a light, normal, or rush schedule; and the radix values for the refresh site represent performing a light, partial, or full maintenance on the truck.

TABLE I
PACKAGE DELIVERY COST

State	0	1	2
Truck A (block A)	30	60	90
Truck B (block B)	25	15	30
Staging Site (block C)	100	150	125
Refresh Site (block D)	110	50	20

D. MDD Traversal

When all TVC input data have been specified, the optimization can be invoked that in turn causes specific MDD traversals to occur. A complete path from an initial to a terminal vertex represents a candidate solution. During the MDD traversal, the overall objective is computed for each path. In the case of a cost objective, an additive relationship is used to sum all of the individual costs resulting in an overall system cost. During the traversal, the calculation of the objective functions is performed using a simple table lookup based upon the current vertex variable label. Traversing every path from the initial to a terminal vertex and calculating the value of all of the objective functions for each traversal is thus an exhaustive search that yields all possible solutions.

The use of MDDs inherently provides significant performance advantages due to the reduction rules previously described. Although the MDD structure is a compact representation of the structure function and traversing the MDD is an efficient operation, exhaustively searching all possible solutions is not required in many cases. TVC thus utilizes certain heuristics to speed up the search. As the traversals are made, certain constraints on the optimization goals can be applied to prune the search and increase performance of TVC. For example, if a partial path traversal indicates that a cost threshold has been exceeded, there is no need to continue the traversal along that particular path. Utilizing these optimizations, we can efficiently create and traverse the MDD with dynamic path pruning. We start the traversal at the initial vertex of the MDD, recursively traversing it in a depth-first fashion while keeping track of the cumulative value associated with each objective function on the given path. Certain portions of the solution space and hence certain paths are dynamically pruned from the search during the traversal when thresholds are exceeded. When a terminal vertex is reached, the results are stored in a table that relates the state/variable path chosen and the objective function values for that path representative of a candidate solution.

When the dynamically pruned recursive traversal completes, TVC processing is complete with regard to the currently selected objective function and it will output a set of non-dominated solutions, meaning that no solution generated is strictly worse than any other solution when measured by every objective function. That is to say, non-dominated solutions are the Pareto-optimal front (or approximate the Pareto-optimal front if heuristics are used to speed up searching). The resulting solutions are then presented to the user, in the style of other *a posteriori* multiple objective optimization algorithms. At this point the user can determine which solution is most appropriate for their goals by using domain expertise.

IV. Example Application

To demonstrate the capabilities of TVC we consider an example in the business domain. In this example, a hypothetical e-commerce company is launching a new website. The company has a number of choices to make, including the quality of the web designer they are hiring, the number of customer service agents and site maintenance technicians they need, and the type of web servers to use. Overall, the company would like to minimize the cost of this new website, minimize the amount of time the website takes to build, but maximize the uptime of the website.

A. Example: Block Diagram

First, a block diagram is created that represents the decisions the company has to make. In this case, the company has a number of trade-offs to decide on. By paying a third party vendor to host the web servers, the company avoids the need to hire its own web technicians. If the company chooses to build out its web servers internally, it can choose between a high-end web server and a low-end web server. Figure 4 shows the block diagram for this example scenario.

B. Example: Objective Functions

The objective functions the company would like to consider are cost, time, and uptime. The cost objective function here refers to the monetary cost of creating and maintaining the website. The company would like to minimize this cost overall. Cost is an additive objective function. The time objective function here refers to the amount of time it takes to create the website. The company would like to minimize this time. Time is also an additive objective function. Lastly, the uptime objective function here refers to the percentage of time that the website is fully operational and taking customers' orders in a given year. For example, a 90% uptime means that the website was non-operational and orders were not processed for about 37 days in a given year. The company would like to maximize the uptime. Uptime is a multiplicative objective function; it can be thought of as the reliability of the website, or the inverse of the risk that the website crashes.

C. Example: Objective Function Table

With the block diagram constructed and the objective functions defined, the remaining step is to relate the input variables to objective function values. Each input variable will have exactly r different values for each objective function. In this example, the value for each variable indicates the following: a higher value for web developer means hiring a more skilled web developer; a higher value for customer support or web technicians means hiring more customer support agents or web technicians; and a higher value for any of the web servers means the web server is more powerful and capable of processing more orders. Table II shows the cost of each variable, Table III shows the time for each variable, and Table IV shows the uptime for each variable.

TABLE II
Cost

State	0	1	2
Web Developer	50	75	100
Web Technicians	20	40	60
Customer Service	25	50	75
Low-End Server	5	10	15
High-End Server	25	50	75
3rd-Party Server	50	100	150

TABLE III
Building Time

State	0	1	2
Web Developer	5	10	20
Web Technicians	1	2	3
Customer Service	2	4	6
Low-End Server	4	6	7
High-End Server	8	10	11
3rd-Party Server	8	10	11

TABLE IV
Uptime

State	0	1	2
Web Developer	0.93	0.95	0.99
Web Technicians	0.96	0.97	0.98
Customer Service	0.94	0.96	0.98
Low-End Server	0.93	0.94	0.95
High-End Server	0.96	0.96	0.96
3rd-Party Server	0.99	0.99	0.99

D. Example: MDD Traversal

To create the MDD, the block diagram is first converted to a discrete radix-r function through the traversal of the block diagram in JSON format. In this case, let the the Web Developer be represented by A, Customer Service by B, 3rd-Party Web Server by C, Internal Web Server (High End) by D, Internal Web Server (Low End) by E, and Web Technicians by F. The block diagram would then be represented in infix as $A * B * (C + ((D + E) * F))$, which translates to postfix as $ABCDE + F * + * *$. From this postfix form, the MDD is created as previously described. We note that an exhaustive representation of the solution space would require 729 possible paths. After traversing the MDD, the algorithm generates 29 candidate solutions that are a heuristic approximation of the Pareto-front; these solutions optimally trade-off with one another. Figure 5 visualizes the trade-offs between cost, building time, and uptime for the 29 candidate solutions in a parallel line diagram where each line corresponds to one of the computed solution paths. From these visualizations, the user can now select the appropriate trade-off among all objectives through various mechanisms. Once a solution is selected, the user can map from the solution to the individual decisions at each node in the block diagram that generated the solution. Also notice that the space of possible solutions to select can