Merging DBMs Efficiently

Alexandre David
Aalborg University
Plan

- Framework
  - Timed Automata
  - DBMs & Federations
  - Why Merging DBMs?

- Merging DBMs
  - The Problem
  - The Different Algorithms
  - Experiments
Warming Up:
Timed Automata in a Nutshell

init → low: push? x:=0
low → high: push? x<=5
high → init: push?

User pushes to transition states:
- low
- high

Lamp transitions:
- push?
- push!

States:
- init
- low
- high
What is it all about?

- Difference Bound Matrix: Data structure for representing clock constraints, i.e., zones.
- DBMs represent \textit{convex} zones. Note: canonical form.
- Some operations (subtractions) may result in non-convex zones, i.e., DBMs must be \textit{split}.
- \textit{Federations}: unions of zones (DBMs).
Example of a DBM

\[ \begin{align*}
    x_0 - x_0 & \leq 0 \\
    x_1 - x_0 & \leq 1 \\
    x_2 - x_0 & \leq 5 \\
    x_0 - x_1 & \leq 2 \\
    x_1 - x_1 & \leq 0 \\
    x_2 - x_1 & \leq 1 \\
    x_0 - x_2 & \leq 1 \\
    x_1 - x_2 & \leq 3 \\
    x_2 - x_2 & \leq 0 \\
\end{align*} \]

\[ x_i - x_j \leq c_{ij} \]
Example of a Federation

<table>
<thead>
<tr>
<th>$x_0 - x_0 \leq 0$</th>
<th>$x_0 - x_1 \leq -2$</th>
<th>$x_0 - x_2 \leq -1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 - x_0 \leq 6$</td>
<td>$x_1 - x_1 \leq 0$</td>
<td>$x_1 - x_2 \leq 3$</td>
</tr>
<tr>
<td>$x_2 - x_0 \leq 5$</td>
<td>$x_2 - x_1 \leq 1$</td>
<td>$x_2 - x_2 \leq 0$</td>
</tr>
</tbody>
</table>

+matrix of the second DBM

Disjoint
Cannot be simplified

$x_i - x_j \leq c_{ij}$
Example of a Federation

\[
x_0 - x_0 \leq 0 \quad x_0 - x_1 \leq -2 \quad x_0 - x_2 \leq -1
\]

\[
x_1 - x_0 \leq 6 \quad x_1 - x_1 \leq 0 \quad x_1 - x_2 \leq 3
\]

\[
x_2 - x_0 \leq 5 \quad x_2 - x_1 \leq 1 \quad x_2 - x_2 \leq 0
\]

\[
x_i - x_j \leq c_{ij}
\]

+matrix of the second DBM

Can be simplified
Example of a Federation

<table>
<thead>
<tr>
<th>x₀-x₀ &lt;= 0</th>
<th>x₀-x₁ &lt;= -2</th>
<th>x₀-x₂ &lt;= -1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁-x₀ &lt;= 6</td>
<td>x₁-x₁ &lt;= 0</td>
<td>x₁-x₂ &lt;= 3</td>
</tr>
<tr>
<td>x₂-x₀ &lt;= 5</td>
<td>x₂-x₁ &lt;= 1</td>
<td>x₂-x₂ &lt;= 0</td>
</tr>
</tbody>
</table>

\[ xᵢ - xⱼ \leq cᵢⱼ \]

+matrix of the second DBM

Cannot be simplified
Why Merging DBMs?

- State explosion: “Split” states give “split” successors etc...

- Even if it is costly (see algorithms), it does work. Justified by operations that make it possible.

- Note: We have not used alternative representations yet on our experiments, e.g., CDDs. We do our best with what we have, i.e., federations.
The Problem

- Given a Federation, is it possible to simplify it?
  - Remove included DBMs
  - Merge adjacent DBMs

- Sure it is possible but how do you choose your DBMs? How many DBMs can you merge?
Removing DBMs

- DBM inclusion (cheap) or exact inclusion (more expensive).

Note: In practice we have dimension $n$. 
Merging DBMs - Principle

- Check if $\text{convex}_\text{hull}(A,B) = A\mid B$
- Problem: $2^n$ ways of choosing DBMs (2, 3, …, n). We don’t know how many DBMs we can merge together.

More complex configurations in practice.
Let’s Do It!

- Algorithms:
  - Reduce: Inclusion checking.
  - ExpensiveReduce: Exact inclusion checking.
  - 2-merge: Merge 2 by 2.
  - N-merge: Dynamically find N DBMs to merge.
  - Partitioned N-merge: Find partitions and apply N-merge + expensiveReduce.
  - ConvexReduce: Recompute the federation.
2-merge

- $N^2$ pairs to try.
- Use cheap test based on 2 necessary conditions (not sufficient):
  - 2 opposite constraints of 2 DBMs must be equal, e.g., $a_{ij} = b_{ij}$ and $a_{ji} = b_{ji}$.
  - Intersection of adherence is not empty.
- Then we try the merge with the convex hull – needs subtractions.
2-merge

OK ij

Not OK ji
2-merge

Not OK $ij + ji$
2-merge

Not OK \( ij + ji \)
2-merge

- Adherence:
  - $x < 3$ and $x \geq 3 \Rightarrow x \leq 3$ and $x \geq 3$

- We also check for DBM inclusion.

- Finally if the conditions are met, we check if $\text{convex_hull}(A,B)-(A|B)$ is empty.
N-merge

Relaxed 2-merge: only one compatible constraint.

Algorithm (inclusion check ommitted):

- For all $i < n$, for all $j < n$ & $j > i$: $n^2$
  - union := DBM[i]
      retry on all j
    - else if “1/2-merge” union |= DBM[j]
  - C := convex_hull(union)
  - For all $j < n$: if DBM[j] included in C, union |= DBM[j]
  - If R := C-union is empty replace union by C
    - Else if size(C-(C-union)) < size(union) replace union by C-(C-union)
    - Else ExpensiveReduce on union.
Partition N-merge

Algorithm:

- Find a partition of our federation
- Fixpoint on the sub-sets of
  - N-merge
  - Followed by ExpensiveReduce if there was a reduction
**ConvexReduce**

- **Idea:** Recompute the federation and reduce “fragmentation”.

- **Algorithm:**
  - $C = \text{convex}_\text{hull}(\text{Fed})$
  - $F = C - (C - \text{fed})$
  - $\text{Fed} = F$ if $\text{size}(F) < \text{size}(\text{Fed})$
Experiments: Does it work?

We need a real case example where federations are heavily used and there is much split:

- Timed game reachability algorithm, backward & forward [CDFLL05].
- Current work: Applying this algorithm to jobshop scheduling.
- Experiments on one instance with and without uncertainties – difficult instance.
- Question: Is there a winning strategy?
Based on The DBM Library

- New API based on past experience and new needs:
  - optimizations for the “close” operation
  - new extrapolations
  - federations

- Written in C, C interface to DBMs and federations.

- Federation C++ class.

  - Dual Xeon 2.8GHz, 4GB RAM, Linux 2.4.
## Without Uncertainties - Easy

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Reduce</td>
<td>5.3s</td>
<td>44.6M</td>
</tr>
<tr>
<td>Reduce</td>
<td>1.9s</td>
<td>20.8M</td>
</tr>
<tr>
<td>ExpensiveReduce</td>
<td>2.0s</td>
<td>21.1M</td>
</tr>
<tr>
<td>2-merge</td>
<td>2.0s</td>
<td>19.9M</td>
</tr>
<tr>
<td>N-merge</td>
<td>2.4s</td>
<td>19.9M</td>
</tr>
<tr>
<td>Partition N-merge</td>
<td>2.4s</td>
<td>19.9M</td>
</tr>
<tr>
<td>ConvexReduce</td>
<td>2.1s</td>
<td>19.9M</td>
</tr>
</tbody>
</table>
Without Uncertainties - Easy

- Small federations.
- Small difference between methods.
- Reduce still important.
- 2-merge best.
- Only one bottleneck in the experiment that really matters.
<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Reduce</td>
<td>4051s</td>
<td>918M</td>
</tr>
<tr>
<td>Reduce</td>
<td>7147s</td>
<td>732M</td>
</tr>
<tr>
<td>ExpensiveReduce</td>
<td>8831s</td>
<td>784M</td>
</tr>
<tr>
<td>2-merge</td>
<td>897s</td>
<td>572M</td>
</tr>
<tr>
<td>N-merge</td>
<td>372s</td>
<td>526M</td>
</tr>
<tr>
<td>Partition N-merge</td>
<td>345s</td>
<td>525M</td>
</tr>
<tr>
<td>ConvexReduce</td>
<td>415s</td>
<td>532M</td>
</tr>
</tbody>
</table>
With Uncertainties - Difficult

- 2-merge best for simple cases, as before.
- Partition & N-merge best for complex cases. If we generate the strategy, N-merge is best.
- One bottleneck that really matters.
Conclusion

- It works and it is **very** important to reduce federations.
- Best method (cheap/expensive) depends on the application.
  - Expensive method on critical bottlenecks.
- Efficient in practice.