Overview of Dual Miner

Paul Nalos

November 16, 2004

Problem Statement

"Find all frequent itemsets whose total price is at least $50."

- Constraining mining improves its speed and usefulness.
- Many practical constraints can be expressed as a conjunction of monotone and antimonotone predicates.
- Other constraints can be approximated this way.

Dual Miner finds frequent itemsets by leveraging monotone and antimonotone constraints at the same time.

Related Work

- Dual Miner is brought to you by the creators of MAFIA.

Related Work

- Dual Miner is brought to you by the creators of MAFIA.
  - They like MAFIA.

Related Work

- Dual Miner is brought to you by the creators of MAFIA.
    - They like MAFIA.
    - Dual Miner is traversal strategy agnostic.

Related Work

- Dual Miner is brought to you by the creators of MAFIA.
  - They like MAFIA.
  - Dual Miner is traversal strategy agnostic.
- Other ways to solve the problem:
  - Run existing algorithm twice and intersect.
  - Run existing algorithm and post-process.
  - Melish's Algorithm

All of these require two distinct phases.

Related Work

- Dual Miner is brought to you by the creators of MAFIA.
  - They like MAFIA.
  - Dual Miner is traversal strategy agnostic.
- Other ways to solve the problem:
  - Run existing algorithm twice and intersect.
  - Run existing algorithm and post-process.
  - Melish's Algorithm

  All of these require two distinct phases.
- Other types of constraints:
  - Succinct
  - Convertible

Dual Miner has to offer...

- First algorithm to leverage P() and Q() simultaneously
- Extreme flexibility
- Non-trivial optimizations
- New issues
- Nice summary of analytical properties

Definition

- P() is a conjunction of antimonotone predicates.
- Q() is a conjunction of monotone predicates.

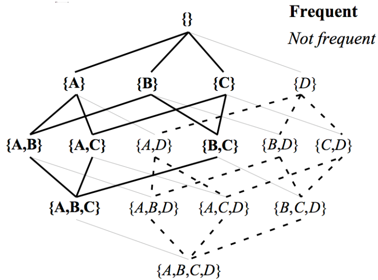
Key Ideas

- Items have attributes and values
- Monotone and antimonotone predicates
  e.g. $\text{sum}(\text{price}(X)) > 30$ or $10 < \text{support}(X) < 100$
- Join predicates of same type
- Approximate other types of constraints
- Different predicates have different cost
  e.g. support vs. sum

More Key Ideas

- Trimming values near top or bottom removes many nodes
- Duality

Find all frequent itemsets

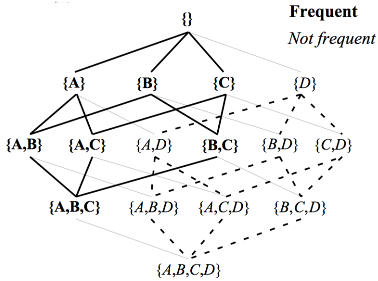


D is OUT because it is not frequent.

More Key Ideas

- Trimming values near top or bottom removes many nodes
- Duality

Find all infrequent itemsets



D is IN because ∼D (ABC) is frequent.

Subalgebras

- don't care about support – look for MFI
- ... but all subsets of MFI may not satisfy Q()

How can we represent portions of the result space?

Subalgebras

- don't care about support – look for MFI
- ... but all subsets of MFI may not satisfy Q()

How can we represent portions of the result space?

Any set of itemsets closed under $\cup$ and $\cap$ can be expressed as a subalgebra.

- A subalgebra consists of a bottom set and a top set.
- All itemsets in a subalgebra contain all of the items in the bottom set,
- and only items from the top set.
- All members of a good subalgebra satisfy P() and Q().

Subalgebras Con't

For example, the subalgebra ({A}, {ABC}) contains the elements:

- A
- AB
- AC
- ABC

$\max(X.price) < 4 \wedge \min(X.price) < 2$

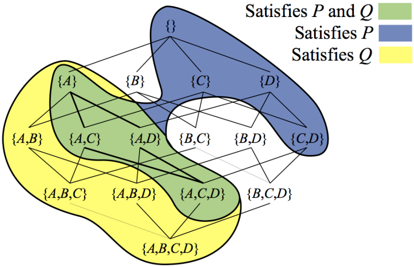
Item	Cost
A	1
B	4
C	3
D	2

- $P(X) = \max(X.price) < 4$
- $Q(X) = \min(X.price) < 2$
- P(B) is false, therefore B is OUT
- $Q(\sim A) = Q(BCD)$ is false, therefore A is IN

max(X.price) $< 4 \wedge$ min(X.price) < 2

- P(B) is false, therefore B is OUT
- Q($\sim$A) = Q(BCD) is false, therefore A is IN

This leads to the subalgebra ({A}, {ACD}), which satisfies P() and Q().



Towards the Basic Algorithm 1/2

As usual, we will traverse nodes in a tree.

- Each node contains IN, OUT, and CHILD sets, which correspond to the subalgebra (IN, ∼OUT).
- This is a good subalgebra if P(∼OUT) ⋀ Q(IN).

Towards the Basic Algorithm 2/2

Dual Miner

- Start with the root node; it is undetermined.
- Repeatedly pick an undetermined node:
  - Optionally, move children to OUT, if P(IN $\cup$ child) fails or to IN, if Q($\sim$(OUT $\cup$ child)) fails
  - Optionally, check if node is a good subalgebra.
  - Pick one child element to split on, and create two child nodes.
  - The node is now determined.

$\text{support}(X) \geq 1 \wedge \text{total\_price}(X) > 50$

Item	Cost
A	26
B	26
C	1
D	1
E	100

Transactions
ABCD
E

Root Node: ({}, {ABCDE}, {})

- P(X) true for all elements X
- Q(∼X) true for all elements X
- ({}, {ABCDE}) is not a good subalgebra
- ... therefore pick a child (say E) to split on

$\text{support}(X) \geq 1 \bigwedge \text{total\_price}(X) > 50$

Item	Cost
A	26
B	26
C	1
D	1
E	100

Transactions
ABCD
E

Undetermined Nodes:

- β: ({E}, {ABCD}, {})
- γ: ({}, {ABCD}, {E})

$\text{support}(X) \geq 1 \bigwedge \text{total\_price}(X) > 50$

Item	Cost
A	26
B	26
C	1
D	1
E	100

Transactions
ABCD
E

Current Node: β: ({E}, {ABCD}, {})

- P(EA) is false, so A is OUT
- β becomes ({E}, {BCD}, {A})

$\text{support}(X) \geq 1 \bigwedge \text{total\_price}(X) > 50$

Item	Cost
A	26
B	26
C	1
D	1
E	100

Transactions
ABCD
E

Current Node: β: ({E}, {ABCD}, {})

- P(EA) is false, so A is OUT
- β becomes ({E}, {BCD}, {A})
- P(EB), P(EC), P(ED) are all false too
- β becomes ({E}, {}, {ABCD})

$\text{support}(X) \geq 1 \bigwedge \text{total\_price}(X) > 50$

Item	Cost
A	26
B	26
C	1
D	1
E	100

Transactions
ABCD
E

Current Node: β: ({E}, {ABCD}, {})

- P(EA) is false, so A is OUT
- β becomes ({E}, {BCD}, {A})
- P(EB), P(EC), P(ED) are all false too
- β becomes ({E}, {}, {ABCD})
- ({E}, {E}) is a good subalgebra

$\text{support}(X) \geq 1 \bigwedge \text{total\_price}(X) > 50$

Item	Cost
A	26
B	26
C	1
D	1
E	100

Transactions
ABCD
E

Current Node: γ: ({}, {ABCD}, {E})

- Q(∼(EA)) is false, so A is IN
- γ becomes ({A}, {BCD}, {E})

$\text{support}(X) \geq 1 \bigwedge \text{total\_price}(X) > 50$

Item	Cost
A	26
B	26
C	1
D	1
E	100

Transactions
ABCD
E

Current Node: γ: ({}, {ABCD}, {E})

- Q(∼(EA)) is false, so A is IN
- γ becomes ({A}, {BCD}, {E})
- Q(∼(EB)) is false, so B is IN
- γ becomes ({AB}, {CD}, {E})

$\text{support}(X) \geq 1 \wedge \text{total\_price}(X) > 50$

Item	Cost
A	26
B	26
C	1
D	1
E	100

Transactions
ABCD
E

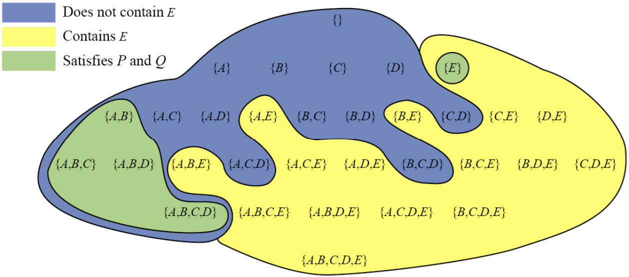
Current Node: γ: ({}, {ABCD}, {E})

- Q(∼(EA)) is false, so A is IN
- γ becomes ({A}, {BCD}, {E})
- Q(∼(EB)) is false, so B is IN
- γ becomes ({AB}, {CD}, {E})
- ({AB}, {ABCD}) is a good subalgebra

$\text{support}(X) \geq 1 \wedge \text{total\_price}(X) > 50$

In Summary, the good subalgebras were:

- ({E}, {E})
- ({AB}, {ABCD})



Finishing Touches

- Don't store IN, OUT, and CHILD;
  just store new_in and new_out for each node.
- Interleave pruning with P() and Q();
  each creates opportunities for the other.

Heuristics

Dual Miner is flexible; it can be tuned for the problem at hand:

- traversal strategy
- pruning order heuristics
- stop heuristics
- choice order heuristics
- control heuristics

Heuristics

Dual Miner is flexible; it can be tuned for the problem at hand:

- traversal strategy
- pruning order heuristics
- stop heuristics
- choice order heuristics
- control heuristics
  - don't prune with P() unless IN has changed
  - don't prune with Q() unless OUT has changed

Leveraging the Underlying Algorithm

Goal: Minimize the cost of executing P() and Q()

- Q() is often much cheaper
- existing algorithms minimize calls to P() in clever ways – reuse them!
- dual situation may occur as well

Extend MAFIA HUTMFI...

- partial list of maximum itemsets which satisfy P()
- partial list of minimum itemsets which satisfy Q()

Dual HUT

Testing P(∼OUT) may be expensive.

Dual HUT

Testing P(∼OUT) may be expensive.

Alternative:

- Examine a sequence of nodes where new children are speculatively added to IN, until you reach a leaf.
- The oldest ancestor in the chain that satisfies Q() is a good subalgebra.
- Don't evaluate any of its children.

Dual HUT

Testing P(∼OUT) may be expensive.

Alternative:

- Examine a sequence of nodes where new children are speculatively added to IN, until you reach a leaf.
- The oldest ancestor in the chain that satisfies Q() is a good subalgebra.
- Don't evaluate any of its children.

Also:

- We don't need to evaluate P() for any nodes below the top of the chain, even if that node doesn't satisfy Q().
- This applies in the dual case also.

Seek out *complete left and right chains.*

Subalgebra Fragmentation

Problem

If Dual Miner splits on the wrong child nodes, it can split good subalgebras.

Mitigation

- Heuristics can help.
- Keep track of good subalgebras and merge them on the fly.

Approximations

Problem

Dual Miner only handles monotone and antimonotone constraints.

Mitigation

A strategy for approximating mean-like functions is proposed.

- average(X) $<$ constant

Theoretical Evaluation

Summary:

- Same (weak) upper bound on Dual Miner / Apriori and Dual Miner / MAFIA
- P() more selective than Q() → CONVERTIBLE wins

COVERTIBLE

Run Apriori and post-process, but don't test Q() on a superset of something that already passed.

Emprical Evaluation

- Assume P() costs 100x Q()
- Test vs. synthetic data
- Observation: Other algorithms operate in two phases
- Competitors
  - CONVERTIBLE
  - MAFIA + free second phase
- Champions
  - Dual Miner
  - Dual Miner with good choice order heuristic

Dual Miner wins when Q() is sufficiently selective.

Thank you!