Multicriteria Reasoning Considering Reliability or Availability

Tarik Hadzic
Cork Constraint Computation Centre
Computer Science Deparmtent
University College Cork
Email: t.hadzic@4c.ucc.ie

Helmut Simonis
Cork Constraint Computation Centre
Computer Science Deparmtent
University College Cork
Email: h.simonis@4c.ucc.ie

***Abstract*—Reliability and/or availability are increasingly important aspects in the design of systems, especially networks and service offerings. Optimization here is a multi-criteria process finding the right compromise between cost and quality. At the same time, results should be explored online, either as part of interactive, user-centric design tools or for Web based service negotiation. In this paper we consider a generic method for multi-criteria optimization based on reliability/availability graphs, but also provide a more specialized solution for dealing with the widely used reliability/availability block diagrams. Initial experiments show that the techniques can be applied to realistic problem sizes from the literature.**

I. Introduction

Reliability and availability are increasingly important aspects in the design of systems, especially networks and service offerings. We recall that *reliability* is a probability measure which indicates if a (non-repairable) system will be able to successfully complete an operation, while *availability* denotes the probability that a (repairable) system will be ready to operate at a future time point. Although the details of estimating reliability or availability are somewhat different, we can handle each within the same formal framework. We will mainly talk about reliability in the following, it should be understood that the results equally apply to availability, unless stated otherwise.

With the increasing importance of networks and Web services, requirements for reliability become more stringent, and need special consideration in the design phase. Often we can choose between many alternative elementary components, each with a specific cost/performance compromise. We are then faced with a combinatorial optimization problem to find the best overall compromise between cost and quality. We may be looking for the cheapest implementation which satisfies minimal reliability requirements, or for the best reliability measure that can be achieved within a given cost budget. We may also want to know if a certain component choice is feasible with respect to the overall quality and cost restrictions. We often need a real-time answer to these queries either as part of a Web service negotiation, or as an interactive component in a user-centric design tool.

In this paper we therefore present an approach to real-time answering of multicriteria queries that involve reliability and cost functions. Our approach is based on generating an *efficient frontier* and compiling reliability into a *multi-terminal decision diagram* (MTMDD) in the off-line phase, so that the relevant cost/reliability queries can be quickly answered online. To the best of our knowledge this is the first approach to such a problem. We consider two dominant descriptive frameworks for expressing reliability, which differ in the complexity of the problem to be handled [1]. We consider reliability functions in the form of *probabilistic graphs* which are especially interesting as they can model many relevant systems, such as wireless and wired network designs. We show how to exploit specific aspects of their computational representation, in a form of a *binary decision diagram* (BDD), to enhance both the efficient frontier generation as well as the MTMDD compilation. We also consider reliability functions in the simpler, yet very widely used form of *reliability block diagrams* (RBDs). In this case we develop a completely novel approach based on recognizing that the RBD corresponds to a graphical representation with a *tree* topology, which we denote as the *syntax tree*. This allows for a structure-sensitive approach to generating the efficient frontier. Our experiments demonstrate that generated frontiers and MTMDDs are suitable for online use. In particular, our structure-sensitive approach to generating efficient frontiers for RBDs improves the performance by orders of magnitude over the general approach.

Related work. Optimal design of reliable systems is an important aspect of systems design and has received widespread attention [2]. Many techniques ranging from dynamic programming to Integer Programming and meta-heuristics have been used in solving the *Reliability redundancy allocation problem* [3], a single criteria optimization sub-class of the problems considered here. Recently, an incomplete multi-criteria optimization method using genetic algorithms has been described in [4]. It handles, like our method, additive costs and multi-linear reliability measures, but only considers a very limited subset of reliability functions considered here, consisting of a single series of alternative, parallel sub-systems, and provides no guarantee for solution quality.

Binary decision diagrams (BDDs) [5] are a state-of-the-art representation that have for a long time been used for reliability estimation of fault trees [6]. BDDs have also been

used for reliability computations over *reliability graphs* and some forms of sensitivity analysis have been suggested [7].

Graphical structures are the primary mechanisms for modeling and reasoning with uncertainty in the AI domain. However, these research efforts typically emphasize the expressive power of Bayesian networks as a modeling framework [8], but do not provide special attention to RBDs - probably because the single-objective probabilistic queries they consider are already efficient even for more complicated topologies. The *syntax-tree* structure we use for describing RBDs is close to an *arithmetic circuit* (AC), a graphical structure often used for evaluating probabilistic expressions [9]. However, ACs have a more complicated topology (directed acyclic graphs), have slightly more specific internal arithmetic operators (multiplication and addition), and have not been considered in the multi-criteria setting.

II. Preliminaries

Given n variables $X = \{x_1, \ldots, x_n\}$ defined over domains $D_1, \ldots, D_n$, a *multi-valued decision diagram* (MDD) defined over X is a directed acyclic graph, $M(V, E)$ with vertices V and edges E. There are two special nodes: a *root node* $\mathbf{r}$, and a terminal node $\mathbf{1}$. Every node $u \neq \mathbf{1}$ is labeled with a variable $var(u) \in \{1, \ldots, n\}$ and its outgoing edges e are labeled with values $val(e)$ from $D_{var(u)}$. An MDD is *ordered* if labeling var respects a fixed variable ordering along all the paths. V_i denotes the vertices labeled with the i^{th} variable. E_i denotes edges originating in V_i. Each edge in E_i corresponds to an assignment $x_i = val(e)$, and each path p from the root to the terminal, denoted as $p : \mathbf{r} \rightsquigarrow \mathbf{1}$, corresponds to a complete assignment in $D_1 \times \ldots\ldots \times D_n$. We will use $e : u \rightarrow u'$ to denote an edge from node u to a child node u'. In the rest of the paper therefore we will use the terms path and assignment interchangeably. An MDD M *represents* solution set $Sol = \{val(p) \mid p : \mathbf{r} \rightsquigarrow \mathbf{1}\}$. In this paper we work with *merged* MDDs, where all isomorphic nodes are merged. We use $p \times \{v\}$ to denote the extension of partial assignment p with the value v. A binary decision diagram (BDD) is an MDD defined over binary domains $D_i = \{0, 1\}$, $i = 1, \ldots, n$.

MDDs can be seen as representing logical functions $f : D_1 \times \ldots\ldots \times D_n \rightarrow \{0, 1\}$. For each assignment $(x_1, \ldots, x_n) = (v_1, \ldots, v_n)$ (denoted as $\overrightarrow{x} = \overrightarrow{v}$) $f(v_1, \ldots, v_n) = 1$ if and only if there is an MDD path $p : \mathbf{r} \rightsquigarrow \mathbf{1}$ that corresponds to assignment $(v_1, \ldots, v_n)$. For example, a logical function over Boolean variables $x_1, \ldots, x_5$, $(x_1 \wedge x_4) \vee (x_2 \wedge x_5) \vee (x_1 \wedge x_3 \wedge x_5) \vee (x_2 \wedge x_3 \wedge x_4)$ is represented by an MDD in Figure 1.

Reliability is a probability measure which indicates if a (non-repairable) system will be able to successfully complete an operation. It is a function that expresses a probability of success of the entire system in terms of reliabilities of individual components, depending on the nature of their connections. The most fundamental types of connections between the components are a *serial* and a *parallel* connection. The reliability of a serial connection of two components with reliabilities r_1, r_2 is $r_1 \cdot r_2$, while the parallel connection of the same components has a reliability $1-(1-r_1)(1-r_2)$. By application of these simple evaluation rules, one can efficiently evaluate the reliability of more complex structures such as *reliability block diagrams*. However, there are more complex types of connections, e.g. captured by *probabilistic graphs*, that cannot be concisely expressed using only these connection types.

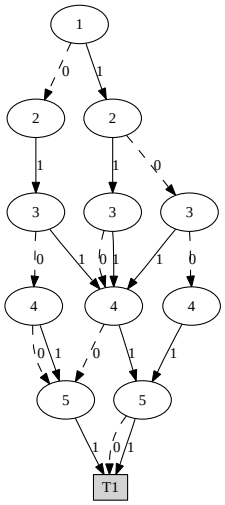


Fig. 1. An MDD encoding of a Boolean function $(x_1 \wedge x_4) \vee (x_2 \wedge x_5) \vee (x_1 \wedge x_3 \wedge x_5) \vee (x_2 \wedge x_3 \wedge x_4)$.

III. Multicriteria Queries

In this paper we consider scenarios where we are specifying a system consisting of n components, and each discrete choice for a component is associated with an atomic reliability and a cost. We are interested in designs that lead to systems of high reliability and low cost. Formally, we are given n variables $x_1, \ldots, x_n$ defined over domains $D_1, \ldots, D_n$. Assigning $x_i = v$ means that for the i-th component we select the option v. Each assignment $x_i = v$ is associated with reliability $r_i(v)$ and cost $c_i(v)$. Each complete assignment $x_1 = v_1, \ldots, x_n = v_n$ is associated with the reliability of the entire system $Rel(v_1, \ldots, v_n)$ and cost $C(v_1, \ldots, v_n)$. We consider answering the following multi-criteria queries:

- Q1 $min\ C(\overrightarrow{x})$, s.t. $Rel(\overrightarrow{x}) \geq R_{min}$.
- Q2 $max\ Rel(\overrightarrow{x})$, s.t. $C(\overrightarrow{x}) \leq C_{max}$.
- Q3 Given $x_i, v \in D_i$, is there a solution x containing $x_i = v$ s.t. $Rel(\overrightarrow{x}) \geq R_{min}$ and $C(\overrightarrow{x}) \leq C_{max}$.

Note that all the queries involve reasoning about all variables $\overrightarrow{x} = (x_1, \ldots, x_n)$ simultaneously. Query $Q1$ asks for the cost of the cheapest implementation of the system that satisfies the minimal reliability requirement R_{min}. Query $Q2$ asks for the value of the most reliable implementation of the system that satisfies the maximal cost restriction C_{max}. Finally, query $Q3$ asks if a specific implementation of an edge $x_i = v$ is possible so that both the maximal cost and minimal reliability restrictions are satisfied. These queries occur naturally in design of systems, such as networks or service offerings.

With the increasing importance of networks and Web services, requirements for reliability become more stringent, and need special consideration in the design phase.

We often need a real-time answer to these queries either as part of a Web service negotiation, or as an interactive component in a user-centric design tool. Since all of these problems are NP-hard in general, in this paper we consider the *knowledge-compilation approach*: invest in solving the problem in the off-line phase so that online queries can be answered efficiently.

In reminder of the paper we assume that the cost function is *additive*, i.e. of the form $C(x_1, \ldots, x_n) = \sum_{i=1}^{n} c_i(x_i)$, where c_i are atomic functions. Additive cost functions are one of the most important modeling formalisms that naturally express a number of important real-world properties, where the cost of entire object depends on the costs of individual parts. In our case, the cost of implementing the entire system is the sum of costs for implementing each component. The class of additive cost functions subsumes a very important class of linear cost functions, that are used in entire research areas, such as integer linear programming.

However, there is no obvious framework for expressing reliability functions - a number of formalisms has been investigated with increasing expressiveness but also more complex evaluation. In reminder of the paper we will discuss our solution approaches wrt. two very common formalisms: *probabilistic graphs* and *reliability block diagrams*.

IV. Answering Q1 and Q2 through Efficient Frontier Computation

To answer queries $Q1$ and $Q2$ efficiently, it suffices to generate the *efficient frontier* in the off-line phase, which is the set of all non-dominated pairs of reliability and cost that correspond to valid solutions. We say that pair (r_1, c_1) is dominated by (r_2, c_2), iff $r_1 \leq r_2 \wedge c_1 \geq c_2$ and at least one of the coordinates is different, i.e. $(r_1, c_1) \neq (r_2, c_2)$. We assume that efficient solutions with reliability less than R^*_{min} or cost larger than C^*_{max} would never be considered, and that real-time queries Q_1, Q_2, Q_3 always refer to some $R_{min} \geq R^*_{min}$ and $C_{max} \leq C^*_{max}$.

A. General Reliability Function

For a general reliability function, Algorithm 1 generates the efficient frontier through a straightforward depth first traversal of all assignments. $p \equiv 0$ denotes an *infeasibility test*, i.e. if all extensions of a partial assignment p lead to solutions outside the R^*_{min}, C^*_{max} bounds. For each complete feasible assignment p the algorithm inserts reliability-cost pair $(Rel(p), C(p))$ into a globally accessible list L and prunes dominated tuples. Note that by labeling each non-dominated tuple with a corresponding partial assignment p, we are able to recover a solution for each element in the efficient frontier. Note also that the number of paths explored is at least linear in the size of the efficient frontier (which can already be exponentially large) and that possibly many more paths might be explored if we do not recognize that some partial assignments p would inevitably lead to solutions outside the R^*_{min}, C^*_{max} bounds.

Algorithm 1: `EF(path p, var i)`: Compute a part of efficient frontier within bounds R^*_{min}, C^*_{max} for reliability $Rel(x)$ and cost $C(x)$.

```
Data: Globally available sorted list L, functions Rel(x), C(x),
      bounds R*_min, C*_max
if p ≡ 0 then
    return;
if i = n then
    if Rel(p) ≥ R*_min ∧ C(p) ≤ C*_max then
        if (Rel(p), C(p)) is nondominated in L then
            L ← L ∪ (Rel(p), C(p));
            remove pairs dominated by (Rel(p), C(p)) from
            L;
    return;
else
    foreach v ∈ D_i do
        EF(p × {v}, i + 1);
return;
```

The time complexity of Algorithm 1 depends on the number of paths explored p, the cost of infeasibility test $p \equiv 0$, and the cost of atomic evaluation tests $Rel(p)$, $C(p)$. In the worst case, the number of paths explored is exponential, so even for efficient evaluation tests $Rel(p)$, $C(p)$, the overall scheme has exponential time complexity. Note that the general scheme can be improved by exploiting the *monotonicity* of the reliability function: more reliable choices for any component can not reduce the overall system reliability. This provides an efficient infeasibility test $p \equiv 0$ for reliability. Given a partial assignment p to $x_1 = v_1, \ldots, x_i = v_i$, the most reliable extension of p is $x_{i+1} = v^{max}_{i+1} \ldots, x_n = v^{max}_n$ such that each v^{max}_j is the most reliable value in the j-th domain, $j = i+1, \ldots, n$. We denote such an extension as p_{max}. Hence, the test $p \equiv 0$ can be implemented as $Rel(p_{max}) < R^*_{min}$, which takes time proportional to evaluating Rel given an assignment.

Once the efficient frontier is generated (e.g. in an off-line preprocessing step), an answer to Q1 and Q2 queries can be easily retrieved online through a simple traversal of the frontier, which is represented in the sorted list L. The complexity of traversing the list depends on the list implementation, but even for simplest implementations is in worst-case linear. Since Algorithm 1 is executed offline, the time to generate efficient frontier is not as critical as the size of list L - which determines the complexity of online traversal. Since for each reliability or cost there is at most one pair in the efficient frontier having such value, the size of L is less than the number of different costs and different reliabilities.

B. Reliability Function as a Probabilistic Graph

A critical issue for our approach in Algorithm 1 is the complexity of evaluating the reliability expression $Rel(p)$ over a complete assignment p (representing a fully specified

system). It is surprising, but a well known fact, that for many formalisms that allow concisely specifying reliability expressions, such evaluation is an NP-hard problem. One of the most general such formalisms are *probabilistic graphs* [1], which are essentially network topologies that capture the dependencies of its components. In this section we will assume that the reliability function $Rel(x_1, \ldots, x_n)$ is modeled by a probabilistic graph. An example of such network topology is given in Figure 2, for a system with five variables. Each edge is associated with a probability of a success. The probability of a system associated with the network typically denotes the probability of existence of a path between two designated points, the source and the terminal node (referred to as the 2-terminal reliability problem) or the probability that all the nodes are connected (all-terminal reliability problem).

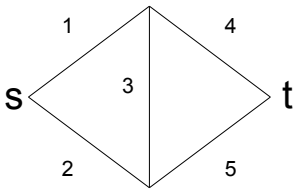


Fig. 2. A reliability network topology example over a bridge system with five edges. Atomic reliabilities for each edge are: $r_1 = 0.8$, $r_2 = 0.8$, $r_3 = 0.6$, $r_4 = 0.6$, $r_5 = 0.8$.

A prominent multicriteria problem over network topologies is the *network design problem*, where each edge can be either included in the design or not. The goal is to find a cheapest subset of edges such that the overall reliability is above some R_{min} threshold. Note that this corresponds to a problem with binary domains, where $x_i = 1$ means that the edge e_i is included. Each included edge e_i has reliability r_i.

C. BDD Representation of Probabilistic Graphs

Just evaluating the reliability of a fixed network topology where the reliability of each edge (a system component) is fixed ($x_i = r_i$), i.e. computing a value of the reliability function $Rel(r_1, \ldots, r_n)$, is known to be an NP-hard problem. Therefore, state of the art approaches encode the reliability function Rel more explicitly, in the form of a *binary decision diagram* (BDD) [5], [7], [10]. Each path in the BDD represents a subset of edges so that the corresponding network is connected (wrt. designated source/sink, or wrt. all nodes).

An example of the BDD encoding two-terminal connectedness of the network topology from Figure 2 is shown in Figure 3. The nodes labeled with `s` and `t` are the source and terminal node respectively. Once the BDD is computed (it can be exponentially large in the size of the input network), evaluating the reliability is efficient in the BDD size using a simple recursive relationship. Given atomic reliabilities r_i for each component, each edge e in the i-th BDD layer is labeled with corresponding probability r_e which is r_i if $val(e) = 1$ or $1 - r_i$ if $val(e) = 0$. Reliability of the terminal is set to $Rel(\mathbf{1}) = 1$ and a simple recursive traversal is used to compute the reliabilities of the remaining nodes: $Rel(u) = \sum_{e:u \to u'} r_e \cdot Rel(u')$. Finally, the reliability of the entire system is the reliability of the root node $Rel(\mathbf{r})$.

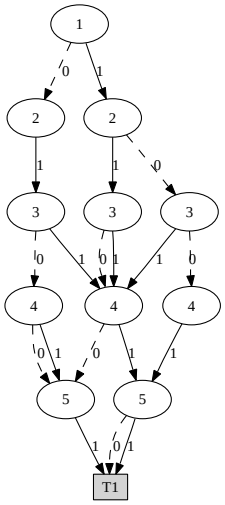


Fig. 3. A reliability BDD for a network topology from Figure 2. Note that this is the same as a BDD from Figure 1 representing Boolean function $(x_1 \wedge x_4) \vee (x_2 \wedge x_5) \vee (x_1 \wedge x_3 \wedge x_5) \vee (x_2 \wedge x_3 \wedge x_4)$. In fact, this Boolean function represents the 2-terminal connectivity between the source and the terminal.

In the *network design problem* however, each network topology edge e_i does not have a fixed reliability r_i since the implementing component has not yet been chosen. In fact, for each edge e_i we have a number of implementation options, each leading to different reliability. Therefore, the reliability of the edge e_i is no longer a constant r_i, but a function which we denote as $r_i(x_i)$ depending on the choice for component x_i. One of the choices is not to have the edge at all, $x_i = 0$, which enforces $r_i = 0$ reliability in the computations. Each assignment to $x_1, \ldots, x_n$ corresponds to a fixed implementation of the network topology, and its reliability can be evaluated as described above. We use $r_i(1) = r_i$ and $1 - r_i(1) = 1 - r_i$ to label corresponding edges.

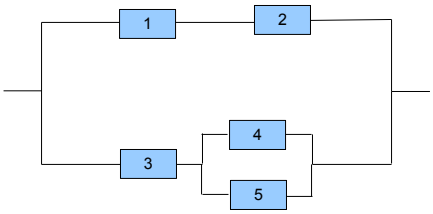
We will now show how to exploit the BDD representation of the reliability function to more efficiently implement the infeasibility test $p \equiv 0$ in Algorithm 1. We precompute *maximal downward reliabilities* $\texttt{DW}_{max}[u]$ of each BDD node u. $\texttt{DW}_{max}[u]$ is equal to the value returned by recursive $Rel(u)$ computation described above when all edges are included, i.e. when $x_1 = 1, \ldots, x_n = 1$. Consider the execution point of Algorithm 1 assuming a partial assignment p to variables $x_1, \ldots, x_{i-1}$. Let us denote the current upward reliabilities wrt. p as $\texttt{UP}[u]$, $u \in V_i$. These values can be maintained during execution of Algorithm 1 by first setting $\texttt{UP}[u'] = 0$ for all nodes in the current layer $u' \in V_i$, and then iterating through all nodes in the previous V_{i-1} layer and for each outgoing edge e, with reliability r_e updating its children nodes $\texttt{UP}[u'] = \texttt{UP}[u'] + r_e \cdot \texttt{UP}[u]$. Once this update is

made, the maximal reliability wrt. the current assignment is $Rel(p_{max}) = \sum_{u \in V_i} \texttt{UP}[u] \cdot \texttt{DW}_{max}[u]$.

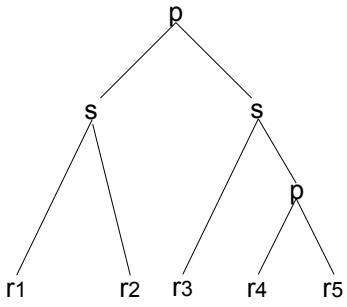
D. Reliability Function as a Block Diagram

In the previous section we have shown how to generate the *efficient frontier* for general reliability and cost function. We then showed how to exploit the *monotonicity* of the reliability function and how to exploit the *BDD representation* of reliability when it is modeled as a *probabilistic graph*. In this section we ask whether we can do better if the probabilistic function has a more specific form. In particular, can we improve the generation of the efficient frontier if the reliability is given as a *reliability block diagram* (RBD)?

RBDs are a one of the most widely used formalisms for encoding reliability functions. An RBD represents a nested combination of parallel and serial connections between components. An example of an RBD is given in Figure 4(a). They are a strictly less expressive modeling formalism than probabilistic graphs, but some tasks such as evaluating a reliability of an assignment $Rel(p)$ become linear in its size. The reliability of a serial connection of two components with reliabilities r_1, r_2 is $r_1 \cdot r_2$, while the parallel connection of the same components has a reliability $1 - (1 - r_1)(1 - r_2)$. By application of these simple evaluation rules, one can efficiently evaluate the reliability of the entire RBD. For example, if $r_1, \ldots, r_5$ are the reliabilities of corresponding components in the reliability block diagram from Figure 4(a), the corresponding reliability function $Rel(r_1, \ldots, r_5)$ is $1 - (1 - r_1 r_2) \cdot (1 - r_3(1 - (1 - r_4)(1 - r_5)))$.



(a) Block Diagram



(b) Syntax Tree

Fig. 4. A reliability block diagram (RBD) is shown in Figure 4(a) and the corresponding *syntax tree* in Figure 4(b).

How can we exploit the inherent decomposition present in the RBD beyond just using it as a black-box computation mechanism in Algorithm 1? A critical step in this direction is to recognize that an algorithm for evaluating the RBD expression can be explicitly represented as a graphical structure in the form of a tree. We refer to this graphical structure as a *syntax tree* (ST). An ST corresponding to the RBD from Figure 4(a) is shown in Figure 4(b). Every internal node of the tree corresponds to either a parallel or a serial connection (designated by p and s labels). We label every parallel and serial node with arithmetic operators, $\otimes_p, \otimes_s$, defined by $r_i \otimes_p r_j =_{def} 1 - (1 - r_i) \cdot (1 - r_j)$, and $r_i \otimes_s r_j =_{def} r_i \cdot r_j$. Every leaf node is labeled with the reliability r_i of the corresponding component. The reliability of a node $Rel(u)$, with children u_l and u_r is $Rel(u_l) \otimes_u Rel(u_r)$, where $\otimes_u$ is the arithmetic operator associated with u. Reliability of the entire system is the reliability of the root node.

E. Generating Efficient Frontier Over Syntax Tree

Note that the syntax tree does not have to be binary, since in reliability block diagram there could be three or more components in a parallel or a serial connection. However, since the parallel and the serial operator $\otimes_p, \otimes_s$ are *associative* and *commutative* we can always transform the reliability function into a *binary* tree e.g. by repeatedly replacing expressions $r_1 \otimes r_2 \otimes r_3$ with $r_1 \otimes (r_2 \otimes r_3)$. Consequently, without loss of generality we can assume that each node u is associated with two children nodes, left child $l(u)$ and right child $r(u)$. Also note that the operators $\otimes_p, \otimes_s$ are *monotonic* over the interval $[0, 1]$, i.e. if $p_1 \geq p_2$ from $[0, 1]$ then $p_1 \otimes p \geq p_2 \otimes p$. This property allows for efficient bound estimation, since for each sub-expression it suffices to take the maximal value.

Algorithm 2 generates the efficient frontier of a reliability function Rel and cost function C through a recursive scheme that follows the topology of the syntax tree. At each node u, the algorithm generates efficient lists associated with the left and right children, $l(u)$ and $r(u)$, respectively. Then the algorithm *combines* the lists incrementally using the operators associated with node u. Combination of a tuple (r_1, c_1) with (r_2, c_2) at node u, is $(r_1 \otimes_u r_2, c_1 + c_2)$, where $\otimes_u$ is either a parallel or a serial operator. The set combination $L \otimes_u (r, c)$ results in the set $\{(r', c') \otimes_u (r, c) \mid (r', c') \in L\}$. $L[i]$ denotes the i-th element of the list. The lists L, L_l, L_r are kept sorted, e.g. in decreasing order of reliability component, so that the merging of lists with elimination of dominated pairs (in function $MergeLists$) can be done more efficiently (in linear time), using an adaptation of advanced techniques for Knapsack optimization [11, Section 3.4].

Note that while the statement of Algorithm 2 is specialized for the RBD-reliability and an additive cost function, it can be applied to a general class of functions. We also note that for any two functions whose syntax tree topologies are *compatible* (one can be mapped to another by exploiting commutativity), and whose internal operators are *monotonic* (thus allowing efficient optimization) the above algorithm can be immediately applied.

Algorithm 2: `EF(node u)`: Compute efficient frontier from the syntax tree rooted at u.

$L \leftarrow \emptyset$;
if *u is a leaf labeled with x_i* **then**
  foreach $v \in D_i$ **do**
    $L \leftarrow L \cup (r_i(v), c_i(v))$;
  return L;
//compute efficient sets for children nodes
$L_l \leftarrow EF(l(u))$;
$L_r \leftarrow EF(r(u))$;
//combine lists
$L \leftarrow L_l \otimes_u L_r[0]$;
foreach $i = 1, \ldots, |L_r| - 1$ **do**
  $L' = L_l \otimes_u L_r[i]$;
  $L \leftarrow MergeLists(L, L')$;
return L;

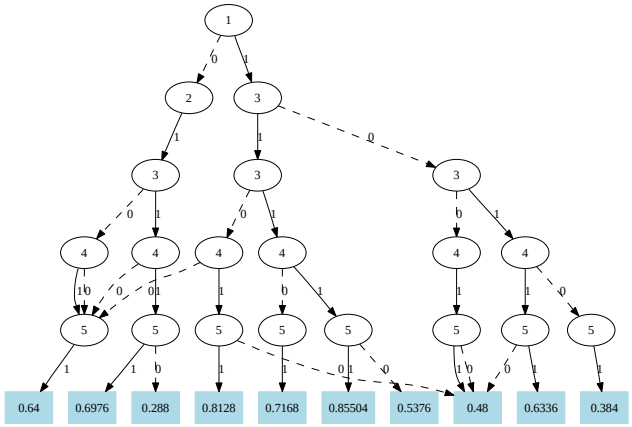


Fig. 5. Multi Terminal MDD for the network topology from Figure 2

V. Answering Q3 through MTMDD Compilation

Query $Q3$ is inherently more complex than queries $Q1$ and $Q2$, as it requires reasoning about a much larger set of solutions than the ones corresponding to efficient frontier. An assignment $x_i = v$ might be feasible wrt. restrictions R_{min} and C_{max} even if it does not participate in a single efficient solution. Hence, just generating an efficient frontier is not sufficient. Furthermore, the user should be able to enforce not just a single assignment but any number of assignments, and get a feedback in real-time.

An immediate approach to repeatedly answer query $Q3$ efficiently in the online setting wrt. different partial assignments is to pre-compute a structure that links every solution explicitly to its reliability and cost. Since there could be an exponential number of feasible solutions, an explicit (tabular) representation will incur a high space complexity. We therefore suggest exploiting MDD-like graphical representation of solution set that saves the space by merging isomorphic nodes. Since the cost function is additive, it would suffice to explicitly link solutions only to their reliability, and reason about the cost dynamically in the online phase, in an analogous procedure as used in [12].

In this paper, we suggest compiling the reliability function Rel into a *multi-terminal MDD* (MTMDD). An MTMDD represents a multi-valued function $f : D_1 \times \ldots \times D_n \rightarrow \mathbb{R}$. It can be understood as an MDD that has multiple terminal nodes $T_1, T_2, \ldots$ one for each value of a function it represents. An MTMDD corresponding to the network topology from Figure 2 with atomic reliabilities $r(x_1 = 1) = 0.8$, $r(x_2 = 1) = 0.8$, $r(x_3 = 1) = 0.6$, $r(x_4 = 1) = 0.6$, $r(x_5 = 1) = 0.8$ is shown in Figure 5.

There are 10 terminal nodes, corresponding to reliabilities 0.288, 0.384, 0.48, 0.5376, 0.6336, 0.64, 0.6976, 0.7168 , 0.8128 and 0.85504. Now, due to additivity of the cost function C, we can use domain pruning based on shortest-path algorithms, see e.g. [13]. To generate the MTMDD we use Algorithm 3, which is an adaptation of a generic compilation scheme presented in [14]. Once the children nodes are recursively created $(u_1, \ldots, u_{|D_i|})$ the algorithm creates the parent node through `get-vertex`$(u_1, \ldots, u_{|D_i|})$. The function `get-vertex` creates a new node only if an isomorphic node (with the same children nodes) does not already exist.

It is well known that MTMDDs grow exponentially with the number of terminals. However, we can control the number of terminal nodes by representing the reliability function *approximately*. We do not have to distinguish exactly between the reliabilities that are sufficiently close. For example, we can associate terminal nodes T_i with *reliability intervals* $[l_i, u_i]$ so that the reliability of each path ending in T_i is $Rel(p) \in [l_i, u_i]$. To compile such MTMDD we need to introduce only a minor modification in line 1 of Algorithm 3 - instead of returning a terminal corresponding only to the value $Rel(p)$, find and return a terminal corresponding to the interval $[l, u]$ that includes $Rel(p)$. Depending on the application domain, we can choose reliability intervals in different ways. In a *linear distribution* with coefficient d, we divide $[0, 1]$ into d intervals of width $1/d$: $0, 1/\mathrm{d}, 2/\mathrm{d}, \ldots, \mathrm{d} - 1/\mathrm{d}, 1$. In high-availability scenarios, e.g. for IP-routed, wired networks, it is only important to distinguish the so called "number of nines" of the reliability, $0.9, 0.99, 0.999, \ldots$. We refer to this as an *exponential distribution*, and for a fixed d we construct the intervals as $0, 1 - 1/10, 1 - 1/10^2, \ldots, 1 - 1/10^{\mathrm{d}-1}, 1$.

Algorithm 3: `MTMDD`(path p, int i): Compiles reliability function Rel into an MTMDD. The initial call is `MDD`$(\emptyset, 1)$

if $Rel(p_{max}) < R^*_{min}$ **then**
  return 0;
if $i = n$ **then**
1 **return** *Terminal*$(Rel(p))$
else
  foreach $j = 1 \ldots |D_i|$ **do**
    $u_j \leftarrow$ `MTMDD`$(p \times \{j\}, i + 1)$;
  result = `get-vertex`$(i, u_1, \ldots, u_{|D_i|})$;
return *result* ;

VI. EXPERIMENTS

We perform experiments to explore two main hypotheses: 1) the size of the MTMDD can be contained for realistic reliability settings and 2) when the probabilistic model can be expressed as reliability block diagrams, our specialized multi-objective optimization based on syntax trees performs better than our generic approach. The experiments ran as a single thread on a Dual Quad Core Xeon CPU, 2.66GHz with 12MB of L2 cache per processor and 16GB of RAM overall, running Linux 2.6.25 x64. All the reported times are in seconds.

A. MTMDD Size

In the first set of experiments we use a network topology from the literature [7]. We choose the largest network topology from the paper, `Relex9`, and evaluate the performance of our algorithms. It is an instance of a 2-reliability problem defined over 20 nodes. TThe corresponding merged MDD (BDD) representing the reliability expression has 2414 nodes and 4730 edges. The topology of the network is shown in Figure 6.

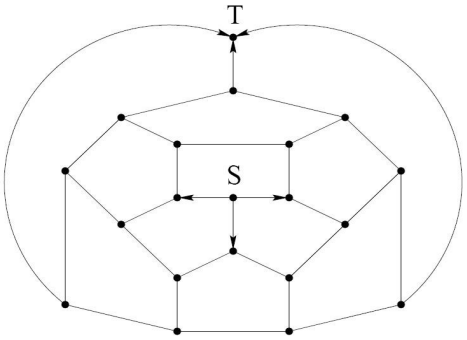


Fig. 6. A network topology of instance Relex 9.

We then compute MTMDDs for the *network design problem* where for each network edge e_i a user can choose whether it is included in the design ($x_i = 1$) with an implementation having reliability r_i or it not not included ($x_i = 0$) enforcing reliability $r_i = 0$. We accept only designs leading to overall reliability greater than $R^*_{min} = 0.9$. Edges that are present have a random reliability r_i which is drawn uniformly from an interval $[r_{min}, 1]$. Constant r_{min} is chosen depending on the application scenario and is either 0.8 or chosen from $\{0.95, 0.98, 0.99\}$. We consider two scenarios: 1) ad-hoc wireless networks where edges included in the topology have minimal reliability of $r_{min} = 0.8$, and where we use a *linear distribution* of reliability intervals $0, 1/\mathrm{d}, 2/\mathrm{d}, \ldots, \mathrm{d}-1/\mathrm{d}, 1$; 2) high-availability scenario, e.g. for IP-routed, wired networks, where each link must have a high reliability, $r_{min} \in \{0.95, 0.98, 0.99\}$ and we use an *exponential distribution* of intervals corresponding to the so called "number of nines" classification $0, 1-1/10, 1-1/10^2, \ldots, 1-1/10^{\mathrm{d}-1}, 1$.

The results are shown in Table I. First note that in both the average and the worst-case, the MTMDDs are of a size that is suitable for real-time shortest path explorations (in our experience the shortest-path based pruning takes a few milliseconds over MDDs with hundreds of thousands of nodes). The time required to compile the MTMDD is less important as it can be done in the off-line phase, e.g. as soon as the topology of the problem is known. While the size of MTMDDs clearly depends on the number of terminals, the compilation time does not, since it depends on the number of feasible paths explored, i.e. the value of the pruning threshold R^*_{min}. As an illustration for $R^*_{min} = 0.001$ the average compilation time is around 6 min. Note that the compilation times are longer in the second scenario. This is since the atomic reliabilities are higher, and hence a larger portion of paths are feasible.

Note that the cost is not reflected in experiments for MTMDDs since MTMDD size is the critical complexity factor: due to the additive nature of cost functions the complexity of online queries is linear in the MTMDD size.

B. Experiments with Reliability Block Diagrams

In the second set of experiments we randomly generate a large number of reliability block diagrams and compare performance of the generic efficient frontier generation (Algorithm 1) against the syntax tree approach (Algorithm 2). The results are shown in Table II. For each number of variables (column **Vars**) we generate 1000 random RBDs and cost functions and report average and maximum running times for the time needed to generate the efficient frontier with the syntax-tree approach (t_{rbd}) without minimal reliability threshold R_{min}, with the MDD approach for $R_{min} = 0.001$ (t_{mdd}) and with the MDD approach for $R_{min} = 0.9$ ($t^{0.9}_{mdd}$). Each RBD is generated as a random binary syntax tree. For a given number of variables n we first create a list of n leaf nodes $u_1, \ldots, u_n$, each associated with random reliabilities $r_1, \ldots, r_n$ drawn uniformly from an interval $[r_{min}, 1]$. We then randomly select two nodes from the list u, u' and construct for them a joint parent node u''. We then remove u, u' from the list and insert u'' instead. We repeat the process until only one node is left, which is the root node of the syntax tree. Afterwards, we traverse the tree and for each non-leaf node we select with 0.5 probability whether it is a serial or a parallel node. In this way we are generating random binary syntax trees (RBDs) in the uniform manner. The cost function is constructed by randomly drawing a cost of each network edge from an interval [0,100].

The results clearly demonstrate that the RBD-specific scheme outperforms the generic method by orders of magnitude, both for the average and the worst-case case. We also show that the RBD-specific scheme scales well with the number of variables, requiring for example on average only a few milliseconds for RBDs with 100 variables. This makes efficient frontier generation possible even in a purely online setting.

Scenario	d	Distr	r_{min}	Avg Term	Avg V	Avg E	Avg t	Max Term	Max V	Max E	Max t
Wireless	10^2	linear	0.80	9	37,422	72,206	25.6s	10	55,380	108,043	67.01s
	10^3	linear	0.80	95	78,716	147,271	25.7s	100	112,186	210,203	65.64s
	10^4	linear	0.80	941	108,335	198,688	25.6s	991	166,922	313,382	57.94s
Wired	7	expo	0.95	4	22,505	44,494	2m 53s	5	25,924	51,329	4m 48s
	7	expo	0.98	4	19,568	38,831	5m 31s	5	23,480	46,582	6m 16s
	7	expo	0.99	4	19,099	37,978	5m 37s	5	20,919	41,595	6m 14s

TABLE I
MTMDD COMPILATION FOR THE RELEX9 INSTANCE. THE RESULTS ARE AVERAGED OVER 100 TRIALS, WITH AVERAGE AND MAXIMAL VALUES ENCOUNTERED SHOWN IN THE TABLE. COLUMN **TERM** INDICATES THE NUMBER OF TERMINALS. COLUMN **T** INDICATES COMPUTATION TIME.

Note that the large performance difference holds even though we have used a carefully designed variable ordering for *evaluation MDDs* (MDDs encoding the reliability function). We group variables that appear closely related in RBD sub expressions and as a result, we get very thin evaluation MDDs, with only few nodes per layer. Also note that, to our surprise, the number of elements in the efficient frontier encountered is orders of magnitude smaller than the number of different reliabilities. For example, for RBDs with more than 20 variables, the efficient frontier contains on average less than 100 elements, but the total number of different reliabilities ranges in tens of thousands. In the experiments shown, we generated the coefficients such that higher reliabilities correspond to higher costs. However, experiments based on other coefficient correlations do not significantly change the results. In particular, the difference between the size of the efficient frontier and the number of reliabilities remains very large. We believe that this is happening due to very high reliability "resolution": due to complicated network topology there is a large number of almost identical reliabilities (differing by less than 0.001, 0.00001, 0.0000001, etc.). At the same time, there is a low cost "resolution", as the cost function is a simple sum of individual costs. Therefore, the same cost can appear with the a number of reliabilities.

Vars	Avg t_{rbd}	Avg t_{mdd}	Avg $t_{mdd}^{0.9}$	Max t_{rbd}	Max t_{mdd}	Max $t_{mdd}^{0.9}$
20	0.0003	0.037	0.008	0.01	0.54	0.51
23	0.0006	0.756	0.137	0.01	4.47	3.58
24	0.0007	1.295	0.22	0.01	8.99	7.57
25	0.0008	3.769	0.29	0.01	23.69	11.45
26	0.0010	5.902	0.36	0.01	31.59	31.23
27	0.0017	30.52	1.80	0.01	119.91	53.05
50	0.0048	-	-	0.02	-	-
75	0.0179	-	-	0.08	-	-
100	0.0594	-	-	0.57	-	-
250	1.3642	-	-	7.48	-	-

TABLE II
RESULTS OVER 1000 TRIALS WITH A TIME CUTOFF FOR EACH RUN OF 10 MINUTES. ALL TIMES ARE IN SECONDS. FOR INSTANCES WITH MORE THAN 50 VARIABLES NO SOLUTIONS WERE GENERATED IN THE MDD-BASED APPROACHES DUE TO TIME-OUT.

VII. CONCLUSIONS

We have introduced a set of techniques for multicriteria reasoning that involve reliability (availability) and cost functions. We have presented a generic scheme for answering a number of multicriteria queries, when the reliability is given as a general probabilistic graph. We have also shown how to further enhance our techniques if the reliability is modeled through less expressive but widely used reliability block diagrams (RBDs). Our experiments demonstrate that all our techniques are suitable for the relevant application scenarios. In particular, our methods exploiting RBDs improve the performance by orders of magnitude over the general approach.

REFERENCES

[1] M. Malhotra, "Power-Hierarchy of Dependability Model Types," *IEEE Transactions on Reliability*, vol. 43, no. 2, pp. 493–502, September 1994.

[2] W. Kuo, R. Prasad, F. A. Tillman, and C.-L. Mwang, *Optimal Reliability Design: Fundamentals and Applications*. Cambridge: Cambridge University Press, 2006.

[3] W. Kuo and R. Wan, "Recent advances in optimal reliability allocation," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 37, no. 2, pp. 143–156, 2007.

[4] H. A. Taboada, J. F. Espiritu, and D. W. Coit, "MOMS-GA: A multi-objective multi-state genetic algorithm for system reliability optimization design problems," *IEEE Transactions on Reliability*, vol. 57, no. 1, pp. 182–191, 2008.

[5] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. 35, pp. 677–691, 1986.

[6] A. Rauzy, "New algorithms for fault trees analysis," *Reliability Engineering and System Safety*, vol. 05, no. 59, pp. 203–211, 1993.

[7] X. Zang, H. Sun, and K. S. Trivedi, "A BDD-Based Algorithm for Reliability Graph Analysis," Department of Electrical Engineering, Duke University, Tech. Rep., 2000, available online: `http://citeseer.ist.psu.edu/213187.html`.

[8] J. G. Torres-Toledano and L. E. Sucar, "Bayesian networks for reliability analysis of complex systems," in *IBERAMIA '98: Proceedings of the 6th Ibero-American Conference on AI*. Springer-Verlag, 1998, pp. 195–206.

[9] A. Darwiche, "A Logical Approach to Factoring Belief Networks," in *KR2002: Principles of Knowledge Representation and Reasoning*, D. Fensel, F. Giunchiglia, D. McGuinness, and M.-A. Williams, Eds. San Francisco, California: Morgan Kaufmann, 2002, pp. 409–420.

[10] G. Hardy, C. Lucet, and N. Limnios, "K-Terminal Network Reliability Measures With Binary Decision Diagrams," *IEEE Transactions on Reliability*, vol. 56, no. 3, pp. 506–515, September 2007.

[11] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[12] H. R. Andersen, T. Hadzic, and D. Pisinger, "Interactive Cost Configuration Over Decision Diagrams," *Journal of Artificial Intelligence Research (JAIR)*, 2010, accepted for publication.

[13] T. Hadzic and H. R. Andersen, "A BDD-based Polytime Algorithm for Cost-Bounded Interactive Configuration," in *Proceedings of AAAI 2006*, 2006, pp. 62–67.

[14] T. Hadzic, J. N. Hooker, B. O'Sullivan, and P. Tiedemann, "Approximate compilation of constraints into multivalued decision diagrams," in *Proceedings of CP 2008*, P. Stuckey, Ed., vol. 5202. Springer-Verlag, 2008, pp. 448–462.