COLUMN ORIENTED COMPILATION OF VARIANT TABLES

Albert Haag
SAP SE, Germany
(partial-retirement)
email: albert.haag@t-online.de

Disclaimer:
All work on my papers “Arc Consistency with Negative Variant Table” and “Column Oriented Compilation of Variant Tables” was performed privately during the last two years after transition into partial retirement.

It and the accompanying implementation are neither endorsed by SAP nor do they reflect ongoing SAP development.

Notwithstanding:
The motivation for this work lies in my past at SAP and is based on insights and experiences with the SAP product configurators. The terminology follows that used in conjunction with the SAP Variant Configurator and originates from roots in the manufacture of „variants“.

See paper(s) for references

Variant Table (VTAB)

- Originally, list of product variants

Variant Table (VTAB)

- Originally, list of product variants
- Example product: T-Shirt

Color	Size	Print
Black	Small	MIB (Men in Black)
Black	Medium	MIB
Black	Large	MIB
Black	Medium	STW (Save the Whales)
Black	Large	STW
Red	Medium	STW
Red	Large	STW
White	Medium	STW
White	Large	STW
Blue	Medium	STW
Blue	Large	STW

Variant Table (VTAB)

- Originally, list of product variants
- Example product: T-Shirt
  - 4 colors, 3 sizes, 2 prints
  - 24 possible variants
  - 11 legal ones listed in VTAB
  - Others illegal

Color	Size	Print
Black	Small	MIB (Men in Black)
Black	Medium	MIB
Black	Large	MIB
Black	Medium	STW (Save the Whales)
Black	Large	STW
Red	Medium	STW
Red	Large	STW
White	Medium	STW
White	Large	STW
Blue	Medium	STW
Blue	Large	STW

Variant Table (VTAB)

- Originally, list of product variants
- Example product: T-Shirt
  - 4 colors, 3 sizes, 2 prints
  - 24 possible variants
  - 11 legal ones listed in VTAB
  - Others illegal
- More generally, list of allowed combinations of product properties in tabular form

Color	Size	Print
Black	Small	MIB (Men in Black)
Black	Medium	MIB
Black	Large	MIB
Black	Medium	STW (Save the Whales)
Black	Large	STW
Red	Medium	STW
Red	Large	STW
White	Medium	STW
White	Large	STW
Blue	Medium	STW
Blue	Large	STW

Variant Table (VTAB)

- Originally, list of product variants
- Example product: T-Shirt
  - 4 colors, 3 sizes, 2 prints
  - 24 possible variants
  - 11 legal ones listed in VTAB
  - Others illegal
- More generally, list of allowed combinations of product properties in tabular form
- Used in configuration
  - Primarily for arc-consistency (GAC)

Color	Size	Print
Black	Small	MIB (Men in Black)
Black	Medium	MIB
Black	Large	MIB
Black	Medium	STW (Save the Whales)
Black	Large	STW
Red	Medium	STW
Red	Large	STW
White	Medium	STW
White	Large	STW
Blue	Medium	STW
Blue	Large	STW

Variant Table (VTAB)

- Originally, list of product variants
- Example product: T-Shirt
  - 4 colors, 3 sizes, 2 prints
  - 24 possible variants
  - 11 legal ones listed in VTAB
  - Others illegal
- More generally, list of allowed combinations of product properties in tabular form
- Used in configuration
  - Primarily for arc-consistency (GAC)
  - As a database table (e.g. pricing)

Color	Size	Print
Black	Small	MIB (Men in Black)
Black	Medium	MIB
Black	Large	MIB
Black	Medium	STW (Save the Whales)
Black	Large	STW
Red	Medium	STW
Red	Large	STW
White	Medium	STW
White	Large	STW
Blue	Medium	STW
Blue	Large	STW

VTAB Maintenance (Modeling)

- VTAB maintenance distinct/separate from rest of configuration model

VTAB Maintenance (Modeling)

- VTAB maintenance distinct/separate from rest of configuration model
  - Different skill-set needed (easier)

VTAB Maintenance (Modeling)

- VTAB maintenance distinct/separate from rest of configuration model
  - Different skill-set needed (easier)
  - Different update cycle

VTAB Maintenance (Modeling)

- VTAB maintenance distinct/separate from rest of configuration model
  - Different skill-set needed (easier)
  - Different update cycle
  - Different knowledge sourcing

VTAB Maintenance (Modeling)

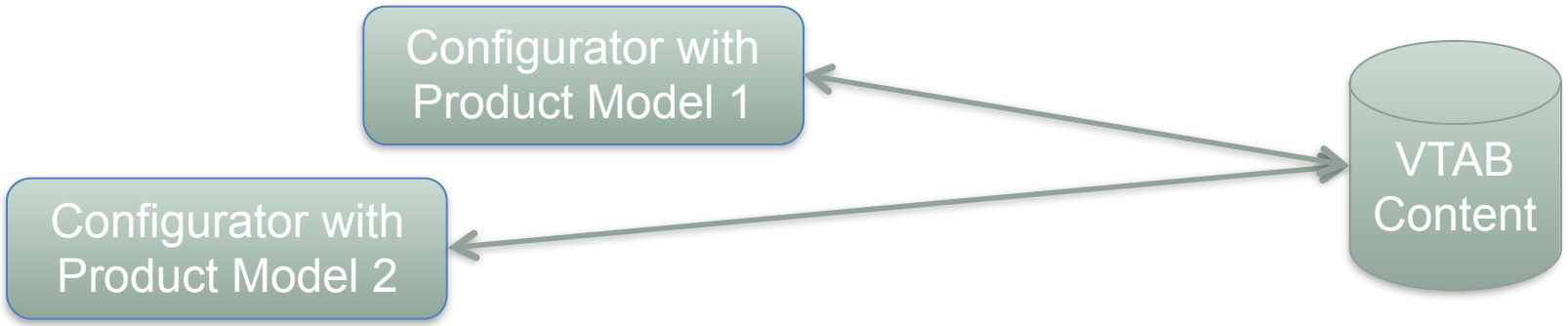
- VTAB maintenance distinct/separate from rest of configuration model
  - Different skill-set needed (easier)
  - Different update cycle
  - Different knowledge sourcing
  - Database-like

VTAB Maintenance (Modeling)

- VTAB maintenance distinct/separate from rest of configuration model
  - Different skill-set needed (easier)
  - Different update cycle
  - Different knowledge sourcing
  - Database-like
- VTAB changes can occur any time (typically daily)

VTAB Maintenance (Modeling)

- VTAB maintenance distinct/separate from rest of configuration model
  - Different skill-set needed (easier)
  - Different update cycle
  - Different knowledge sourcing
  - Database-like
- VTAB changes can occur any time (typically daily)



What is Needed During Configuration

What is Needed During Configuration

Fast “easy to read” queries of individual VTABs

What is Needed During Configuration

Fast “easy to read” queries of individual VTABs

In particular, support for arc-consistency (GAC algorithm)

What is Needed During Configuration

Fast “easy to read” queries of individual VTABs

In particular, support for arc-consistency (GAC algorithm)

- We want to “filter out” any values from any given domains that are not supported by a tuple in the VTAB

Known Approaches

Known Approaches

- VTABs in databases
  - Relational database
  - Column Oriented Databases

Known Approaches

- VTABs in databases
  - Relational database
  - Column Oriented Databases
- Decision diagrams: BDD/MDD/… [Knuth, Andersen et al.]

Known Approaches

- VTABs in databases
  - Relational database
  - Column Oriented Databases
- Decision diagrams: BDD/MDD/… [Knuth, Andersen et al.]
- Compression of VTABs into a small number of Cartesian products/cuboids, which I call *c-tuples* following [Katsirelos/Walsh]

My Approach

My Approach

- Experimented with BDD approach using JavaBDD but ran into issues
  - Tool complex and tool not completely suited to task at hand
  - Missing natural heuristics as starting point

My Approach

- Experimented with BDD approach using JavaBDD but ran into issues
  - Tool complex and tool not completely suited to task at hand
  - Missing natural heuristics as starting point
- Then explored a simple recursive VTAB decomposition
  - Yields a directed acyclic graph (DAG) which I call a Variant Decomposition Diagram (VDD)
  - Overriding development goals:
    - “Easy to implement”
    - fast lean development

Main Ideas of Decomposing a VTAB *T*

Print	Size	Color
MIB	Large	Black
MIB	Medium	Black
MIB	Small	Black
STW	Large	Black
STW	Large	Blue
STW	Large	Red
STW	Large	White
STW	Medium	Black
STW	Medium	Blue
STW	Medium	Red
STW	Medium	White

Columns reordered for ease of depiction:
3, 2, 1

Main Ideas of Decomposing a VTAB *T*

- Pick a value *x* in a column (*j*)

Print	Size	Color
MIB	Large	Black
MIB	Medium	Black
MIB	Small	Black
STW	Large	Black
STW	Large	Blue
STW	Large	Red
STW	Large	White
STW	Medium	Black
STW	Medium	Blue
STW	Medium	Red
STW	Medium	White

Columns reordered for ease of depiction:
3, 2, 1

Main Ideas of Decomposing a VTAB *T*

- Pick a value *x* in a column (*j*)
  - e.g., MIB in column 3

Print	Size	Color
MIB	Large	Black
MIB	Medium	Black
MIB	Small	Black
STW	Large	Black
STW	Large	Blue
STW	Large	Red
STW	Large	White
STW	Medium	Black
STW	Medium	Blue
STW	Medium	Red
STW	Medium	White

Columns reordered for ease of depiction:
3, 2, 1

Main Ideas of Decomposing a VTAB *T*

- Pick a value *x* in a column (*j*)
  - e.g., MIB in column 3

Print	Size	Color
MIB	Large	Black
MIB	Medium	Black
MIB	Small	Black
STW	Large	Black
STW	Large	Blue
STW	Large	Red
STW	Large	White
STW	Medium	Black
STW	Medium	Blue
STW	Medium	Red
STW	Medium	White

Columns reordered for ease of depiction:
3, 2, 1

Main Ideas of Decomposing a VTAB *T*

- Pick a value *x* in a column (*j*)
  - e.g., MIB in column 3
- *x* decomposes *T* into three parts

Print	Size	Color
MIB	Large	Black
MIB	Medium	Black
MIB	Small	Black
STW	Large	Black
STW	Large	Blue
STW	Large	Red
STW	Large	White
STW	Medium	Black
STW	Medium	Blue
STW	Medium	Red
STW	Medium	White

Columns reordered for ease of depiction:
3, 2, 1

Main Ideas of Decomposing a VTAB *T*

- Pick a value *x* in a column (*j*)
  - e.g., MIB in column 3
- *x* decomposes *T* into three parts
  - *B*(*T*,*j*,*x*): the cells in column *j* containing the value *x*

Print	Size	Color
MIB	Large	Black
MIB	Medium	Black
MIB	Small	Black
STW	Large	Black
STW	Large	Blue
STW	Large	Red
STW	Large	White
STW	Medium	Black
STW	Medium	Blue
STW	Medium	Red
STW	Medium	White

Columns reordered for ease of depiction:
3, 2, 1

Main Ideas of Decomposing a VTAB *T*

- Pick a value *x* in a column (*j*)
  - e.g., MIB in column 3
- *x* decomposes *T* into three parts
  - *B*(*T*,*j*,*x*): the cells in column *j* containing the value *x*
  - *L*(*T*,*j*,*x*): the rows in *T* that don't have value *x* in column *j*

Print	Size	Color
MIB	Large	Black
MIB	Medium	Black
MIB	Small	Black
STW	Large	Black
STW	Large	Blue
STW	Large	Red
STW	Large	White
STW	Medium	Black
STW	Medium	Blue
STW	Medium	Red
STW	Medium	White

Columns reordered for ease of depiction:
3, 2, 1

Main Ideas of Decomposing a VTAB *T*

- Pick a value *x* in a column (*j*)
  - e.g., MIB in column 3
- *x* decomposes *T* into three parts
  - *B*(*T*,*j*,*x*): the cells in column *j* containing the value *x*
  - *L*(*T*,*j*,*x*): the rows in *T* that don't have value *x* in column *j*
  - *R*(*T*,*j*,*x*): rest after removing *L*(*T*,*j*,*x*) and *B*(*T*,*j*,*x*)

Print	Size	Color
MIB	Large	Black
MIB	Medium	Black
MIB	Small	Black
STW	Large	Black
STW	Large	Blue
STW	Large	Red
STW	Large	White
STW	Medium	Black
STW	Medium	Blue
STW	Medium	Red
STW	Medium	White

Columns reordered for ease of depiction:
3, 2, 1

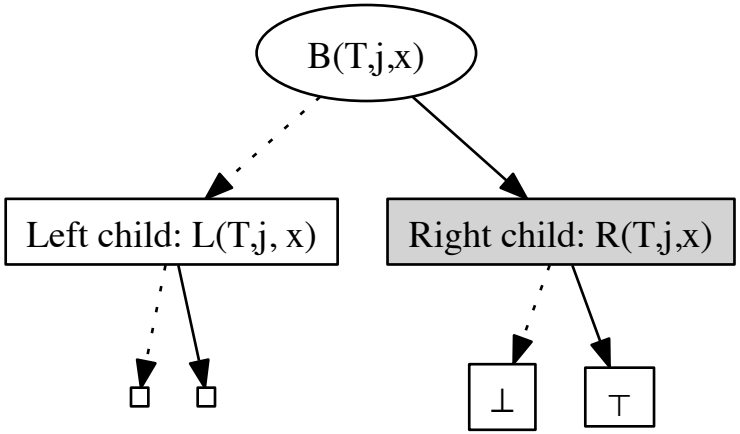
Main Ideas of Decomposing a VTAB *T*

- Pick a value *x* in a column (*j*)
  - e.g., MIB in column 3
- *x* decomposes *T* into three parts
  - *B*(*T*,*j*,*x*): the cells in column *j* containing the value *x*
  - *L*(*T*,*j*,*x*): the rows in *T* that don't have value *x* in column *j*
  - *R*(*T*,*j*,*x*): rest after removing *L*(*T*,*j*,*x*) and *B*(*T*,*j*,*x*)
- *L*(*T*,*j*,*x*) and *R*(*T*,*j*,*x*) in turn can be decomposed further

Print	Size	Color
MIB	Large	Black
MIB	Medium	Black
MIB	Small	Black
STW	Large	Black
STW	Large	Blue
STW	Large	Red
STW	Large	White
STW	Medium	Black
STW	Medium	Blue
STW	Medium	Red
STW	Medium	White

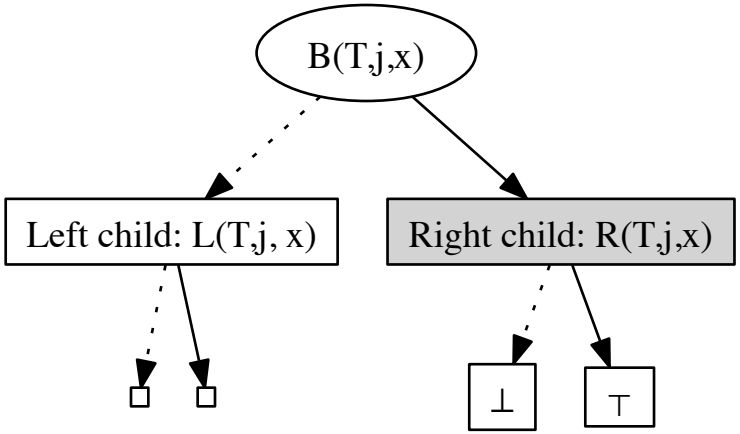
Columns reordered for ease of depiction:
3, 2, 1

Basic Graph Structure of a VDD (Variant Decomposition Diagram)



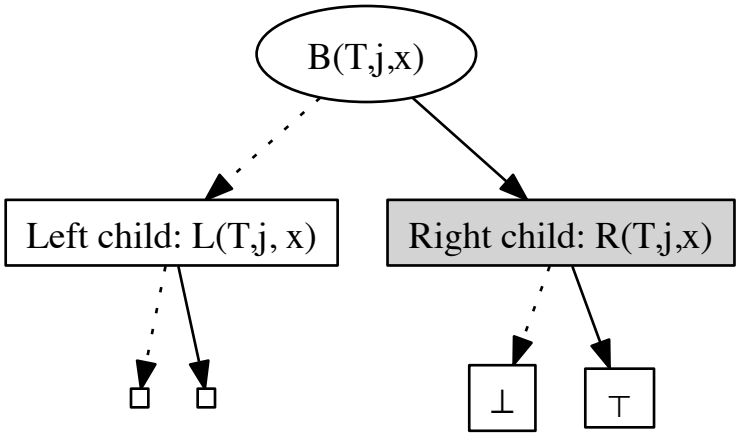
Basic Graph Structure of a VDD (Variant Decomposition Diagram)

- A root node represents all of *T* (here labeled *B*(*T*,*j*,*x*))



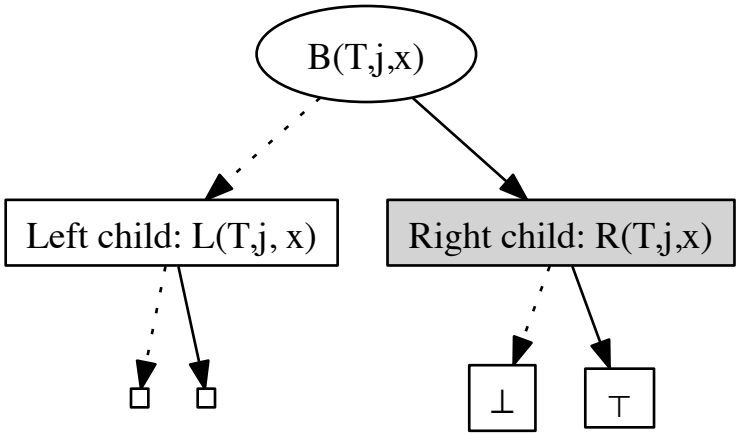
Basic Graph Structure of a VDD (Variant Decomposition Diagram)

- A root node represents all of *T* (here labeled *B*(*T*,*j*,*x*))
- *A LO* link (dotted line) points to a node representing *L*(*T*,*j*,*x*) ($v_j \neq x$)



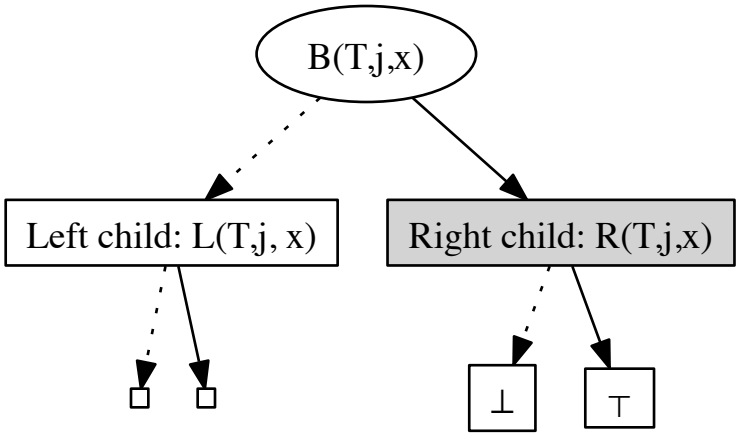
Basic Graph Structure of a VDD (Variant Decomposition Diagram)

- A root node represents all of *T* (here labeled *B*(*T*,*j*,*x*))
- *A LO* link (dotted line) points to a node representing *L*(*T*,*j*,*x*) ($v_j \neq x$)
- *A HI* link (filled line) points to a node representing *R*(*T*,*j*,*x*) ($v_j = x$)

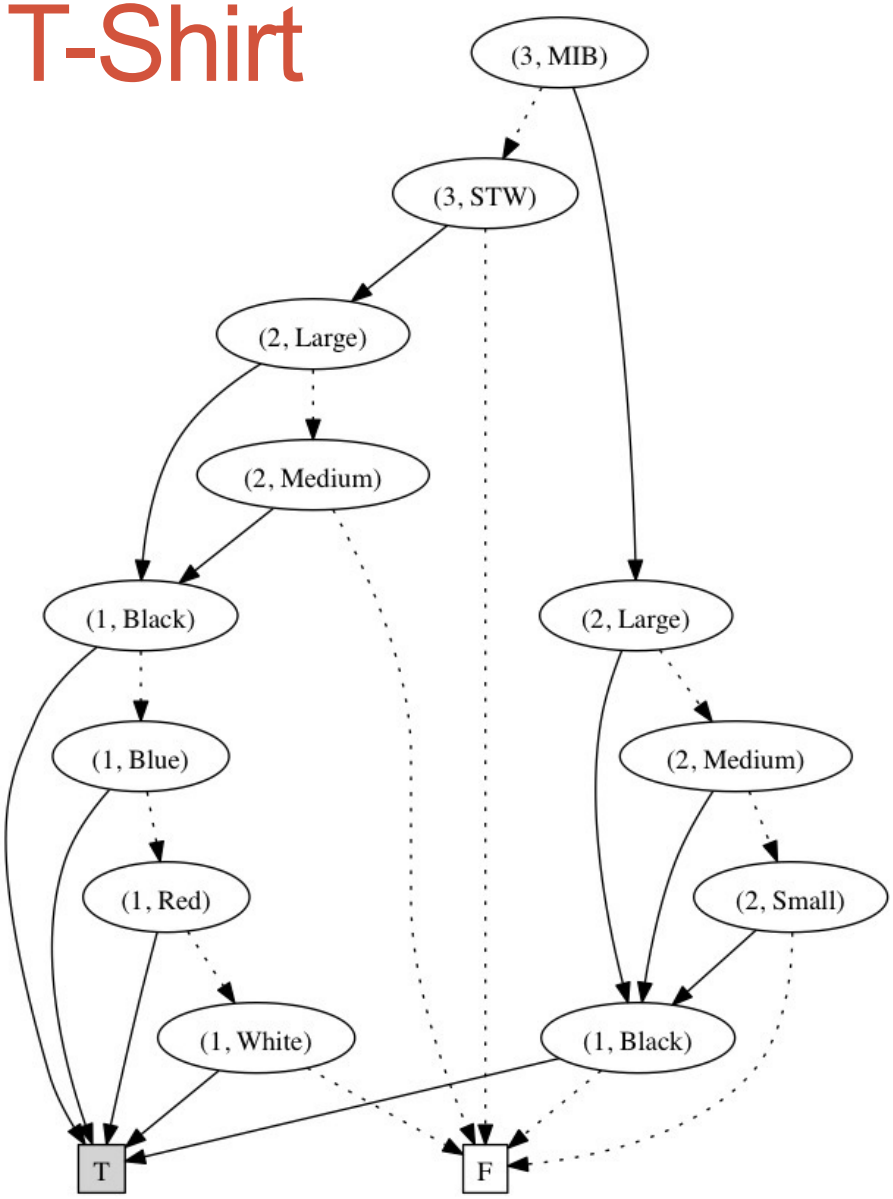


Basic Graph Structure of a VDD (Variant Decomposition Diagram)

- A root node represents all of *T* (here labeled *B*(*T*,*j*,*x*))
- *A LO* link (dotted line) points to a node representing *L*(*T*,*j*,*x*) ($v_j \neq x$)
- *A HI* link (filled line) points to a node representing *R*(*T*,*j*,*x*) ($v_j = x$)
- Terminal nodes (sinks)
  - An empty *L*(•,*j*,*x*) is represented by the predefined sink (node) ⊥ (FALSE)
  - An empty *R*(•,*j*,*x*) is represented by the predefined sink (node) ⊤ (TRUE)

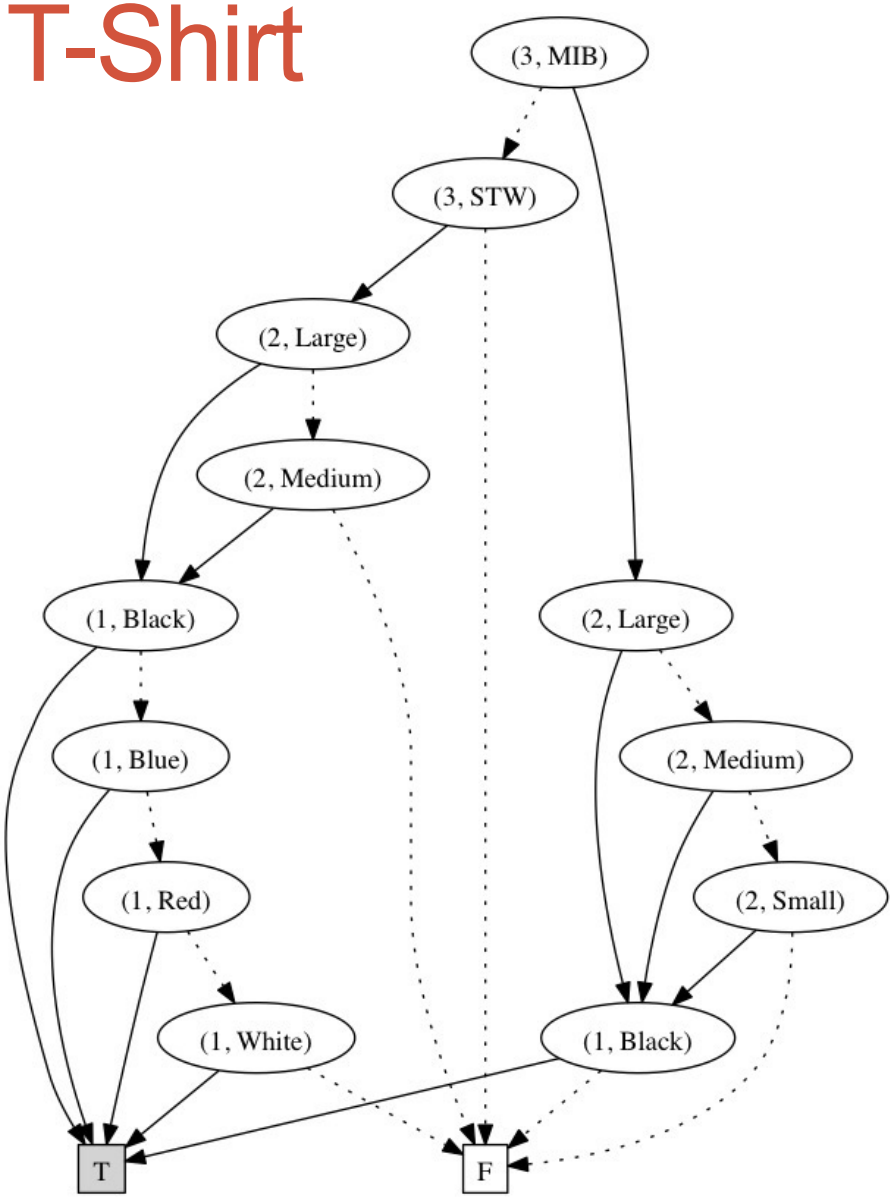


One VDD of VTAB T-Shirt



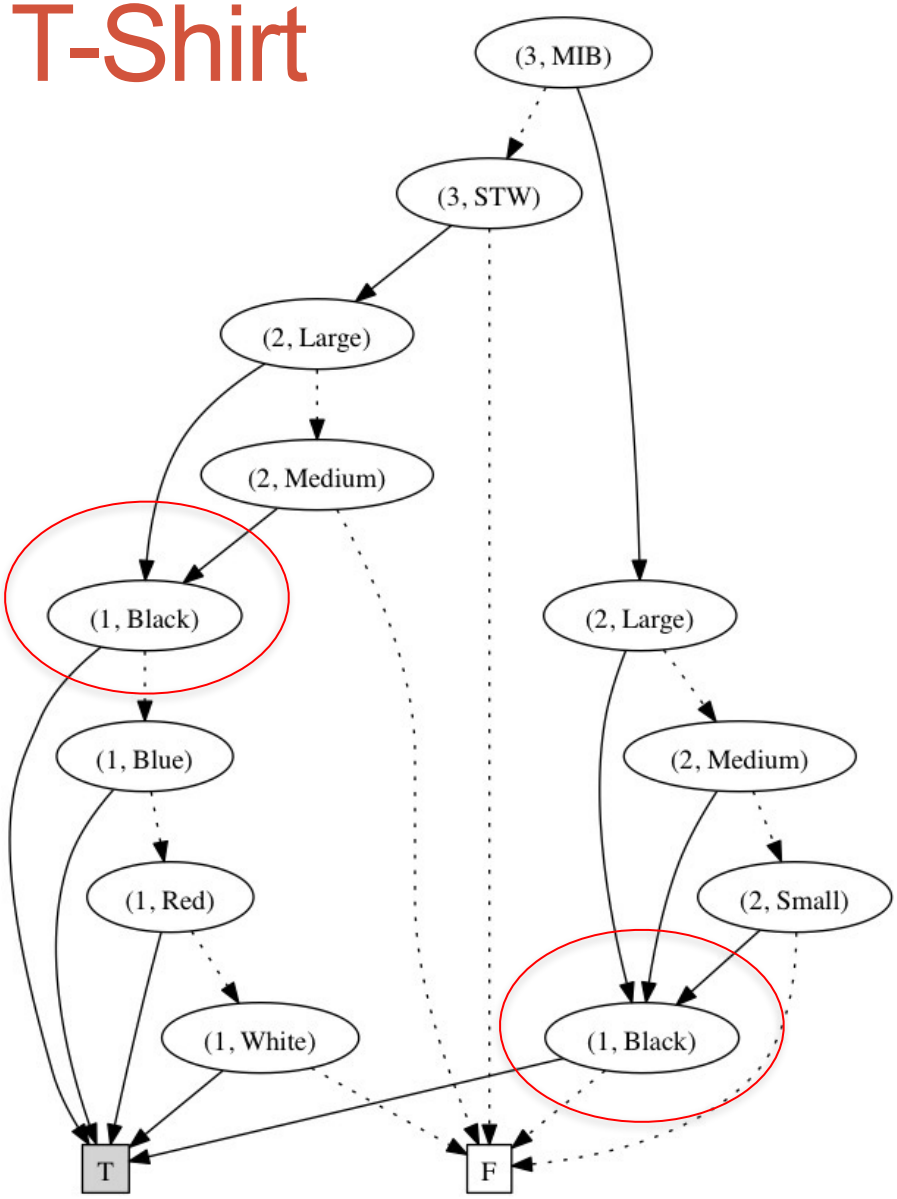
One VDD of VTAB T-Shirt

- Node label <(*j*, *x*)>



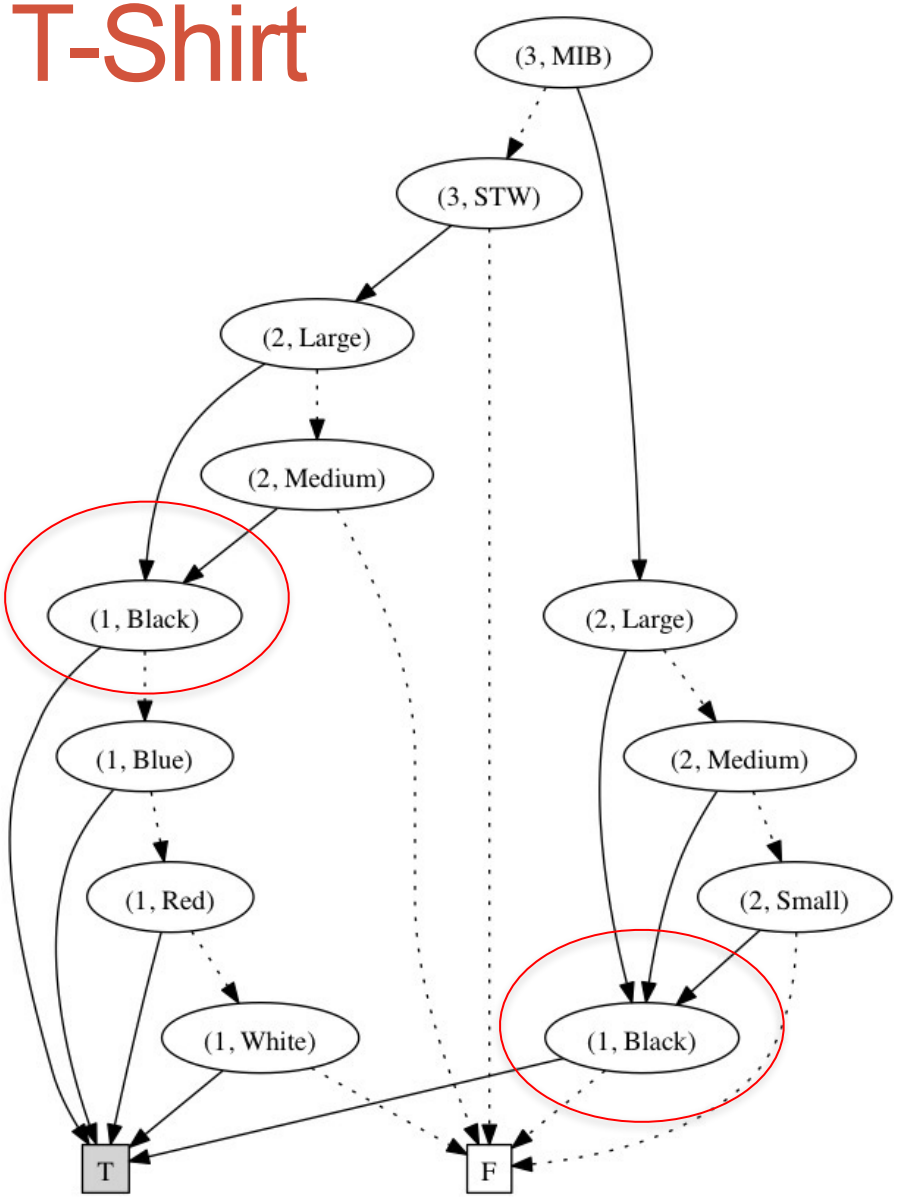
One VDD of VTAB T-Shirt

- Node label <(*j*, *x*)>
- Nodes for identical subtables (circled) are represented only once



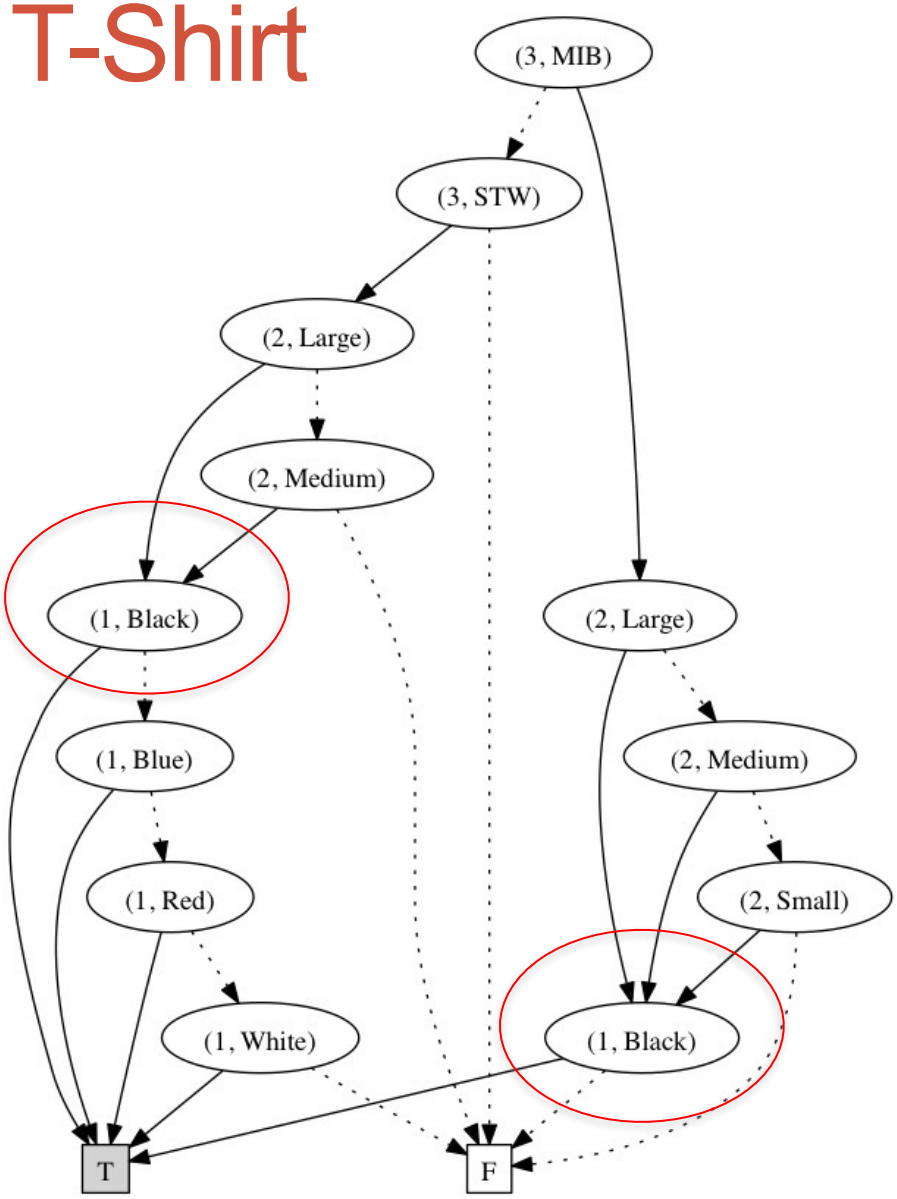
One VDD of VTAB T-Shirt

- Node label <(*j*, *x*)>
- Nodes for identical subtables (circled) are represented only once
- Results in a DAG (usually not a tree)



One VDD of VTAB T-Shirt

- Node label <(*j*, *x*)>
- Nodes for identical subtables (circled) are represented only once
- Results in a DAG (usually not a tree)
- With care could be interpreted as a ZDD (Zero-Supressed-DD) [Knuth]



Two Implemented Heuristics: h_1 and h_2

Two Implemented Heuristics: h_1 and h_2

***h*$_1$** is a *classical heuristic*:

Two Implemented Heuristics: h_1 and h_2

***h*$_1$** is a *classical heuristic*:

Calculates an estimated "utility" of all possible next choices of *x* and chooses a "best" one

Two Implemented Heuristics: h_1 and h_2

***h*₁** is a *classical heuristic*:

Calculates an estimated "utility" of all possible next choices of *x* and chooses a "best" one

***h*₂** is *very simple*:

Two Implemented Heuristics: h_1 and h_2

***h*$_1$** is a *classical heuristic*:

Calculates an estimated "utility" of all possible next choices of *x* and chooses a "best" one

***h*$_2$** is *very simple*:

Choose first value in first column at each step

Two Implemented Heuristics: h_1 and h_2

***h*$_1$** is a *classical heuristic*:

Calculates an estimated "utility" of all possible next choices of *x* and chooses a "best" one

***h*$_2$** is *very simple*:

Choose first value in first column at each step

- Assumes VTAB is sorted

Two Implemented Heuristics: h_1 and h_2

***h*$_1$** is a *classical heuristic*:

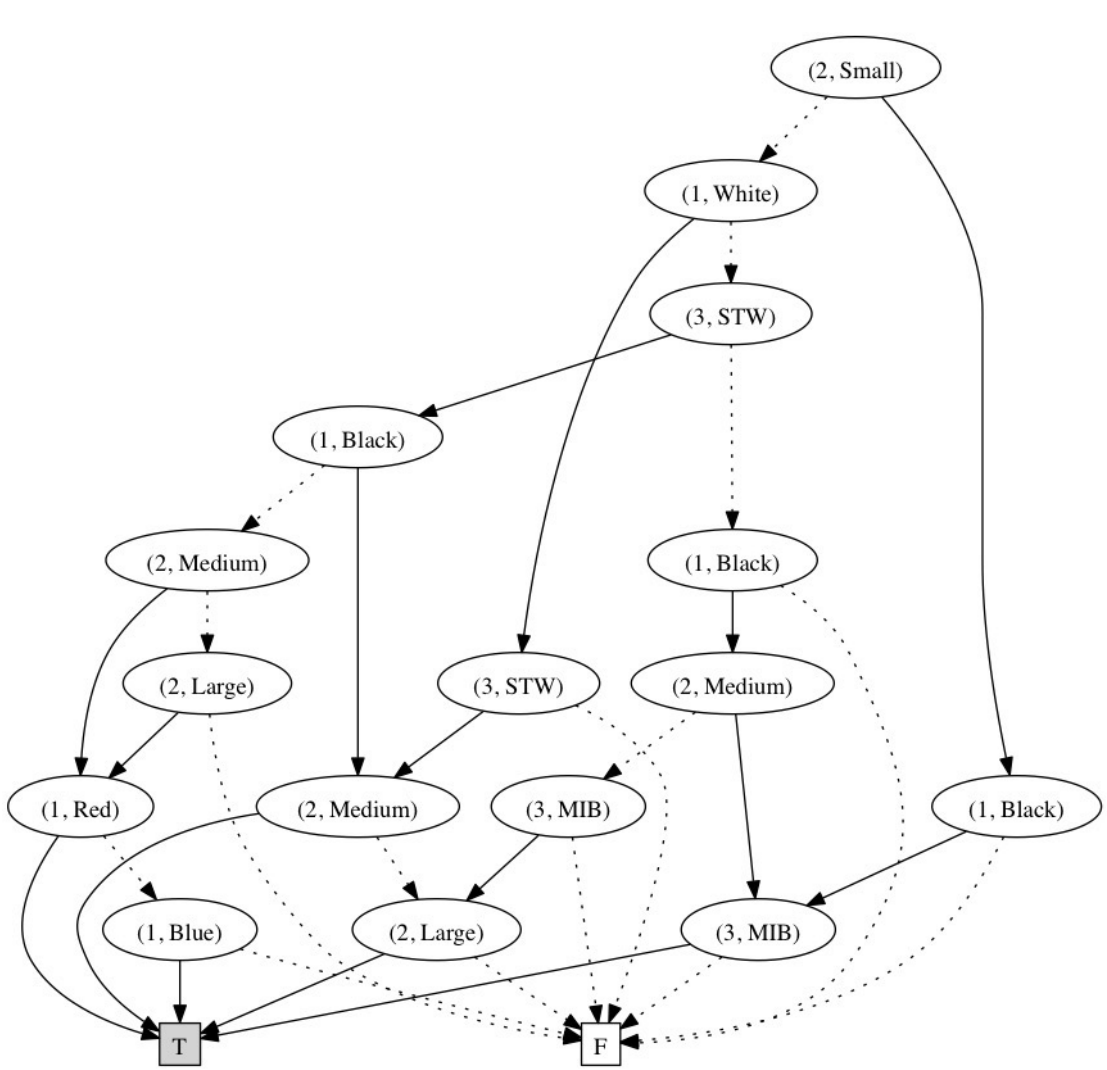
Calculates an estimated "utility" of all possible next choices of *x* and chooses a "best" one

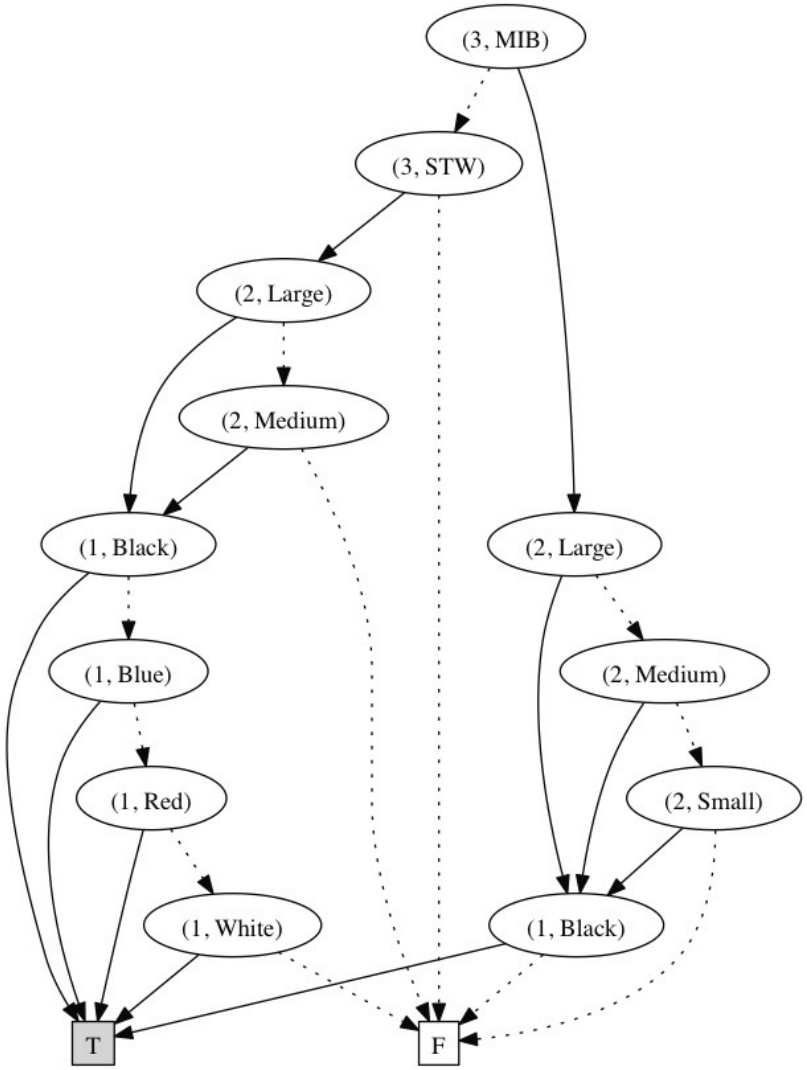
***h*$_2$** is *very simple*:

Choose first value in first column at each step

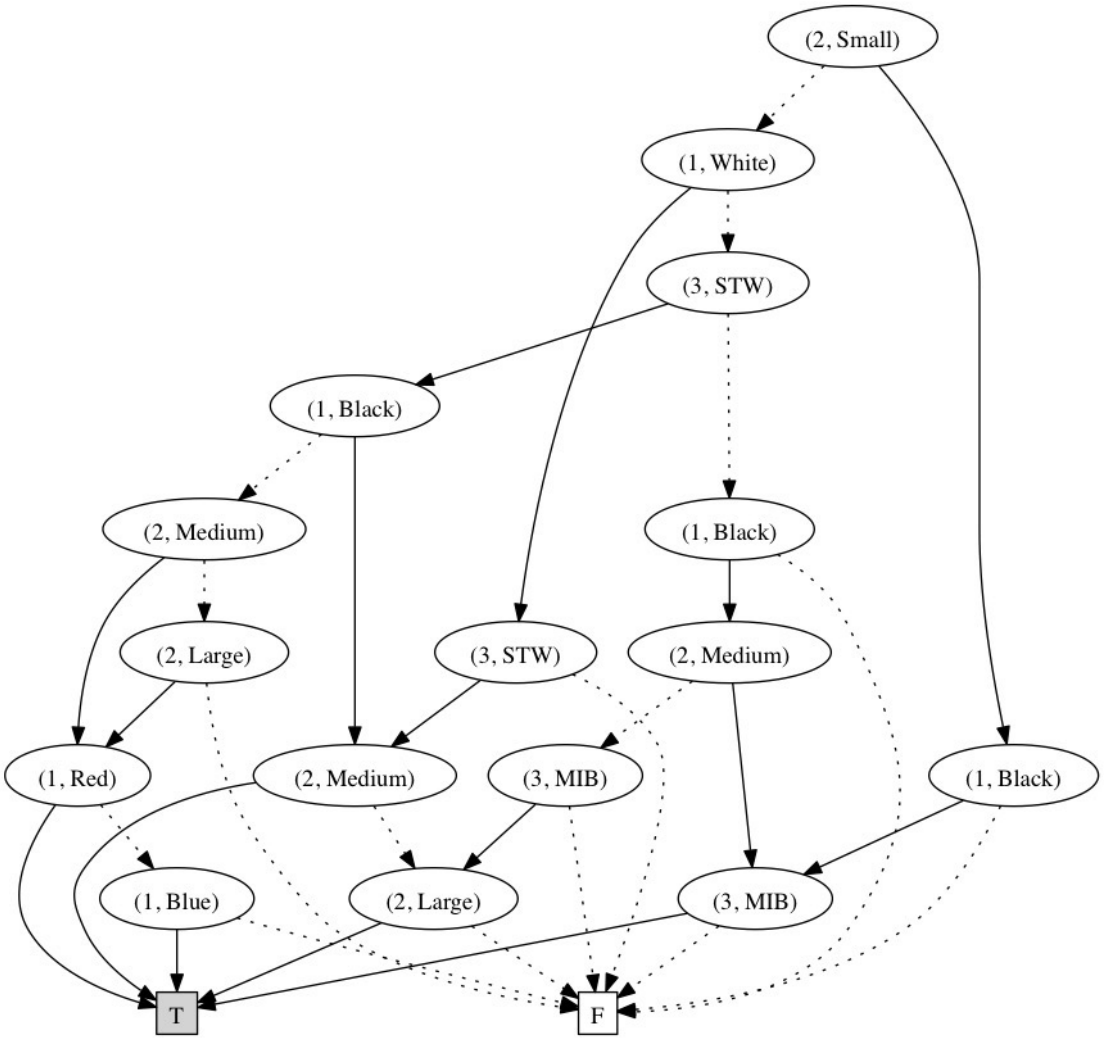
- Assumes VTAB is sorted
- My results are obtained with also sorting the columns by number of distinct occurring values (row with most values last – *Print*, *Size*, *Color* in t-shirt example)

T-Shirt: VDDs for Heuristics h_1 and h_2

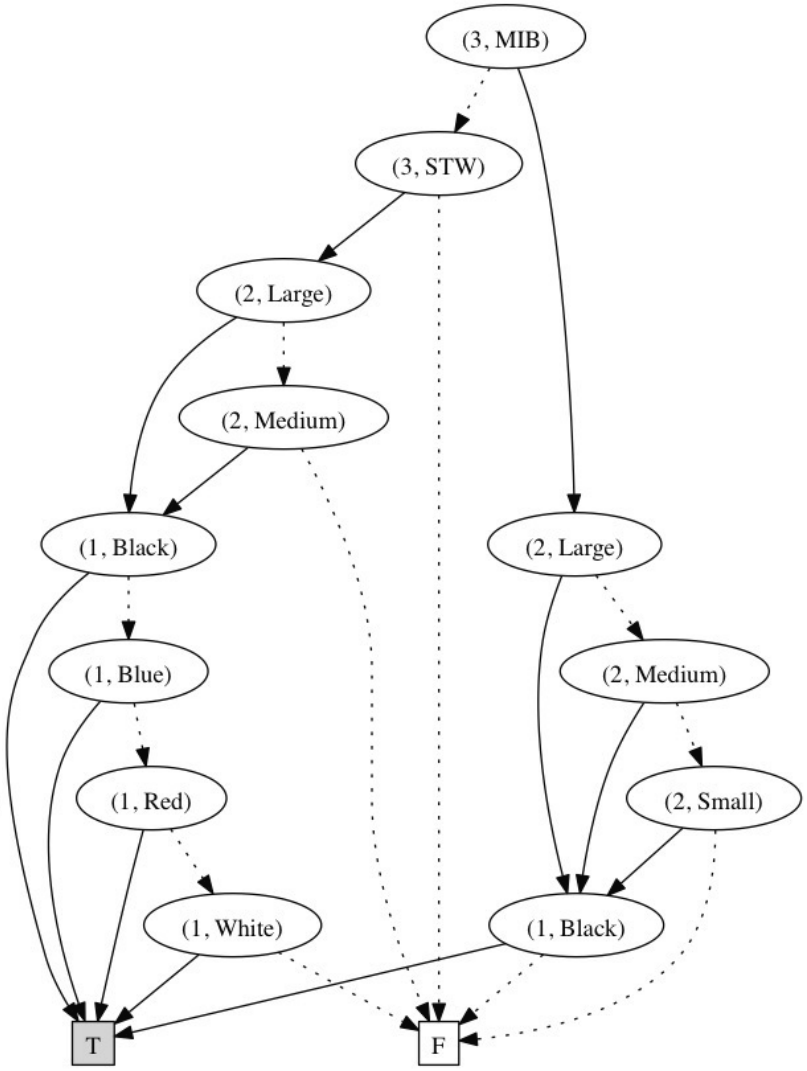




T-Shirt: VDDs for Heuristics h_1 and h_2



h_1: 16 nodes



h_2: 12 nodes

Comparison of Heuristics h_1 and h_2

Comparison of Heuristics h_1 and h_2

- Test bed: 238 “real” customer VTABs

Comparison of Heuristics h_1 and h_2

- Test bed: 238 “real” customer VTABs
- h_2 outperforms h_1 in most (but not all) cases

Comparison of Heuristics h_1 and h_2

- Test bed: 238 “real” customer VTABs
- h_2 outperforms h_1 in most (but not all) cases
- Compilation of largest VTAB takes
  - 20 min for h_1
  - 0.6 sec for h_2

Comparison of Heuristics h_1 and h_2

- Test bed: 238 “real” customer VTABs
- h_2 outperforms h_1 in most (but not all) cases
- Compilation of largest VTAB takes
  - 20 min for h_1
  - 0.6 sec for h_2
- Overall h_2 clearly outperforms h_1

Set-labeled Nodes

Set-labeled Nodes

- SAP Variant Configurator allows VTABs with real-valued continuous intervals in cells

Set-labeled Nodes

- SAP Variant Configurator allows VTABs with real-valued continuous intervals in cells
- Therefore, my implementation provides for set-valued nodes representing a disjunction (any value in set is possible)

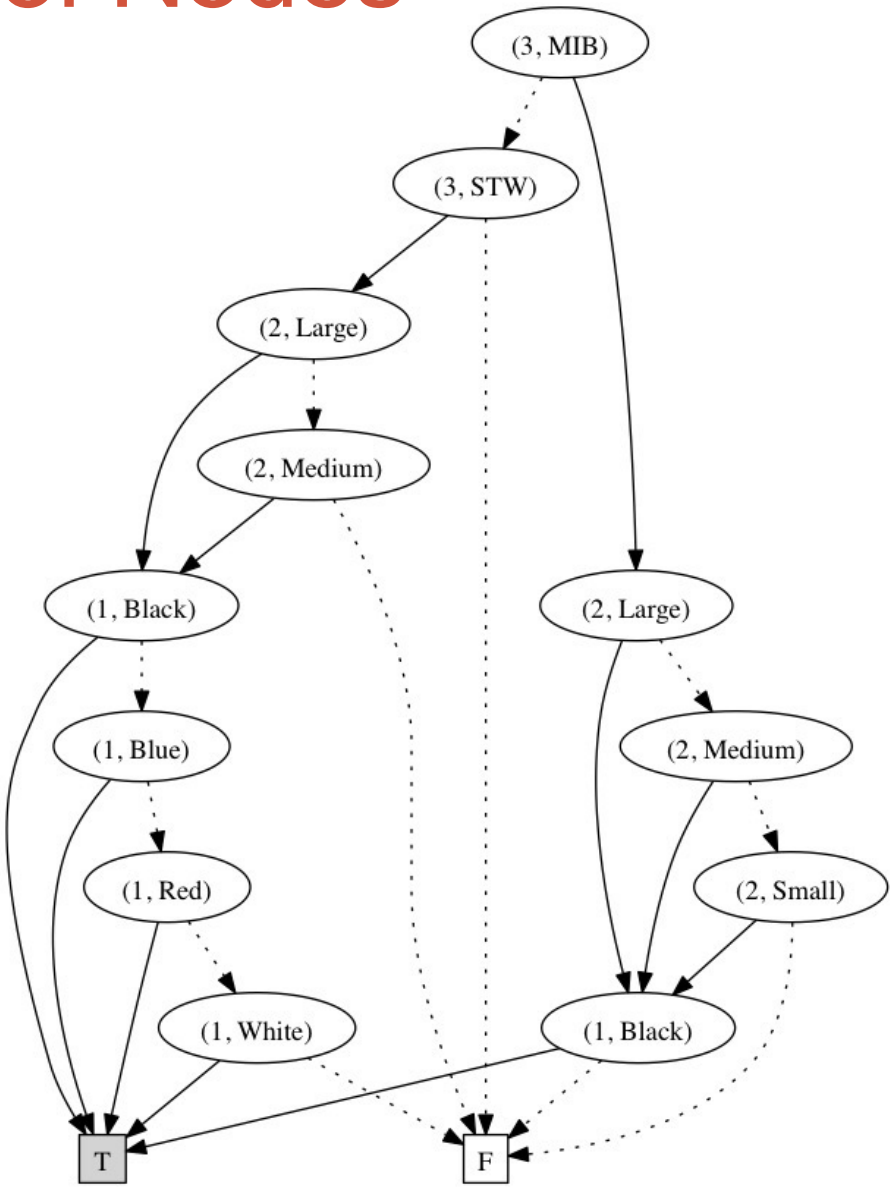
Set-labeled Nodes

- SAP Variant Configurator allows VTABs with real-valued continuous intervals in cells
- Therefore, my implementation provides for set-valued nodes representing a disjunction (any value in set is possible)
  - Includes real-valued continuous intervals

Set-labeled Nodes

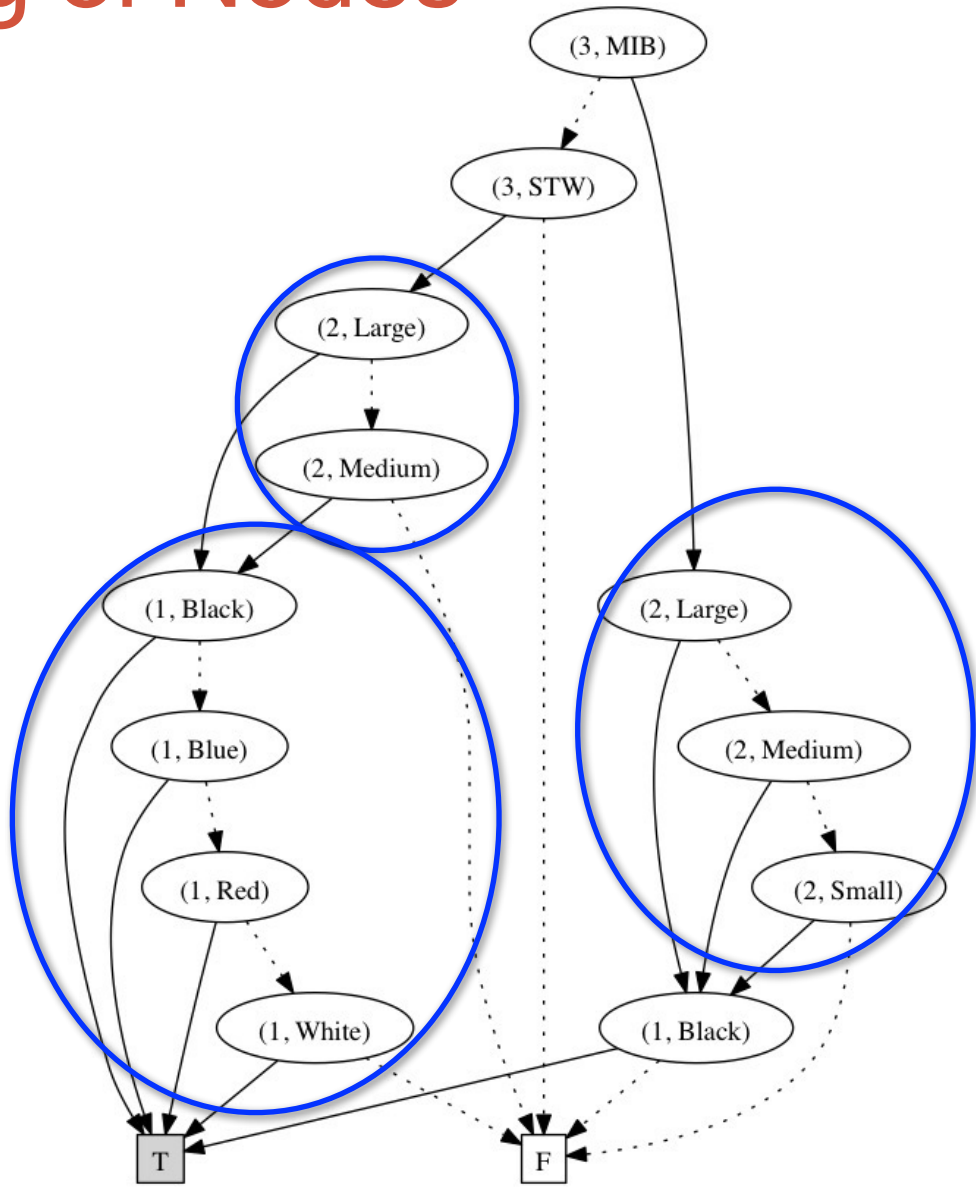
- SAP Variant Configurator allows VTABs with real-valued continuous intervals in cells
- Therefore, my implementation provides for set-valued nodes representing a disjunction (any value in set is possible)
  - Includes real-valued continuous intervals
  - But, in particular, allow nodes labeled with discrete sets

h_2^*: h_2 with Merging of Nodes



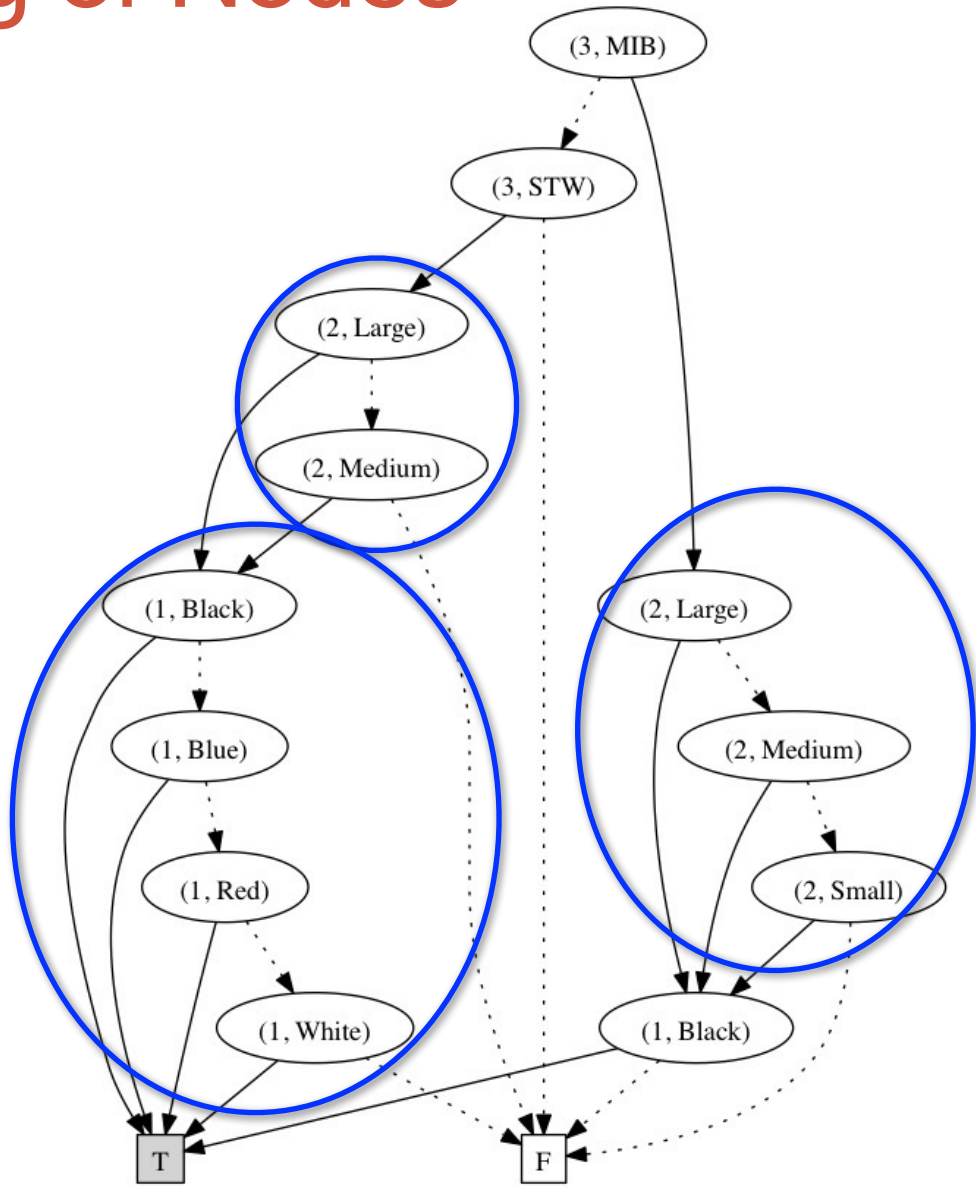
h_2^*: h_2 with Merging of Nodes

- Idea: Merge nodes representing a disjunction into one set-labeled node



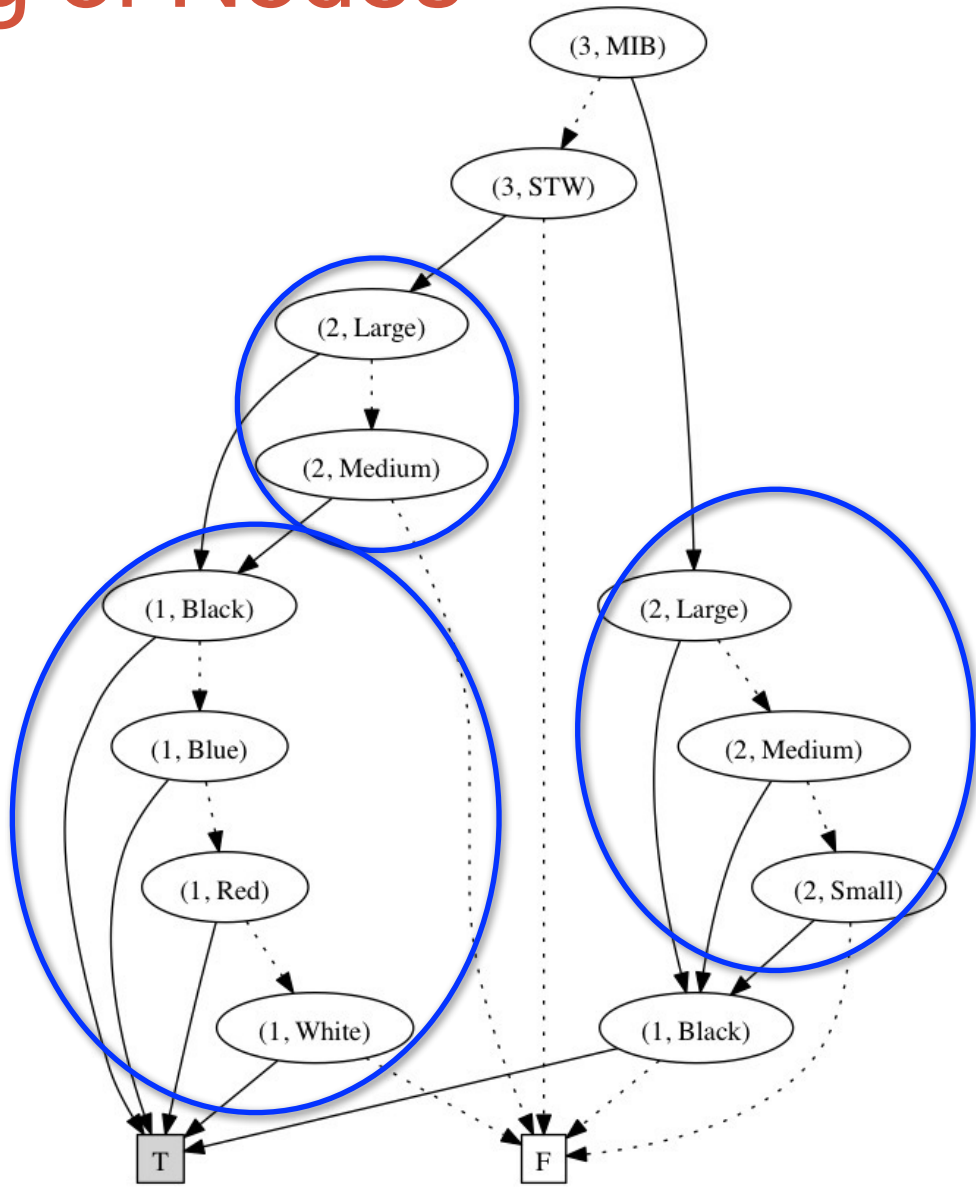
h_2^*: h_2 with Merging of Nodes

- Idea: Merge nodes representing a disjunction into one set-labeled node
- Column order plays a role



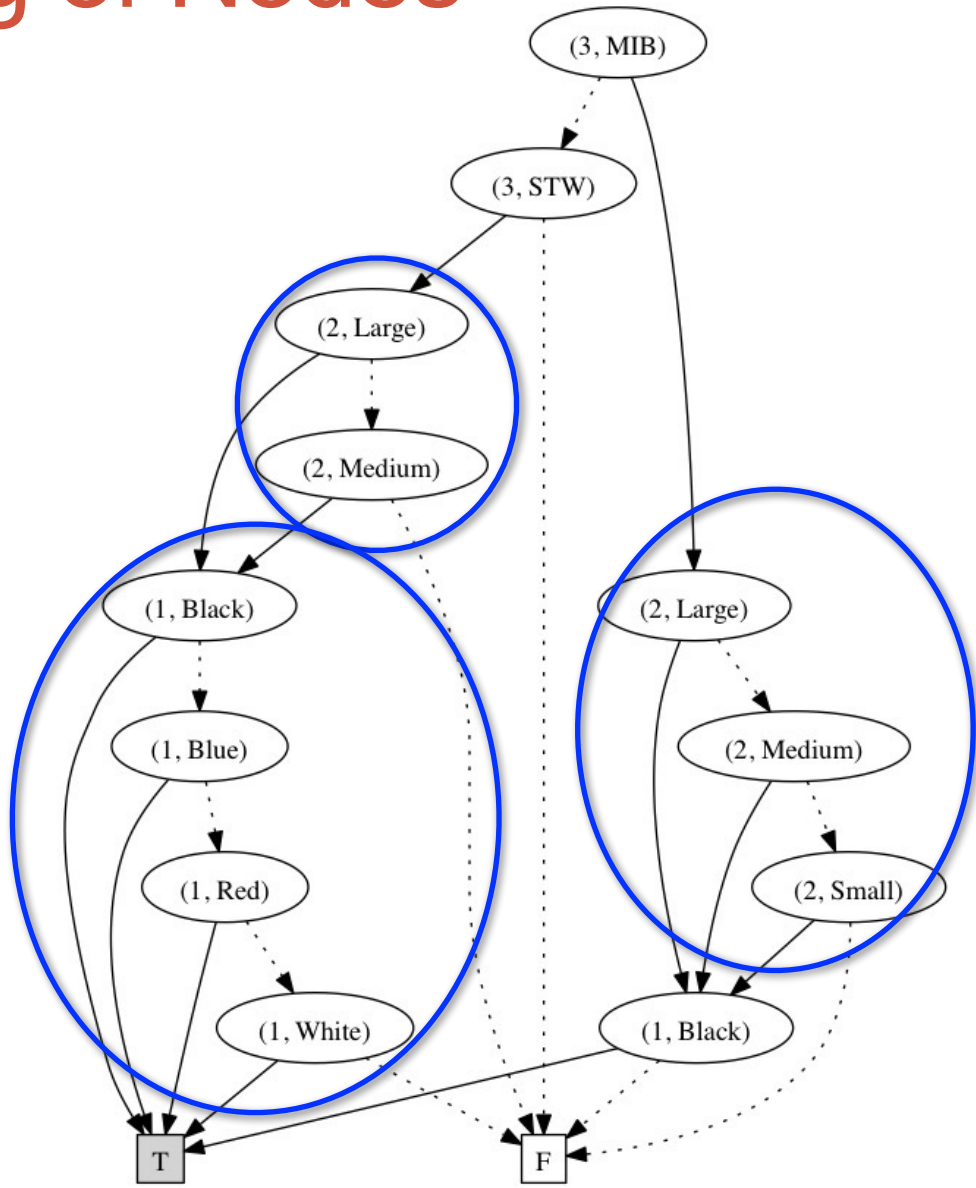
h_2^*: h_2 with Merging of Nodes

- Idea: Merge nodes representing a disjunction into one set-labeled node
- Column order plays a role
- I refer to h_2 with the stated column order heuristic and merging as h_2^*



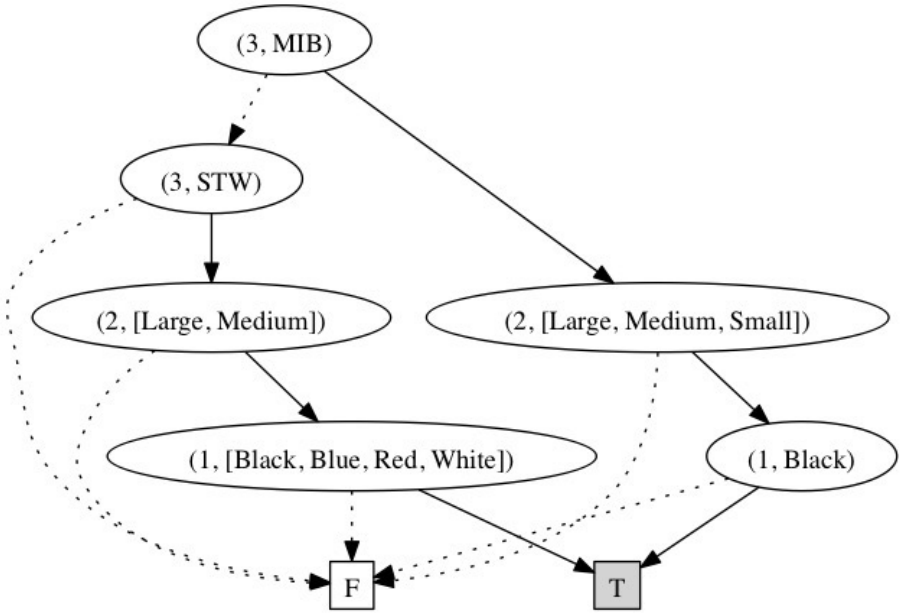
h_2^*: h_2 with Merging of Nodes

- Idea: Merge nodes representing a disjunction into one set-labeled node
- Column order plays a role
- I refer to h_2 with the stated column order heuristic and merging as h_2^*
- The results from querying a VDD may now be c-tuples. A final additional intersection with given a given run-time restriction R is needed



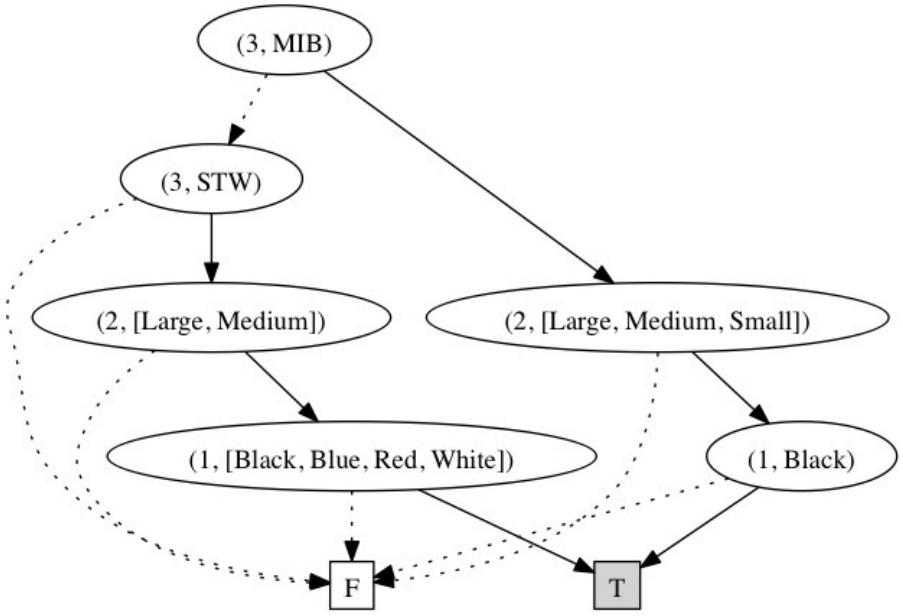
Visual Comparison of h_2^* and MDD for the T-shirt

Visual Comparison of h_2^* and MDD for the T-shirt

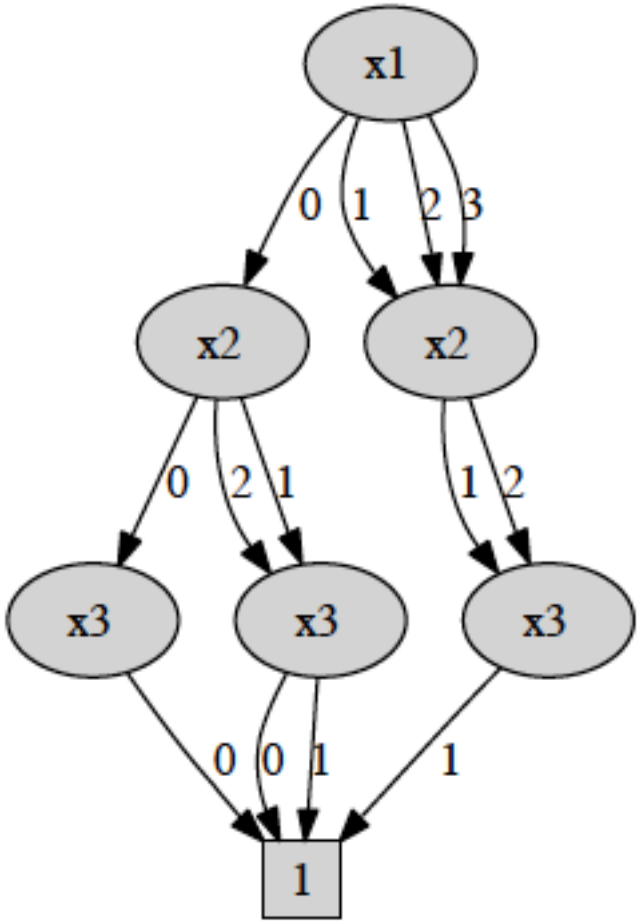


VDD for t-shirt for h_2^*

Visual Comparison of h_2^* and MDD for the T-shirt



VDD for t-shirt for h_2^*



MDD for t-shirt [Andersen et al.]
(same number of nodes)

Statistical Characteristics of Test Bed — 238 “Real” VTABs

Statistical Characteristics of Test Bed — 238 “Real” VTABs

- Arity
  - Maximum: 16
  - Average : 4.29

Statistical Characteristics of Test Bed — 238 “Real” VTABs

- Arity
  - Maximum: 16
  - Average : 4.29
- Size (cells)
  - Maximum: 213,720
  - Average : 1,591

Statistical Characteristics of Test Bed — 238 “Real” VTABs

- Arity
  - Maximum: 16
  - Average : 4.29
- Size (cells)
  - Maximum: 213,720
  - Average : 1,591
- Number of distinct values (*s*)
  - Maximum: 998
  - Average : 79

Statistical Characteristics of Test Bed — 238 “Real” VTABs

- Arity
  - Maximum: 16
  - Average : 4.29
- Size (cells)
  - Maximum: 213,720
  - Average : 1,591
- Number of distinct values (*s*)
  - Maximum: 998
  - Average : 79

Arity/ distinct values (*s*) biggest VTAB: 10/152

Comparison of Compilation Results: h_1 / h_2 / h_2^*

Comparison of Compilation Results: h_1 / h_2 / h_2^*

Compilation of largest VTAB:

- Compilation time (msec): 1,192,988 / 598 / 659
- Number of nodes: 391 / 431 / 145

Comparison of Compilation Results: h_1 / h_2 / h_2^*

Compilation of largest VTAB:

- Compilation time (msec): 1,192,988 / 598 / 659
- Number of nodes: 391 / 431 / 145

Average compilation results:

- Compilation time (msec): 5,476 / 10 / 12
- Number of nodes: 179 / 150 / 84

Comparison of Compilation Results: h_1 / h_2 / h_2^*

Compilation of largest VTAB:

- Compilation time (msec): 1,192,988 / 598 / 659
- Number of nodes: 391 / 431 / 145

Average compilation results:

- Compilation time (msec): 5,476 / 10 / 12
- Number of nodes: 179 / 150 / 84

Summary performance (number of nodes)

Comparison of Compilation Results: h_1 / h_2 / h_2^*

Compilation of largest VTAB:

- Compilation time (msec): 1,192,988 / 598 / 659
- Number of nodes: 391 / 431 / 145

Average compilation results:

- Compilation time (msec): 5,476 / 10 / 12
- Number of nodes: 179 / 150 / 84

Summary performance (number of nodes)

- h_2 outperformed h_1 for: 143 VTABS

Comparison of Compilation Results: h_1 / h_2 / h_2^*

Compilation of largest VTAB:

- Compilation time (msec): 1,192,988 / 598 / 659
- Number of nodes: 391 / 431 / 145

Average compilation results:

- Compilation time (msec): 5,476 / 10 / 12
- Number of nodes: 179 / 150 / 84

Summary performance (number of nodes)

- h_2 outperformed h_1 for: 143 VTABS
- h_2 equal to h_1 for: 71 VTABS

Comparison of Compilation Results: h_1 / h_2 / h_2^*

Compilation of largest VTAB:

- Compilation time (msec): 1,192,988 / 598 / 659
- Number of nodes: 391 / 431 / 145

Average compilation results:

- Compilation time (msec): 5,476 / 10 / 12
- Number of nodes: 179 / 150 / 84

Summary performance (number of nodes)

- h_2 outperformed h_1 for: 143 VTABS
- h_2 equal to h_1 for: 71 VTABS
- h_2 beaten by h_1 for: 24 VTABS

Conclusion

Conclusion

- Summary:
  - h_2/h_2^* is a very simple/ very fast compilation technique for VTABs
  - Number of nodes seems comparable to that of the examples reported for MDDs [Andersen et al.]and table compression [Katsirelos/Walsh]
  - Implementation (Java) supports real-valued continuous intervals

Conclusion

- Summary:
  - h_2/h_2^* is a very simple/ very fast compilation technique for VTABs
  - Number of nodes seems comparable to that of the examples reported for MDDs [Andersen et al.]and table compression [Katsirelos/Walsh]
  - Implementation (Java) supports real-valued continuous intervals
- Results
  - A VDD allows fast queries (proportional to their size – like a BDD)
  - Direct queries for filtering out unsupported values are easily implemented
  - Simple database queries are subsumed by these queries
  - Iteration and counting are implemented as well

Conclusion

- Summary:
  - h_2/h_2^* is a very simple/ very fast compilation technique for VTABs
  - Number of nodes seems comparable to that of the examples reported for MDDs [Andersen et al.]and table compression [Katsirelos/Walsh]
  - Implementation (Java) supports real-valued continuous intervals
- Results
  - A VDD allows fast queries (proportional to their size – like a BDD)
  - Direct queries for filtering out unsupported values are easily implemented
  - Simple database queries are subsumed by these queries
  - Iteration and counting are implemented as well
- Outlook
  - Formal analysis and application to wider test bed is future work
  - Objective run-time measurements still missing

Thank You