Using Bit Operations for Fast Evaluation of T-norms

Michal Burda

Institute for Research and Applications of Fuzzy Modeling
National Supercomputing Center IT4Innovations – Division University of Ostrava
30. dubna 22, 701 03 Ostrava, Czech Republic

michal.burda@osu.cz

January 30, 2014 (FSTA, Liptovský Ján, Slovakia)

IT4Innovations
national01$#&0
supercomputing
center@#01%101

EUROPEAN UNION
EUROPEAN REGIONAL DEVELOPMENT FUND
INVESTING IN YOUR FUTURE

2007–13
OP Research and
Development for Innovation

Outline

Definitions

- **Fuzzy attribute** A is defined by a membership function $A : U \to [0,1]$, where U is a universe.
- $A(u)$ denotes membership of $u \in U$ in A.

- **T-norm** $\otimes$ is a binary function that generalizes logical conjunction:
  - minimum t-norm: $\otimes_{\min}(\alpha, \beta) = \min(\alpha, \beta)$;
  - Łukasiewicz t-norm: $\otimes_{\text{Łuk}}(\alpha, \beta) = \max(0,\ \alpha + \beta - 1)$.

Fuzzy Association Rules

middle age	big city	...	high income
0.1	0.5	⋯	0.5
0.3	0.1	⋯	0.2
1.0	0.9	⋯	0.8
⋮	⋮	⋱	⋮
0.2	0.3	⋯	1.0

- data in the form of fuzzy attributes;
- graded memberships from interval $[0, 1]$;
- rules in the form:

$$A \Rightarrow B,$$
$$\text{middle age} \wedge \text{big city} \Rightarrow \text{high income.}$$

Rule intensity measures:

1. $support(A_1 \wedge \ldots \wedge A_n \Rightarrow B) = \sum_{\forall u \in U} \big(A_1(u) \otimes \ldots \otimes A_n(u) \otimes B\big);$
2. $confidence(A \Rightarrow B) = \frac{support(A \wedge B)}{support(A)}.$

Algorithms for Association Rules

- association rules mining = searching for all rules with $support$ and $confidence$ above some given minimum thresholds;
- generating thousands of variants of rules and testing rule intensity measures;
- various heuristics based e.g. on minimum support threshold etc. (off-topic);
- data: hundreds of columns, thousands (or millions) of rows;
- bottleneck: evaluation of rule $support$ – need to traverse all rows and compute conjunctions.

Objective

middle age	big city	...	high income
0.1	0.5	⋯	0.5
0.3	0.1	⋯	0.2
0.9	0.8	⋯	0.7
1.0	0.9	⋯	0.8
⋮	⋮	⋱	⋮
0.2	0.3	⋯	1.0

1. Given "column vectors" A, B of membership degrees, how to perform fast evaluation of vectorized t-norm $C(u) = A(u) \otimes B(u)$, for each u?
2. How to perform fast evaluation of $support(C) = \sum_{\forall u} C(u)$?

Note – limited instruction set in CPU:

- arithmetic operations ($+$, $-$, $*$, $/$)
- bitwise operations ($\&$, |, **!**, $\ll$, $\gg$)
- ... but no vectorized instructions (such as SSE)

Naive Implementation

- A, B are vectors of floating point numbers from $[0, 1]$;
- $A[x]$ denotes x-th value of vector A.

function tnorm$_{naive}(A, B)$
  $R \leftarrow$ new array of size $|A|$
  for $x \in \{1, 2, \ldots, |A|\}$ **do**
    $R[x] \leftarrow A[x] \otimes B[x]$
  end for
  return R
end function

function support$_{naive}(A)$
  return $\sum_{x=1}^{|A|} A[x]$
end function

Idea of Bitwise Implementation

- "Vectorized approach"
- Encode multiple column values into a single integer.
- By doing operations on that integers, compute multiple t-norms.

Bitwise Implementation

- 7 bits per membership degree (1 bit for overflow)
- 8 membership degrees encoded into a single 64bit integer
- $[0, 1]$ values mapped into $\{0, 1, \ldots, 127\}$

Example:

original	encoded binary	encoded decimal	
0	00000000	0	
0.0079	00000001	1	
0.0157	00000010	2	
0.5039	01000000	64	
0.9921	01111110	126	
1	01111111	127	$= max$

Data Format

Arrangement of chunks inside of a single 64bit integer:

bit index:	56–64	...	9-16	1–8
data:	$chunk_8$	...	$chunk_2$	$chunk_1$

The meaning of bits inside of a chunk of 8 bits:

bit index:	8	7	6	5	4	3	2	1
data:	$sign/overflow$	b_7	b_6	b_5	b_4	b_3	b_2	b_1

Bitwise Implementation of Łukasiewicz t-norm

Theorem

Let p, q, r, s *be integer numbers encoded in* $precision$ *bits,*

$$\begin{aligned} p, q &\in \{0, 1, \dots, max\}, \\ s &= \begin{cases} 011\dots1 & \text{if } p + q > max, \\ 000\dots0 & \text{otherwise,} \end{cases} \\ r &= (p + q + 1) \mathbin{\&} s, \end{aligned}$$

and

$$\alpha = \frac{p}{max}, \qquad \beta = \frac{q}{max}.$$

Then

$$\otimes_{\text{Łuk}}(\alpha, \beta) = \max(0, \alpha + \beta - 1) = \frac{r}{max}.$$

Bitwise Evaluation of Łukasiewicz t-norm

$m_1 =$	00000001	...	00000001	00000001
$m_8 =$	10000000	...	10000000	10000000
$p =$	00001111 (15)	...	01110001 (113)	01111111 (127)
$q =$	00100001 (33)	...	01010110 (86)	01111111 (127)
$t =$	00000000 (0)	...	00000000 (0)	00000000 (0)
$s =$	00000000	...	00000000	00000000
$r =$	00000000 (0)	...	00000000 (0)	00000000 (0)

Operations performed: **0**
T-norms computed: **0**

$$\otimes_{\text{Łuk}}(p, q) = \max(0,\ p + q - 127)$$

1. Initialize p, q.
2. $t := p + q$.
3. $s := t \,\&\, m_8$.
4. $s := s \mid (s \gg 1)$
5. $s := s \mid (s \gg 2)$
6. $s := s \mid (s \gg 3)$
7. $s := s \gg 1$
8. $r := (t + m_1) \,\&\, s$

Bitwise Evaluation of Łukasiewicz t-norm

$m_1 =$	00000001	...	00000001	00000001
$m_8 =$	10000000	...	10000000	10000000
$p =$	00001111 (15)	...	01110001 (113)	01111111 (127)
$q =$	00100001 (33)	...	01010110 (86)	01111111 (127)
$t =$	00110000 (48)	...	11000111 (199)	11111110 (254)
$s =$	00000000	...	00000000	00000000
$r =$	00000000 (0)	...	00000000 (0)	00000000 (0)

Operations performed: **1**
T-norms computed: **0**

$$\otimes_{\text{Łuk}}(p, q) = \max(0,\ p + q - 127)$$

1. Initialize p, q.
2. $t := p + q$.
3. $s := t \,\&\, m_8$.
4. $s := s \,|\, (s \gg 1)$
5. $s := s \,|\, (s \gg 2)$
6. $s := s \,|\, (s \gg 3)$
7. $s := s \gg 1$
8. $r := (t + m_1) \,\&\, s$

Bitwise Evaluation of Łukasiewicz t-norm

$m_1 =$	00000001		...	00000001		00000001
$m_8 =$	10000000		...	10000000		10000000
$p =$	00001111 (15)		...	01110001 (113)		01111111 (127)
$q =$	00100001 (33)		...	01010110 (86)		01111111 (127)
$t =$	00110000 (48)		...	11000111 (199)		11111110 (254)
$s =$	00000000		...	10000000		10000000
$r =$	00000000 (0)		...	00000000 (0)		00000000 (0)

Operations performed: **2**
T-norms computed: **0**

$$\otimes_{\text{Łuk}}(p, q) = \max(0,\ p + q - 127)$$

1. Initialize p, q.
2. $t := p + q$.
3. $s := t \,\&\, m_8$.
4. $s := s \mid (s \gg 1)$
5. $s := s \mid (s \gg 2)$
6. $s := s \mid (s \gg 3)$
7. $s := s \gg 1$
8. $r := (t + m_1) \,\&\, s$

Bitwise Evaluation of Łukasiewicz t-norm

$m_1 =$	00000001	...	00000001	00000001
$m_8 =$	10000000	...	10000000	10000000
$p =$	00001111 (15)	...	01110001 (113)	01111111 (127)
$q =$	00100001 (33)	...	01010110 (86)	01111111 (127)
$t =$	00110000 (48)	...	11000111 (199)	11111110 (254)
$s =$	00000000	...	11000000	11000000
$r =$	00000000 (0)	...	00000000 (0)	00000000 (0)

Operations performed: **4**
T-norms computed: **0**

$$\otimes_{\text{Łuk}}(p, q) = \max(0,\ p + q - 127)$$

1. Initialize p, q.
2. $t := p + q$.
3. $s := t \,\&\, m_8$.
4. $s := s \mid (s \gg 1)$
5. $s := s \mid (s \gg 2)$
6. $s := s \mid (s \gg 3)$
7. $s := s \gg 1$
8. $r := (t + m_1) \,\&\, s$

Bitwise Evaluation of Łukasiewicz t-norm

$m_1 =$	00000001	...	00000001	00000001
$m_8 =$	10000000	...	10000000	10000000
$p =$	00001111 (15)	...	01110001 (113)	01111111 (127)
$q =$	00100001 (33)	...	01010110 (86)	01111111 (127)
$t =$	00110000 (48)	...	11000111 (199)	11111110 (254)
$s =$	00000000	...	11110000	11110000
$r =$	00000000 (0)	...	00000000 (0)	00000000 (0)

Operations performed: **6**

T-norms computed: **0**

$$\otimes_{\text{Łuk}}(p,q) = \max(0,\ p+q-127)$$

1. Initialize p, q.
2. $t := p + q$.
3. $s := t \,\&\, m_8$.
4. $s := s \,|\, (s \gg 1)$
5. $s := s \,|\, (s \gg 2)$
6. $s := s \,|\, (s \gg 3)$
7. $s := s \gg 1$
8. $r := (t + m_1) \,\&\, s$

Bitwise Evaluation of Łukasiewicz t-norm

$m_1 =$	00000001	...	00000001	00000001
$m_8 =$	10000000	...	10000000	10000000
$p =$	00001111 (15)	...	01110001 (113)	01111111 (127)
$q =$	00100001 (33)	...	01010110 (86)	01111111 (127)
$t =$	00110000 (48)	...	11000111 (199)	11111110 (254)
$s =$	00000000	...	11111110	11111110
$r =$	00000000 (0)	...	00000000 (0)	00000000 (0)

Operations performed: **8**
T-norms computed: **0**

$$\otimes_{\text{Łuk}}(p, q) = \max(0,\ p + q - 127)$$

1. Initialize p, q.
2. $t := p + q$.
3. $s := t \,\&\, m_8$.
4. $s := s \,|\, (s \gg 1)$
5. $s := s \,|\, (s \gg 2)$
6. $s := s \,|\, (s \gg 3)$
7. $s := s \gg 1$
8. $r := (t + m_1) \,\&\, s$

Bitwise Evaluation of Łukasiewicz t-norm

$m_1 =$	00000001	...	00000001	00000001
$m_8 =$	10000000	...	10000000	10000000
$p =$	00001111 (15)	...	01110001 (113)	01111111 (127)
$q =$	00100001 (33)	...	01010110 (86)	01111111 (127)
$t =$	00110000 (48)	...	11000111 (199)	11111110 (254)
$s =$	00000000	...	01111111	01111111
$r =$	00000000 (0)	...	00000000 (0)	00000000 (0)

Operations performed: **9**
T-norms computed: **0**

$$\otimes_{\text{Łuk}}(p, q) = \max(0,\ p + q - 127)$$

1. Initialize p, q.
2. $t := p + q$.
3. $s := t \mathbin{\&} m_8$.
4. $s := s \mid (s \gg 1)$
5. $s := s \mid (s \gg 2)$
6. $s := s \mid (s \gg 3)$
7. $s := s \gg 1$
8. $r := (t + m_1) \mathbin{\&} s$

Bitwise Evaluation of Łukasiewicz t-norm

$m_1 =$	00000001	...	00000001	00000001
$m_8 =$	10000000	...	10000000	10000000
$p =$	00001111 (15)	...	01110001 (113)	01111111 (127)
$q =$	00100001 (33)	...	01010110 (86)	01111111 (127)
$t =$	00110000 (48)	...	11000111 (199)	11111110 (254)
$s =$	00000000	...	01111111	01111111
$r =$	00000000 (0)	...	01001000 (72)	01111111 (127)

Operations performed: **11**
T-norms computed: **8**

$$\otimes_{\text{Łuk}}(p, q) = \max(0,\ p + q - 127)$$

1. Initialize p, q.
2. $t := p + q$.
3. $s := t \,\&\, m_8$.
4. $s := s \,|\, (s \gg 1)$
5. $s := s \,|\, (s \gg 2)$
6. $s := s \,|\, (s \gg 3)$
7. $s := s \gg 1$
8. $r := (t + m_1) \,\&\, s$

Computation of Support

$$support(C) = \sum_{\forall u} C(u)$$

- compute sum of values stored in chunks of integers

Naive approach:

- unpack values from chunks (one-by-one),
- then sum up.

Optimized approach:

1. sum multiple odd and even chunks separately (a single "+" operation sums 4 pairs of chunks);
2. unpack partial sums to obtain a total sum.

Optimized Computation of Support

data:

...	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

odd:

...	17	–	15	–	13	–	11	–	9	–	7	–	5	–	3	–	1

odd sum:

sum_7	sum_5	sum_3	sum_1

even:

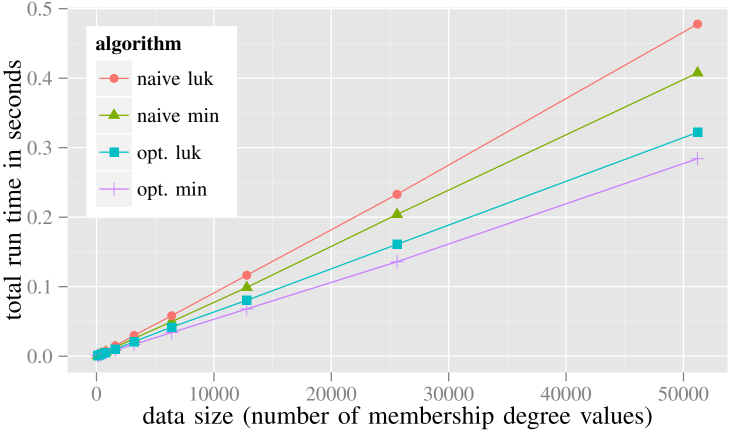
...	–	16	–	14	–	12	–	10	–	8	–	6	–	4	–	2	–

even sum:

sum_8	sum_6	sum_4	sum_2

- sum odd and even chunks separately;
- single "+" operation sums 4 pairs of chunks;
- perform only so many summations for not to overflow;
- after that: unpack partial odd and even sums and sum them up.

Performance Analysis



Approx. 30 % less computational time

Conclusion

Bitwise approach is:

- suitable for platforms without vectorized instructions on CPU
- faster than naive approach (both are in $O(kn)$, bitwise approach has lower k) – approx. 30 % less time is needed;
- lower memory consumption;
- available for Łukasiewicz and Minimum t-norms;
- for product t-norm, the bitwise approach is still slower than naive approach.

Thank you. . . Questions?