Multicriteria Reasoning Considering Reliability or Availability

Tarik Hadzic and Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

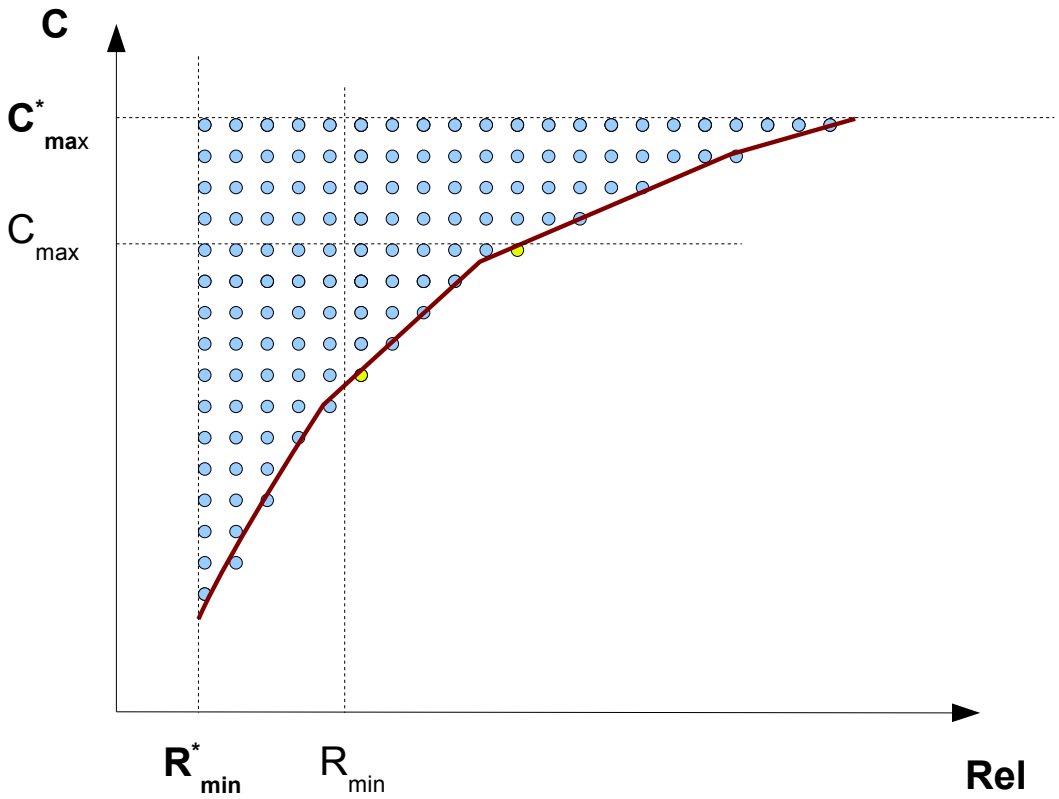
ICTAI 2010, Arras

Motivation

- Reliability and availability are important aspects of system design
- Many life-critical systems are required to operate without a system failure for a given period of time (nuclear, aerospace, spacecraft) - *reliability*
- Fraction of time the system is providing service to its users - *availability*
- We want to design systems as reliable (available) as possible. But, higher the reliability - higher the *cost*. This is a *multi-criteria optimization problem*.
- We often need real-time answers (e.g. in an interactive user-centric design tool)
- We suggest a generic method for *real-time interactive* multi-criteria optimization involving reliability or availability as one of the criteria

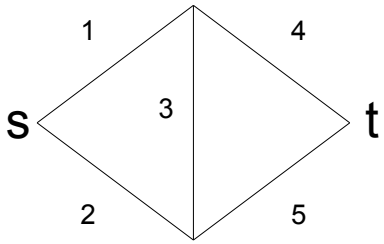
Overview of our Approach

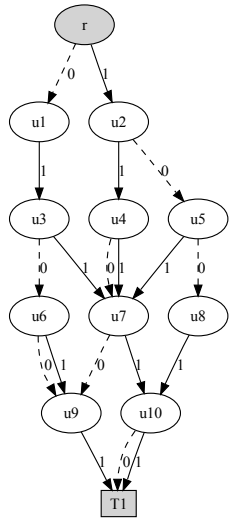
We generate *efficient frontier* in the offline phase to support efficient online interaction



Overview of our Approach

We handle reliability functions modeled as *probability graphs* by using *decision diagrams*

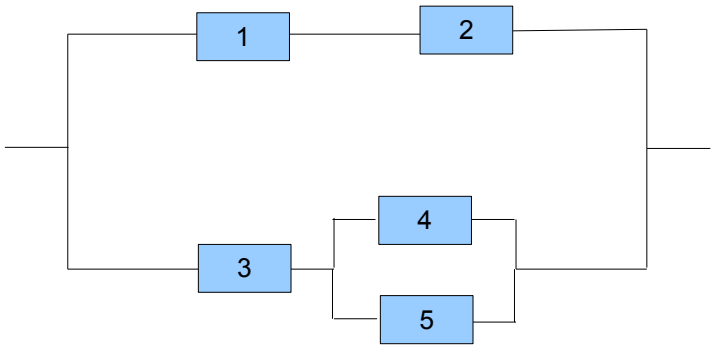


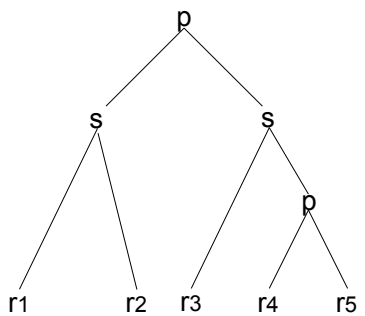


We introduce several algorithmic schemes that exploit specific aspects of decision diagrams to enhance efficient frontier generation

Overview of our Approach

We develop a specialized approach when reliability function is specified as a *block diagram*. We introduce the notion of *syntax trees* to exploit internal structure of block diagrams for faster frontier generation

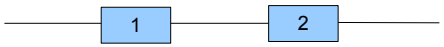




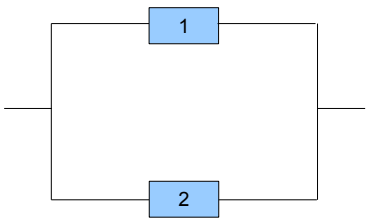
We demonstrate through experiments the value of our approach

Reliability and Availability

- Computing reliability and availability requires the same computational mechanism.
- Reliability of the i-th component expressed as a value $r_i \in [0, 1]$.
  - $r_i = 0$ - component not available (not functional)
  - $r_i = 1$ - component available (functional)
  - $r_i = 0.999$ - component available 99.9% of the time (functional with probability 99.9%)
- Two fundamental connection types for components r_1, r_2:
  - *serial*, $Rel(r_1, r_2) = r_1 \cdot r_2$

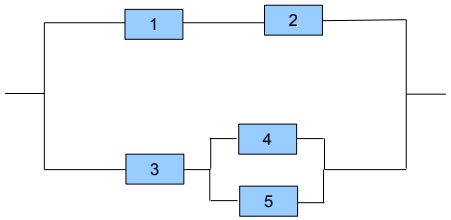


  - *parallel*, $Rel(r_1, r_2) = 1 - (1 - r_1) \cdot (1 - r_2)$

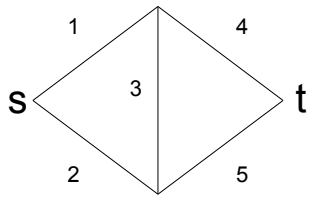


Block Diagrams and Probability Graphs

- *Block diagrams* consist only of parallel and serial connections and can be *efficiently* evaluated.



- *Probability graphs* are general graphs with edges labeled with probabilities. Reliability corresponds to a probability of existence of a path between a source and a terminal. Computing this probability is *NP-hard* (Valiant, 1979)[1]. State-of-the-art approach is based on *decision diagrams*.



[1] The complexity of enumeration and reliability problems, SIAM J. Comput.

Decision Diagrams

Definition (Multivalued Decision Diagrams)

A *decision diagram* is a rooted directed acyclic graph $G = (V, E)$ where every node u is labeled with a variable x_i and every edge e, originating from a node labeled x_i, is labeled with a value $a_i \in D_i$. The decision diagram contains a special *terminal* node **1**, that has no outgoing edges.

- Every path from root to terminal encodes an assignment
- MDDs are almost always assumed to be *ordered*
- MDDs achieve exponential space savings by *merging isomorphic nodes*
- MDDs can achieve additional savings by *removing redundant nodes*
- If all domains D_i are binary, i.e. $D_1 = \ldots = D_n = \{0, 1\}$, then we have a *binary decision diagram* (BDD)

Decision Diagrams

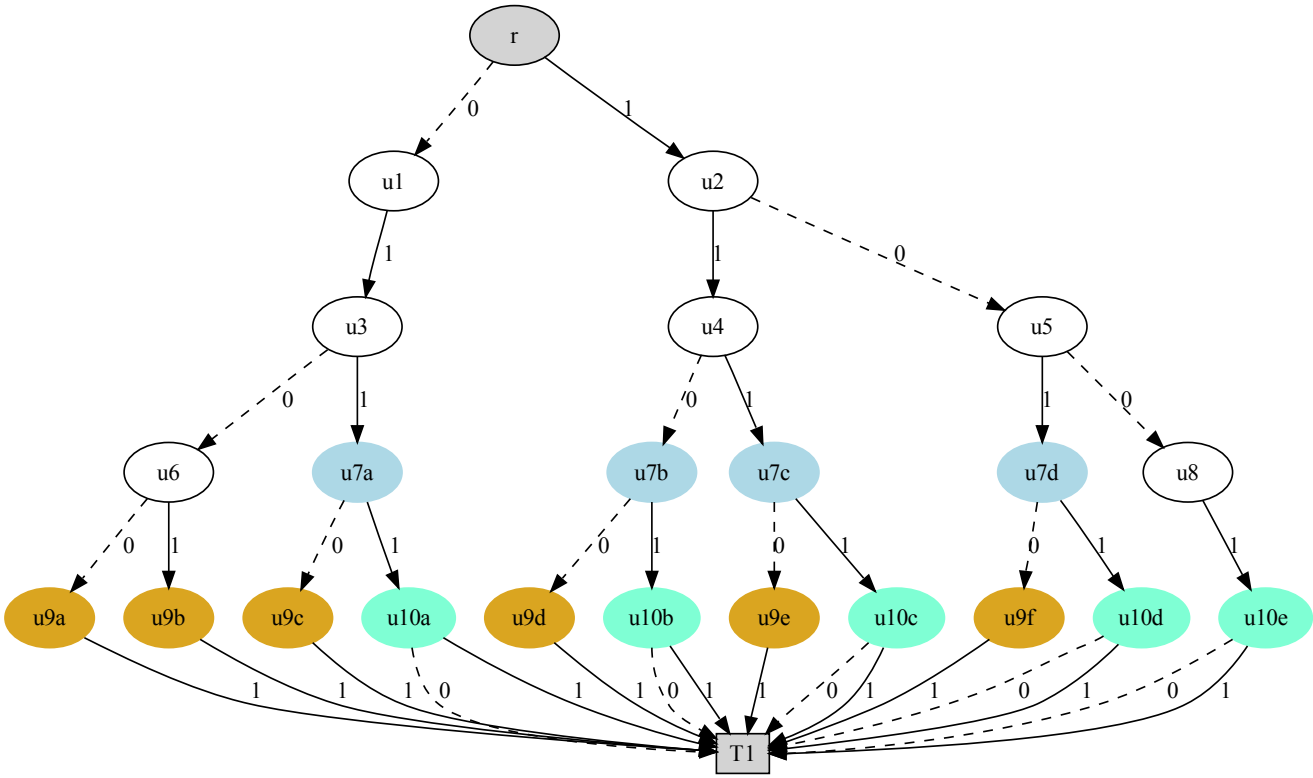


Figure: Expanded MDD for a Boolean function
$(x_1 \wedge x_4) \vee (x_2 \wedge x_5) \vee (x_1 \wedge x_3 \wedge x_5) \vee (x_2 \wedge x_3 \wedge x_4)$

Decision Diagrams

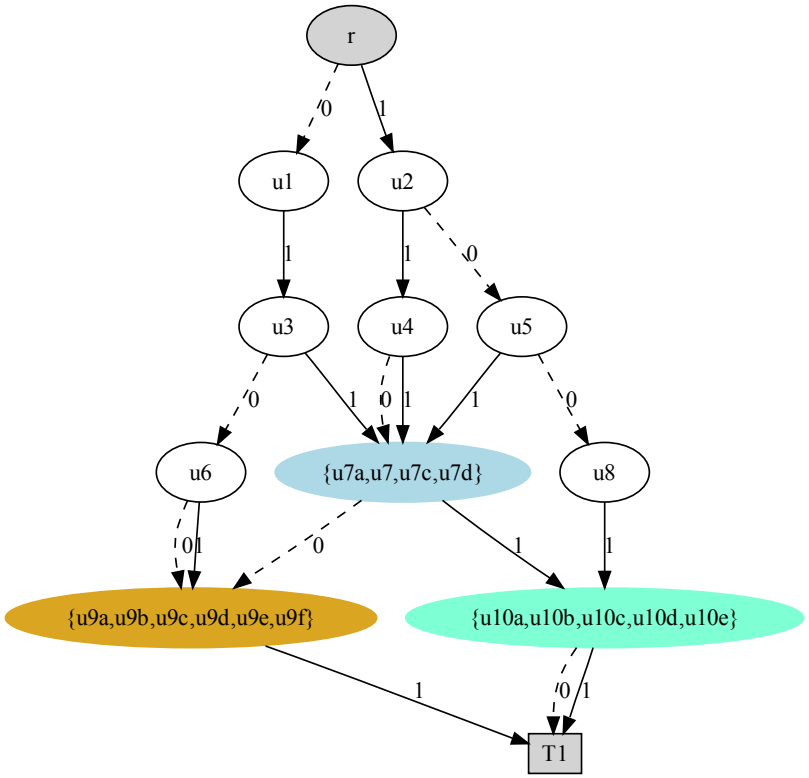


Figure: Merged MDD for a Boolean function
$(x_1 \wedge x_4) \vee (x_2 \wedge x_5) \vee (x_1 \wedge x_3 \wedge x_5) \vee (x_2 \wedge x_3 \wedge x_4)$

Decision Diagrams

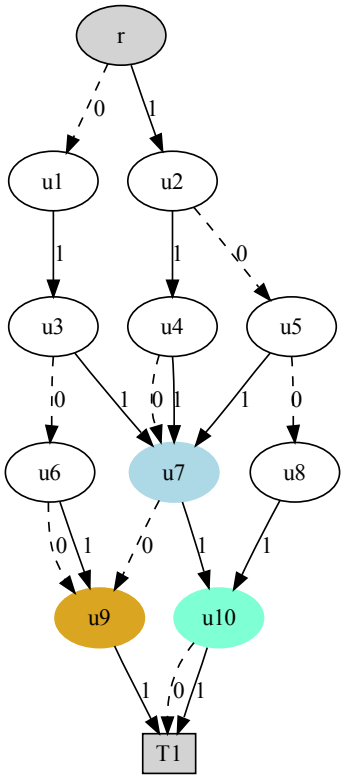


Figure: Merged MDD for a Boolean function
$(x_1 \wedge x_4) \vee (x_2 \wedge x_5) \vee (x_1 \wedge x_3 \wedge x_5) \vee (x_2 \wedge x_3 \wedge x_4)$

Decision Diagrams

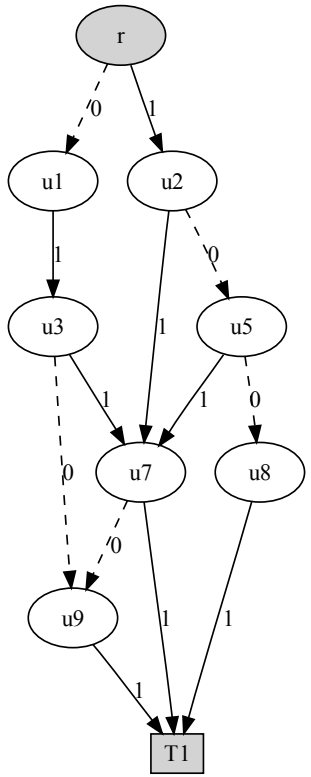
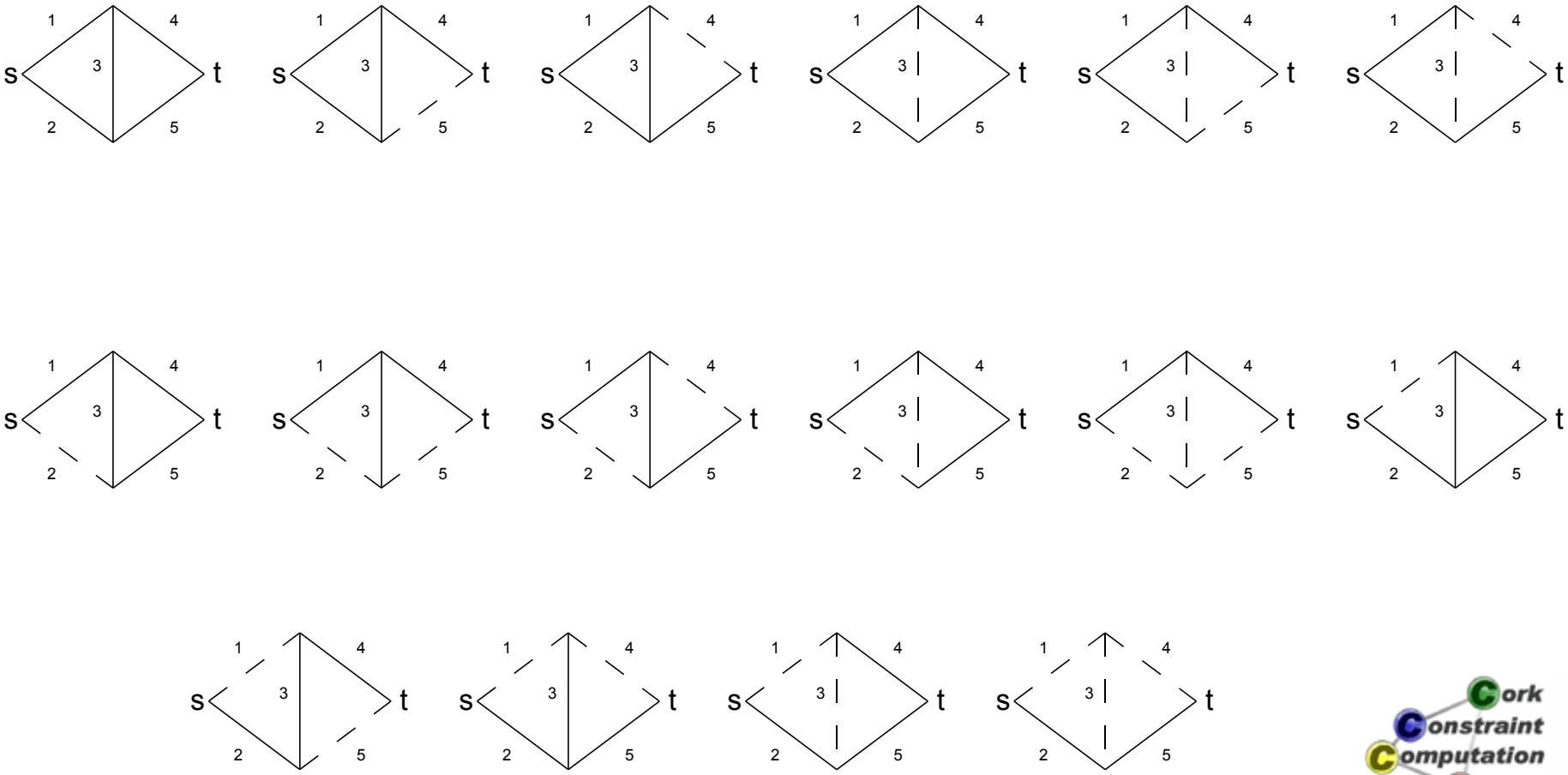


Figure: Reduced MDD for a Boolean function
$(x_1 \wedge x_4) \vee (x_2 \wedge x_5) \vee (x_1 \wedge x_3 \wedge x_5) \vee (x_2 \wedge x_3 \wedge x_4)$

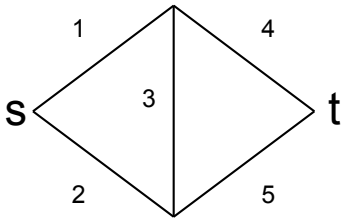
Computing Reliability of Probabilistic Graphs

Probability of existence of an $s - t$ path is the probability of the graph having one of the following 16 configurations:

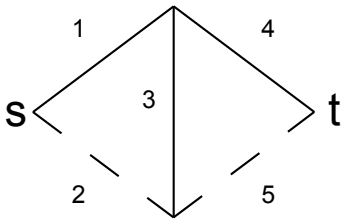


Computing Reliability of Probabilistic Graphs

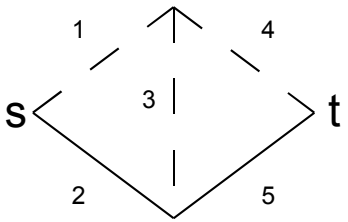
Each graph configuration has a probability of occurrence.



$$P(e_1 = 1, e_2 = 1, e_3 = 1, e_4 = 1, e_5 = 1) = r_1 \cdot r_2 \cdot r_3 \cdot r_4 \cdot r_5$$



$$P(e_1 = 1, e_2 = 0, e_3 = 1, e_4 = 1, e_5 = 0) = r_1 \cdot (1 - r_2) \cdot r_3 \cdot r_4 \cdot (1 - r_5)$$



$$P(e_1 = 0, e_2 = 1, e_3 = 0, e_4 = 0, e_5 = 1) = (1 - r_1) \cdot r_2 \cdot (1 - r_3) \cdot (1 - r_4) \cdot r_5$$

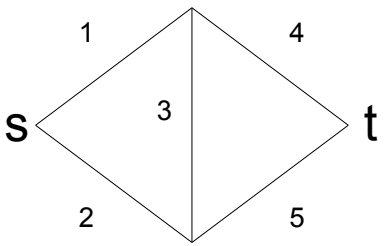
Computing Reliability of Probabilistic Graphs

Probability of occurrence of one of the feasible graph configurations is the sum of the probabilities of each configuration.

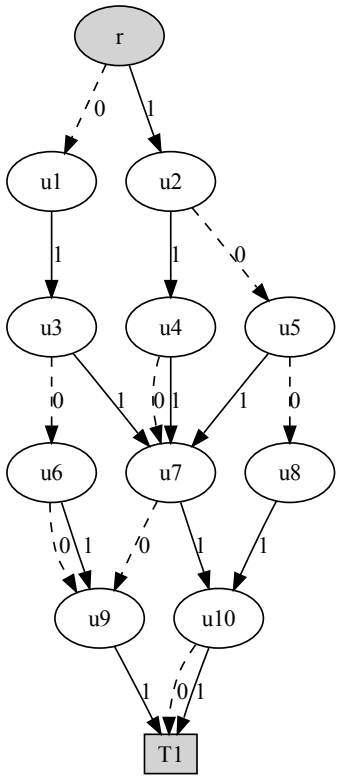
$$
\begin{aligned}
P_{st} = \quad & P(e_1 = 1, e_2 = 1, e_3 = 1, e_4 = 1, e_5 = 1) + \\
& P(e_1 = 1, e_2 = 1, e_3 = 1, e_4 = 1, e_5 = 0) + \\
& \ldots \\
& P(e_1 = 0, e_2 = 1, e_3 = 0, e_4 = 0, e_5 = 1)
\end{aligned}
$$

Computing Reliability of Probabilistic Graphs

We compute a BDD representing the set of all feasible assignments.

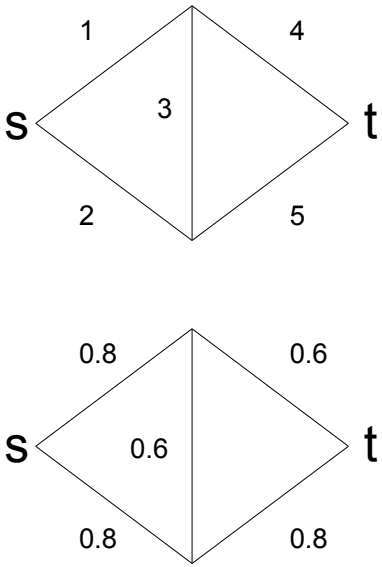


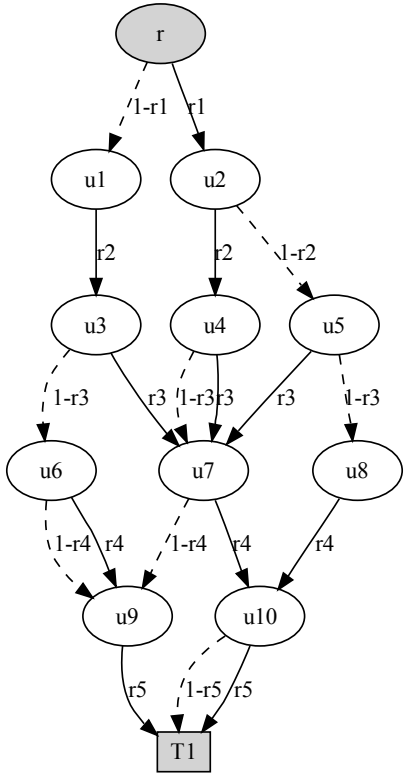
e_1	e_2	e_3	e_4	e_5
1	1	1	1	1
1	1	1	1	0
1	1	1	0	1
1	1	0	1	1
1	1	0	1	0
1	1	0	0	1
1	0	1	1	1
1	0	1	1	0
1	0	1	0	1
1	0	0	1	1
1	0	0	1	0
0	1	1	1	1
0	1	1	1	0
0	1	1	0	1
0	1	0	1	1
0	1	0	0	1

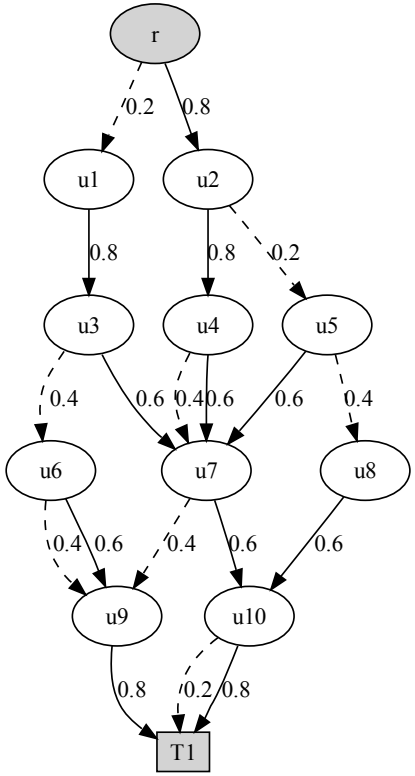


Computing Reliability of Probabilistic Graphs

For given edge reliabilities $r_1, \ldots, r_n$ we label BDD edges correspondingly: $e_i = 1$ edge with r_i and $e_i = 0$ edge with $1 - r_i$







Use recursive evaluation over BDD (linear in BDD size):
$Rel(\mathbf{1}) = 1$, $Rel(u) = \sum_{e:u \to u'} r_e \cdot Rel(u')$.

Network Design Problem

- In previous example we *evaluate* reliability function *Rel* for fixed values: $Rel(0.8, 0.8, 0.6, 0.6, 0.8) = 0.85504$. And this alone is an NP-hard problem!
- In a *network design problem* we have *multiple options* for each edge. Let $x_i \in D_i$ represent an implementation of edge e_i. For each implementation we have a *reliability* $r_i(x_i)$ and a *cost* $c_i(x_i)$. For a full assignment $x_1, \ldots, x_n$ we can compute reliability of the system $Rel(x_1, \ldots, x_n)$ and its (additive) cost $C(x_1, \ldots, x_n) = \sum_i c_i(x_i)$.
- We are interested in finding an implementation of the system $x_1 = v_1, \ldots, x_n = v_n$ that satisfies requirements for minimal reliability R_{min} and maximal budget C_{max}

Multicriteria Queries

For a given reliability function $Rel(x_1, \ldots, x_n)$ and given cost function $C(x_1, \ldots, x_n) = \sum_i c_i(x_i)$ we consider the following queries:

Q1 - Minimal Cost for a given Reliability

$$min\ C(\overrightarrow{x}),\ \text{s.t.}\ Rel(\overrightarrow{x}) \geq R_{min}.$$

Q2 - Maximal Reliability within a Budget

$$max\ Rel(\overrightarrow{x}),\ \text{s.t.}\ C(\overrightarrow{x}) \leq C_{max}.$$

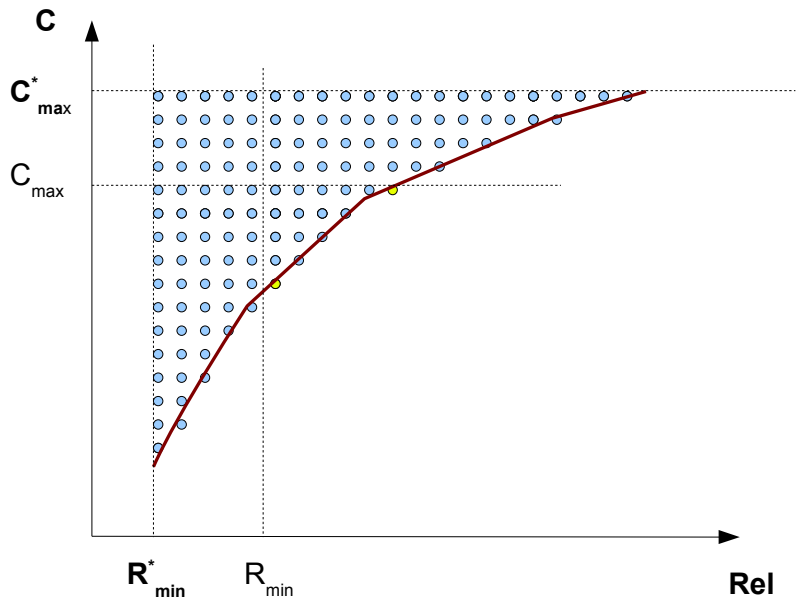
Q3 - Feasibility of a Component Given $x_i, v \in D_i$, is there a solution $\overrightarrow{x}$ containing $x_i = v$ s.t. $Rel(\overrightarrow{x}) \geq R_{min}$ and $C(\overrightarrow{x}) \leq C_{max}$.

Answering Q1 and Q2

We answer Q1 and Q2 by generating an *efficient frontier* - the set of all *nondominated* pairs $(Rel(x), C(x))$.

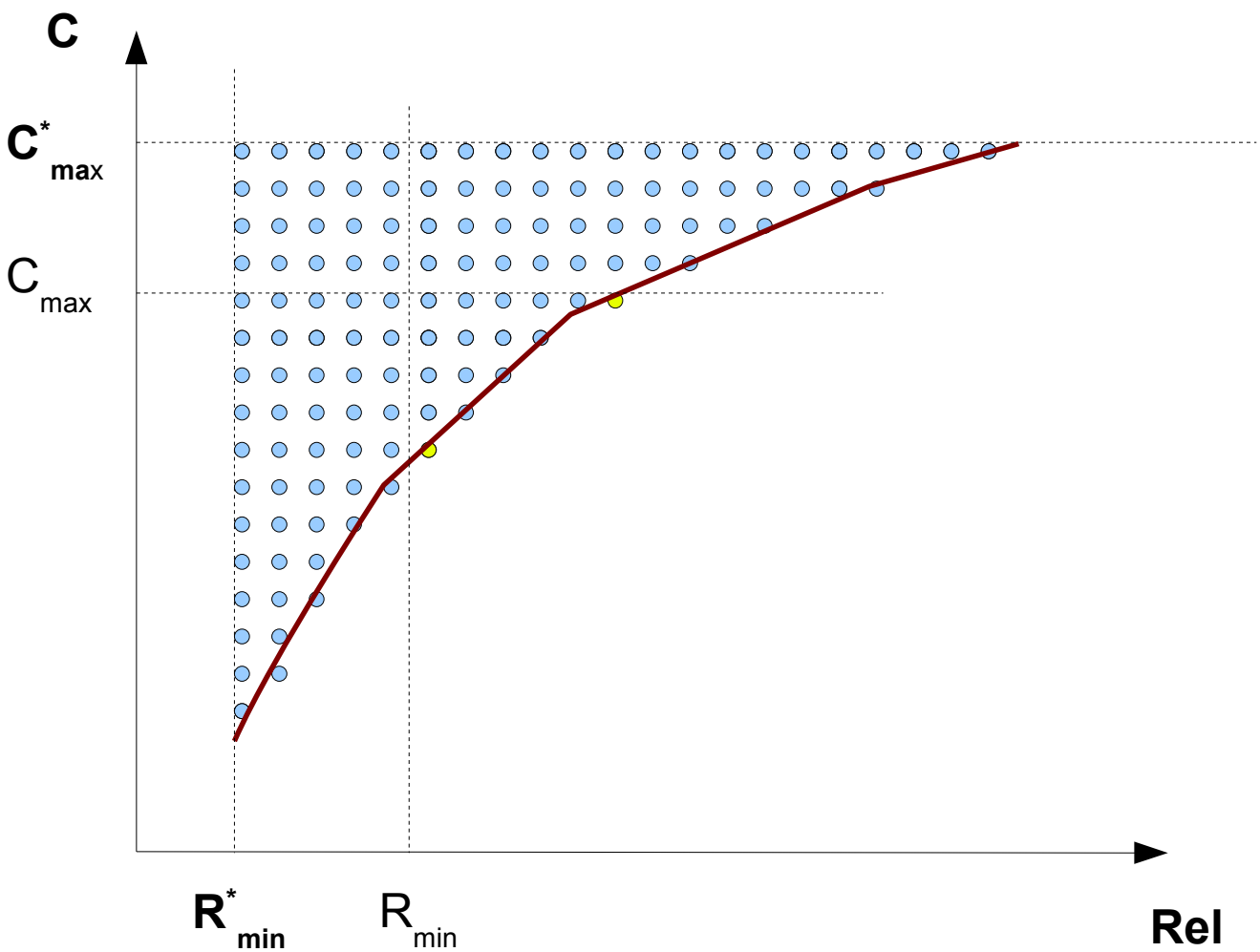
Definition (Domination)

Pair $(Rel(\overrightarrow{x}_1), C(\overrightarrow{x}_1))$ *is dominated by* $(Rel(\overrightarrow{x}_2), C(\overrightarrow{x}_2))$, iff $Rel(\overrightarrow{x}_1) \leq Rel(\overrightarrow{x}_2) \wedge C(\overrightarrow{x}_1) \geq C(\overrightarrow{x}_2)$



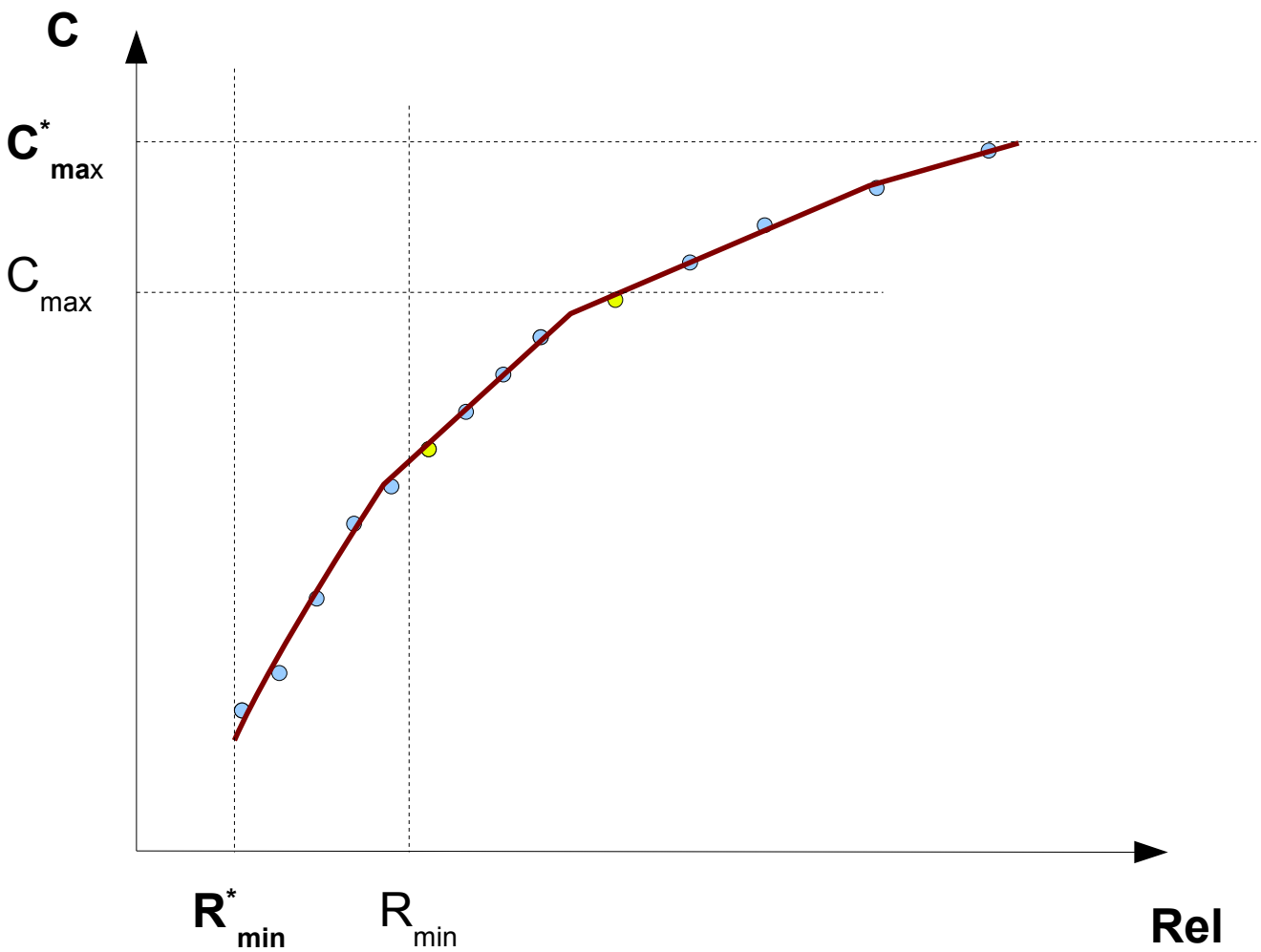
Answering Q1 and Q2

Efficient frontier and the dominated feasible assignments.



Answering Q1 and Q2

We compute only the efficient frontier.



Efficient Frontier Generation for Probabilistic Graphs

- We generate efficient frontier through *exhaustive depth-first search*
- The algorithm takes the number of steps linear in the number of feasible (possibly dominated) solutions
- In the worst case we have to evaluate function $Rel(x_1, \ldots, x_n)$ (NP-Hard) exponential number of times.
- Do we have to solve an NP-hard problem from scratch every time we evaluate $Rel(x_1, \ldots, x_n)$?

Efficient Frontier Generation for Probabilistic Graphs

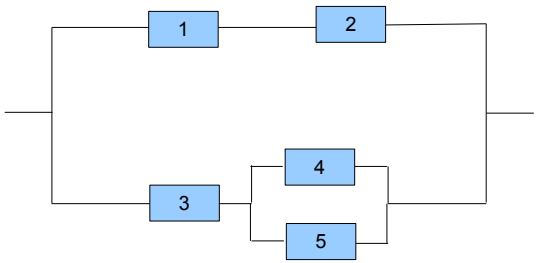
- The answer is *no*.
- We use a BDD representation of the reliability function. We generate BDD encoding of graph topology *only once*, and then repeatedly apply different labels $r_1(x_1), \ldots, r_n(x_n)$.

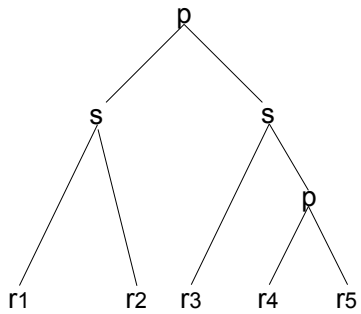
Efficient Frontier Generation for Probabilistic Graphs

- Furthermore, we can do this *incrementally* by restricting DFS branching order to the *BDD variable ordering* and using *intelligent caching* techniques over BDD nodes
- We never explore infeasible partial assignments: we exploit *monotonicity of reliability function*

Efficient Frontier Generation for Block Diagrams

- For a reliability function given as a block diagram we generate efficient frontier over *syntax trees*





- We initialize efficient lists at each leaf node. For a leaf node u_i corresponding to x_i we set $L = \{(r_i(v), c_i(v)) | v \in D_i\}$
- We aggregate lists at each internal node. For a node u with a left child list L_l and right child list L_r we set $L \leftarrow L_l \otimes_u L_r$
- $(r_1, c_1) \otimes_u (r_2, c_2) = (r_1 \otimes_u r_2, c_1 + c_2)$. For a serial node $r_1 \otimes_u r_2 = r_1 \cdot r_2$. For a parallel node $r_1 \otimes_u r_2 = 1 - (1 - r_1) \cdot (1 - r_2)$.

Experiments

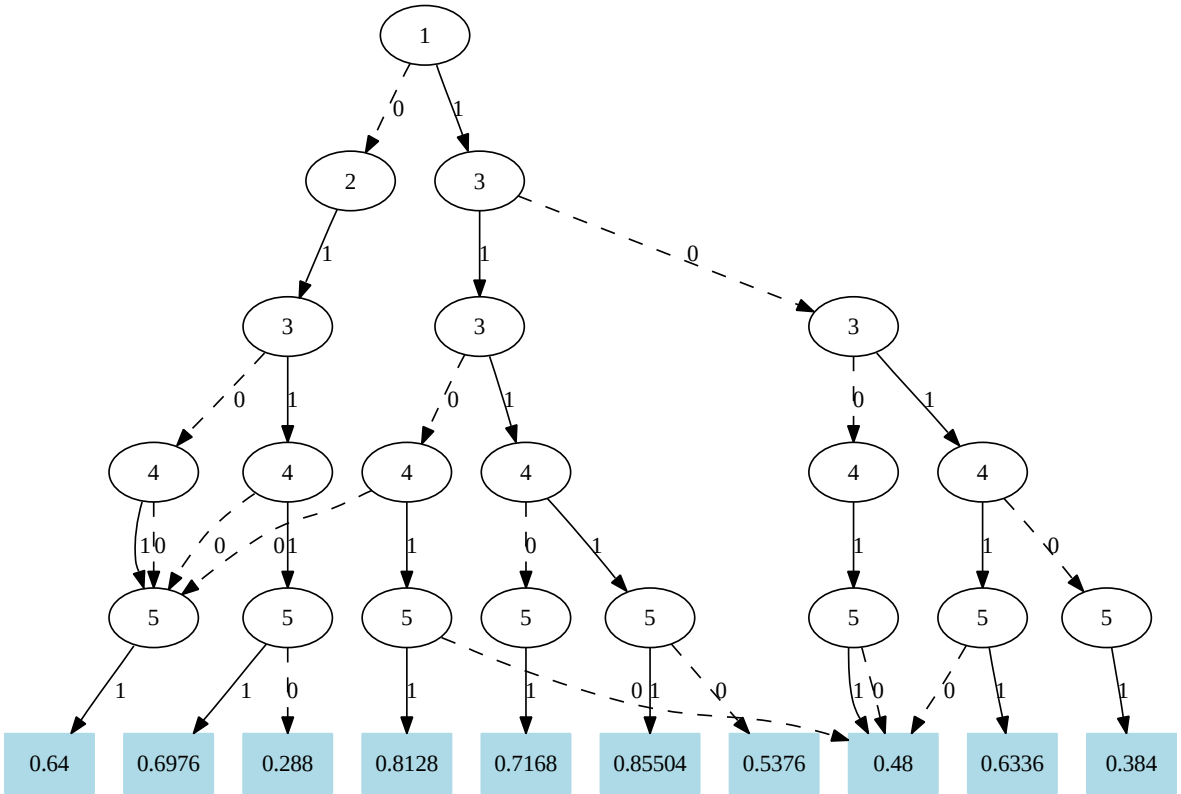
Vars	Avg			Max		
	t_{rbd}	t_{mdd}	$t_{mdd}^{0.9}$	t_{rbd}	t_{mdd}	$t_{mdd}^{0.9}$
20	0.0003	0.037	0.008	0.01	0.54	0.51
23	0.0006	0.756	0.137	0.01	4.47	3.58
24	0.0007	1.295	0.22	0.01	8.99	7.57
25	0.0008	3.769	0.29	0.01	23.69	11.45
26	0.0010	5.902	0.36	0.01	31.59	31.23
27	0.0017	30.52	1.80	0.01	119.91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

Table: 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.

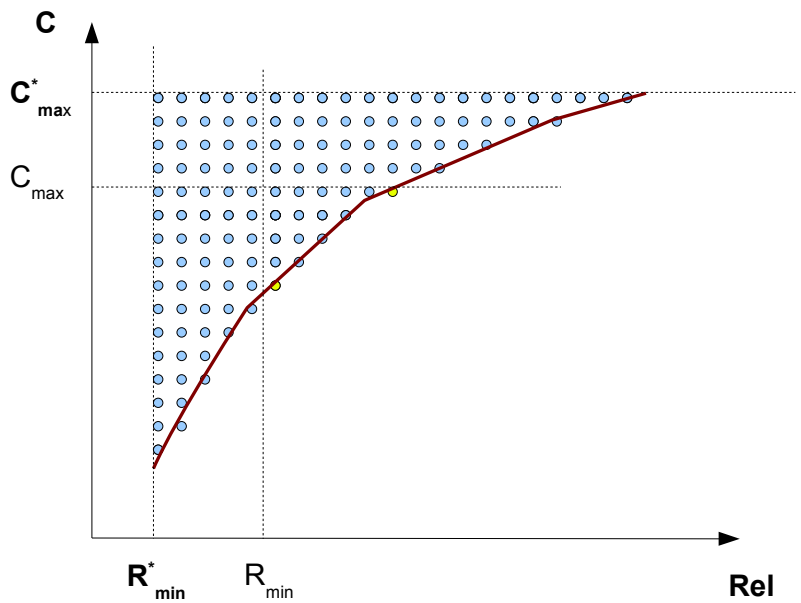
Answering Q3

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

We construct a *multi-terminal decision diagram* (MTMDD).

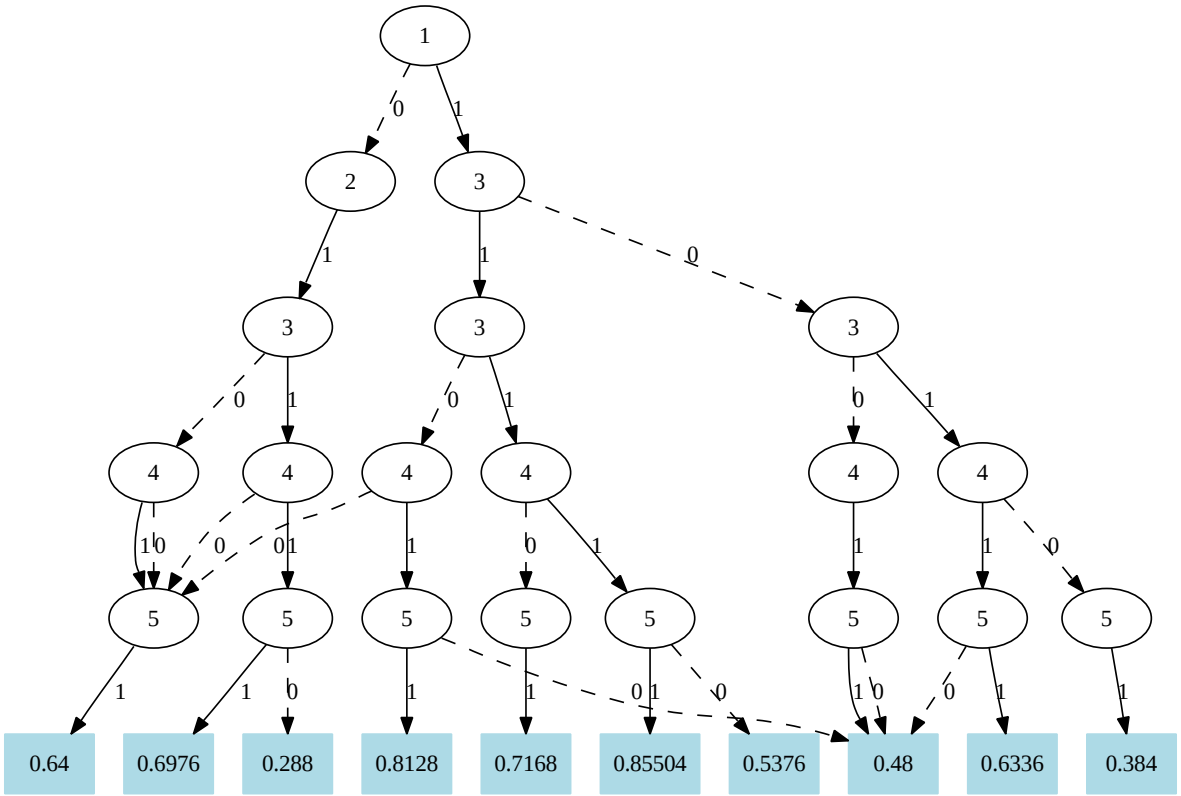


Multi-terminal decision diagram represents non-dominated solutions as well.



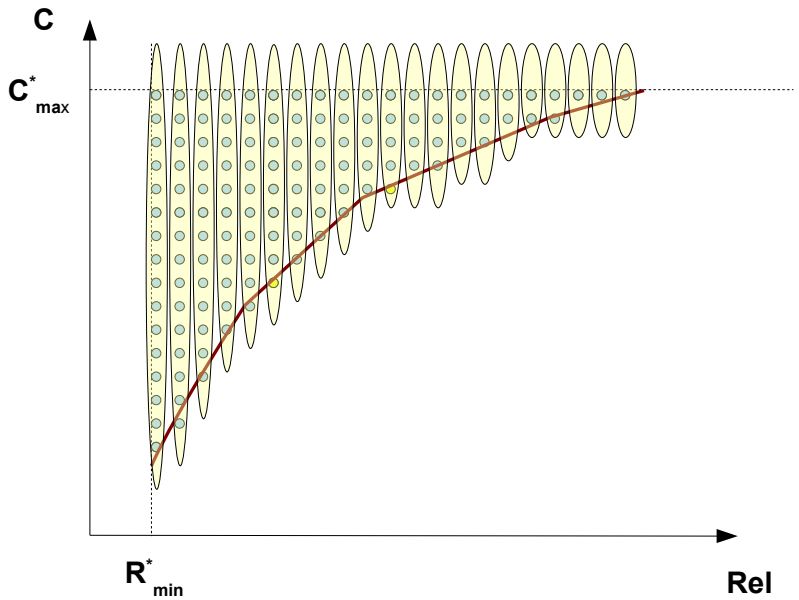
Answering Q3

- Every path corresponds to an assignment to $\overrightarrow{x} = (x_1, \ldots, x_n)$ variables.
- Every terminal node corresponds to the reliability of the assignment $Rel(x_1, \ldots, x_n)$ or to the *reliability interval*.
- All isomorphic nodes are merged.



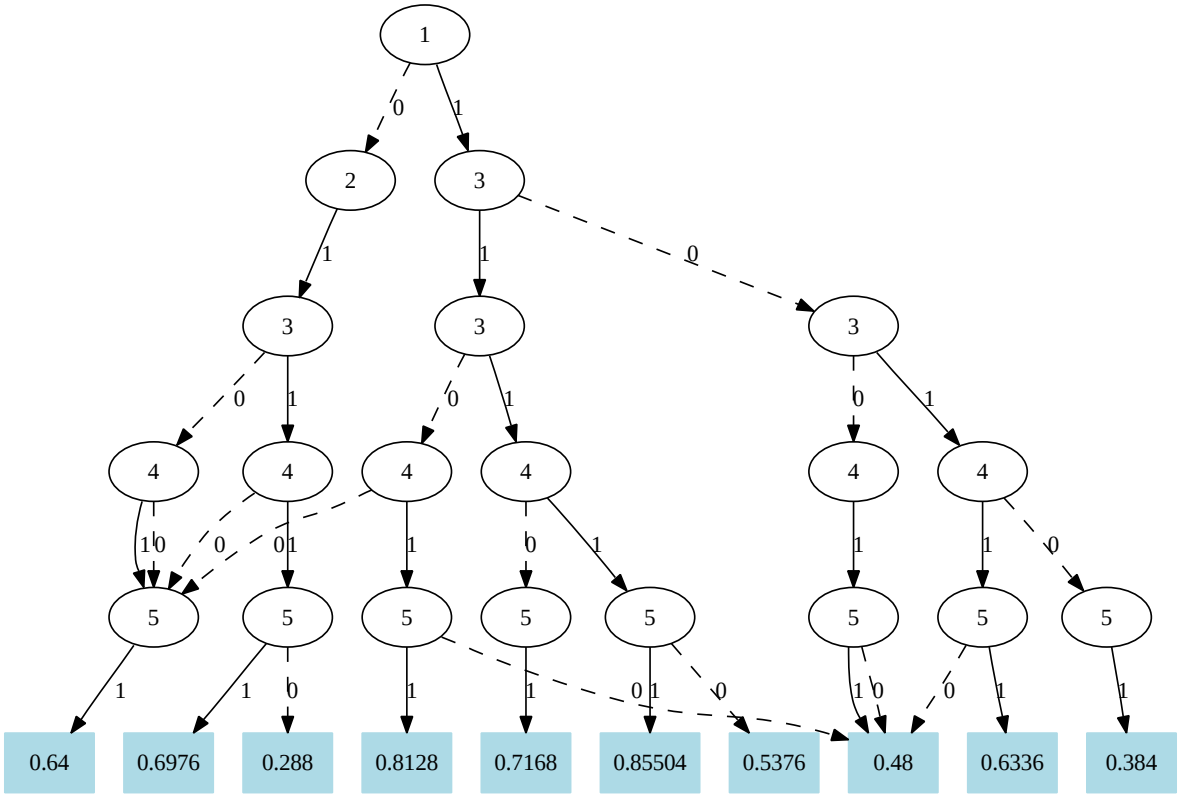
Answering Q3

- Every path corresponds to an assignment to $\overrightarrow{x} = (x_1, \ldots, x_n)$ variables.
- Every terminal node corresponds to the reliability of the assignment $Rel(x_1, \ldots, x_n)$ or to the *reliability interval*.
- All isomorphic nodes are merged.



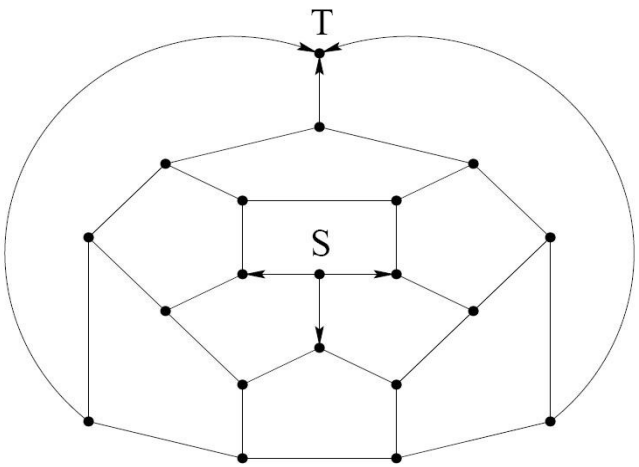
Answering Q3

- Label each edge $x_i = v$ with cost $c_i(v)$.
- Compute and store the shortest path values to the root node and to each terminal.
- Use local edge-based pruning to answer Q3.



Experiments

We evaluated performance on a large instance from the literature, containing 20 nodes. Corresponding merged MDD has 2414 nodes and 4730 edges.



Experiments

Scenario	d	Distr	r_{min}	Avg Term	Avg V	Avg E	Avg t
Wireless	10^2	linear	0.80	9	37,422	72,206	25.6s
	10^3	linear	0.80	95	78,716	147,271	25.7s
	10^4	linear	0.80	941	108,335	198,688	25.6s
Wired	7	expo	0.95	4	22,505	44,494	2m 53s
	7	expo	0.98	4	19,568	38,831	5m 31s
	7	expo	0.99	4	19,099	37,978	5m 37s

Table: 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 compilation time.