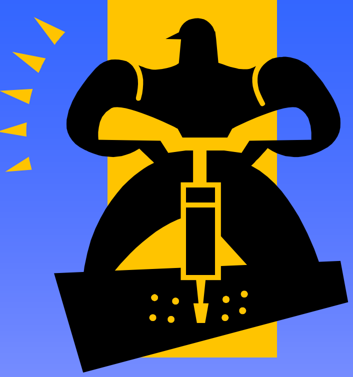
# ETMCC v2.0

**Ivan S Zapreev,**

**Maneesh Khattri**

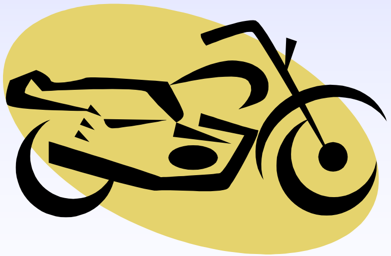
# Outline

- Goals
- Data structures and Algorithms
- PRCTL
- ETMCC v2.0 vs. v1.0
- Extensions
- Conclusions



# Goals

- Develop a unified tool for **PCTL**, **CSL**, **PRCTL** and **CSRL**
- Improve and extend ETMCC v1.0
  - Use efficient data structures
  - Use improved algorithms for **CSL**
    - Steady state detection
    - Faster until operators
    - Faster BSCCs search
    - etc. ;



# Data structures and Algorithms

- Data structures:
  - Sparse Matrix special representation
  - Fast Matrix Vector multiplication
  - Linear memory allocation
  - Predecessor sets
- Algorithms:
  - Direct search only for required BSCCs
  - Bounded until (CSL)
  - Unbounded until (CSL)
  - Collapse $\varphi \,\&\, \neg\phi \wedge \neg\varphi$ states
  - Bisimulation minimization
  - On the fly steady state detection

# Data Structure

- Make states absorbing
- Compute Uniformized DTMC from CTMC

$$M = \begin{bmatrix} 0.5 & 0.15 & 0.0 \\ 0.25 & 0.0 & 0.75 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \Rightarrow \begin{bmatrix} \mathit{Ncols} = 2 \\ \mathit{Diag} = 0.5 \\ \mathit{Column} \to [2] \\ \mathit{Value} \to [0.15] \\ \hline \mathit{Ncols} = 2 \\ \mathit{Diag} = 0 \\ \mathit{Column} \to [1, 3] \\ \mathit{Value} \to [0.25, 0.75] \\ \hline \mathit{Ncols} = 0 \\ \mathit{Diag} = 0 \\ \mathit{Column} \to \mathit{NULL} \\ \mathit{Value} \to \mathit{NULL} \end{bmatrix}$$

# Fast M*v and v*M multiplication

- Linear memory representation,
- Multiply only certain – valid elements,
- Rely on the matrix row elements ordering,
- Multiply v*M by row;

$$M = \begin{bmatrix} 0.5 & 0.5 & 0.0 \\ 0.25 & 0.0 & 0.75 \\ 0.0 & 0.3 & 0.7 \end{bmatrix}$$

| | 1 row | 2 row | | 3 row |
|---|---|---|---|---|
| Col. id | 1 | 0 | 2 | 1 |
| Val. | 0.5 | 0.25 | 0.75 | 0.3 |
| Diag. | 0.5 | 0.0 | | 0.7 |

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} * \begin{bmatrix} \boxed{0.5} & 0.5 \\ 0.25 & 0.75 \\ \boxed{0.7} & 0.3 \end{bmatrix} = \begin{matrix} & \begin{bmatrix} 0.05 & 0.05 & 0.0 \end{bmatrix} \\ + & \begin{bmatrix} 0.05 & 0.0 & 0.15 \end{bmatrix} \\ + & \begin{bmatrix} 0.0 & 0.09 & 0.21 \end{bmatrix} \end{matrix} = \begin{bmatrix} 1.0 & 0.14 & 0.36 \end{bmatrix}$$

# Re-sorting the matrix

Speed up matrix vector multiplication in iterative methods for steady state operator (BSCC) and making states absorbing.

$$M = \overbrace{\boxed{0.5}\boxed{0.5}}^{\textbf{1 row}}\overbrace{\boxed{0.2}\boxed{0.8}}^{\textbf{2 row}}\overbrace{\boxed{0.25}\boxed{0.75}}^{\textbf{3 row}}\overbrace{\boxed{0.1}\boxed{0.8}\boxed{0.1}}^{\textbf{4 row}} \Leftarrow \text{nnz}$$

$$M = \boxed{\phantom{0.5}}\boxed{\phantom{0.5}}\boxed{\phantom{0.5}}\boxed{\phantom{0.5}}\overbrace{\boxed{0.2}\boxed{0.8}}^{\textbf{2 row}}\overbrace{\boxed{0.1}\boxed{0.8}\boxed{0.1}}^{\textbf{4 row}}$$

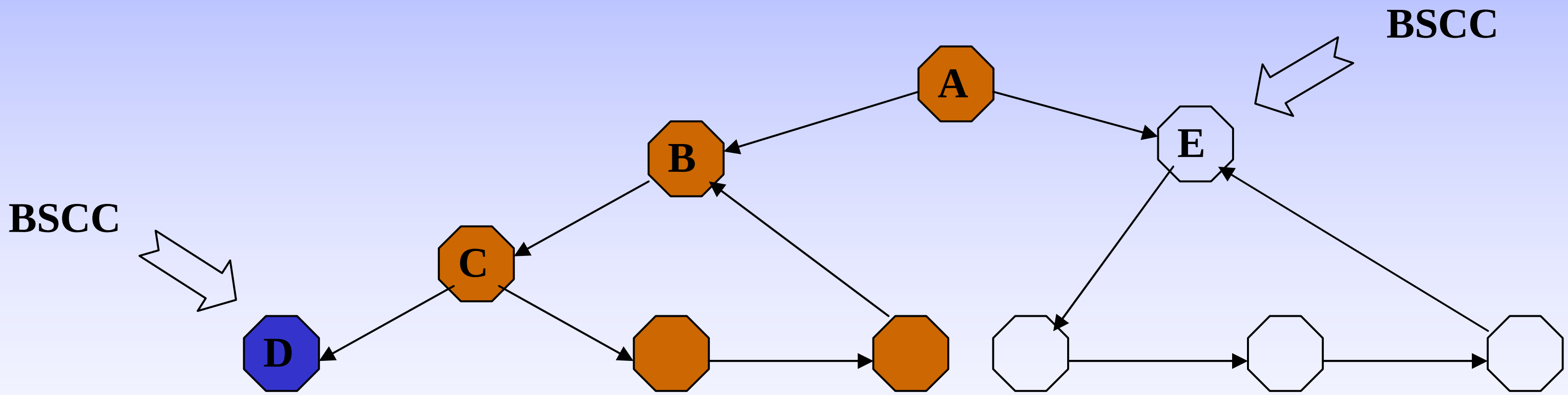
$$Ms = \overbrace{\boxed{0.5}\boxed{0.5}}^{\textbf{1 row}}\overbrace{\boxed{0.25}\boxed{0.75}}^{\textbf{3 row}}$$

# Searching for BSCCs, $S_{\triangleleft p}(F)$

Based on the Tarjan's algorithm for searching MSCCs:

- Search for BSCC, not MSCC
- Find BSCCs for Set(F) states
- If $x \in Set(F) \wedge \exists\, path\ from\ x\ to\ MSCC$ => stop

The complexity remains O(N+M)



# Steady state detection

$$\sum_{k=0}^{\infty} e^{-qt} \frac{(q \cdot t)^k}{k!} \overrightarrow{p_0} \cdot P^k$$

- Steady state

$$\exists k : \forall j > 0 \ \overrightarrow{p_0} \cdot P^k \approx \overrightarrow{p_0} \cdot P^{k+j}$$

- Detect steady state
  - Check sequence convergence

$$\left| \overrightarrow{p_0} \cdot P^{i \cdot m} - \overrightarrow{p_0} \cdot P^{(i-1) \cdot m} \right| \xrightarrow[i]{} 0$$

  - Check final convergence

$$\left| \overrightarrow{p_0} \cdot P^{s+1} - \overrightarrow{p_0} \cdot P^{s} \right| \leq \frac{\varepsilon}{8}$$

# Unbounded until

- Unbounded Until
- $U_0$= no path through Φ-states exists to a Ψ-state
- $U_1$= Prob. Measure to reach Ψ-state through Φ-states is 1
- $U_? = S \setminus (U_0 \bigcup U_1)$
- Then Prob(s, Φ U Ψ) is:

$$\underline{x}_s = \begin{cases} 0 : s \in U_0 \\ 1 : s \in U_1 \\ \sum_{s' \in U_?} P(s,s') \cdot \underline{x}_{s'} + \sum_{s' \in U_1} P(s,s') : s \in U_? \end{cases}$$

# Bounded until (CSL)

## Bounded Until can be computed as:

- The solution:

$$\mathrm{Prob}^{\mathrm{M}}\left(\phi\ \mathrm{U}^{\leq \mathrm{t}}\ \varphi\right) = \sum_{s''} \pi^{M[\neg\phi \vee \varphi]}\left(s, s'', t\right)$$

$$\pi^{M[\neg\phi \vee \varphi]}\left(s, s'', t\right) = \mathrm{Prob}^{\mathrm{M}[\neg\phi \vee \varphi]}\left(s, \Diamond^{[t,t]} at_{s''}\right)$$

- Using uniformisation:

$$\mathrm{Prob}\left(\phi\, U^{\leq t}_{\triangleright\triangleleft p}\ \varphi\right) = e^{q \cdot t \cdot (P - I)} \cdot \vec{i}_{\varphi} = \sum_{k=0}^{\infty} e^{-qt} \frac{(q \cdot t)^k}{k!} P^k \cdot \vec{i}_{\varphi}$$

- Interval Until: Modification of $\mathrm{Prob}^{\mathrm{M}}\left(\phi\ \ \mathrm{U}^{\leq \mathrm{t}}\ \ \varphi\right)$

# Outline

- Goals
- Data structures and Algorithms
- PRCTL
- ETMCC v2.0 vs. v1.0
- Extensions
- Conclusions

# PRCTL logic

- Syntax:

$$\Phi ::= tt \mid a \in AP \mid \Phi \wedge \Phi \mid \neg\Phi \mid L_{\lhd p}(\Phi) \mid P_{\lhd p}\left(\Phi U_J^N \Phi\right) \mid$$

$$\mathrm{E}_J^n(\Phi) \mid \mathrm{E}_J(\Phi) \mid C_J^n(\Phi) \mid Y_J^n(\Phi)$$

- Semantics:

$L_{\lhd p}(\Phi)$ - Long-run probability meets probability bound

$P_{\lhd p}\left(\Phi U_J^N \Phi\right)$ - Path-probability operator for until formula

$\mathrm{E}_J^n(\Phi)$ - Expected reward rate at n-th transition for phi-states meets reward bound

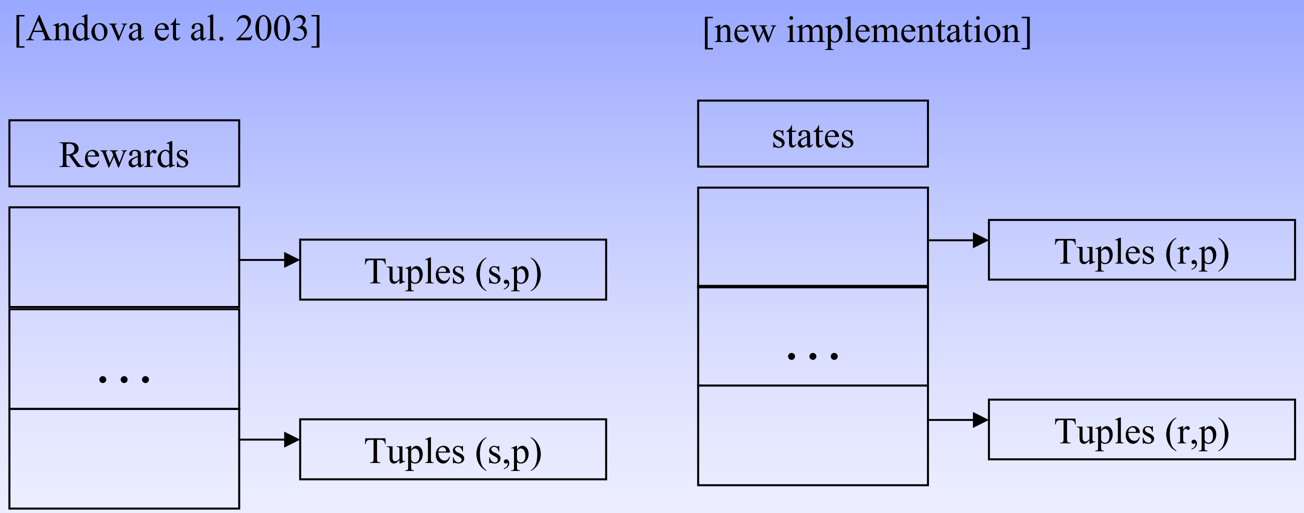
$\mathrm{E}_J(\Phi)$ - Long-run expected reward rate for phi-states meets reward bound

$C_J^n(\Phi)$ - Instantaneous reward rate in phi-states meets the reward bound

$Y_J^n(\Phi)$ - Expected accumulated reward until the n-th transition meets the reward bound

# PRCTL model checking

- All formulas have been implemented
- Slight modifications: Until formula (Path Graph)



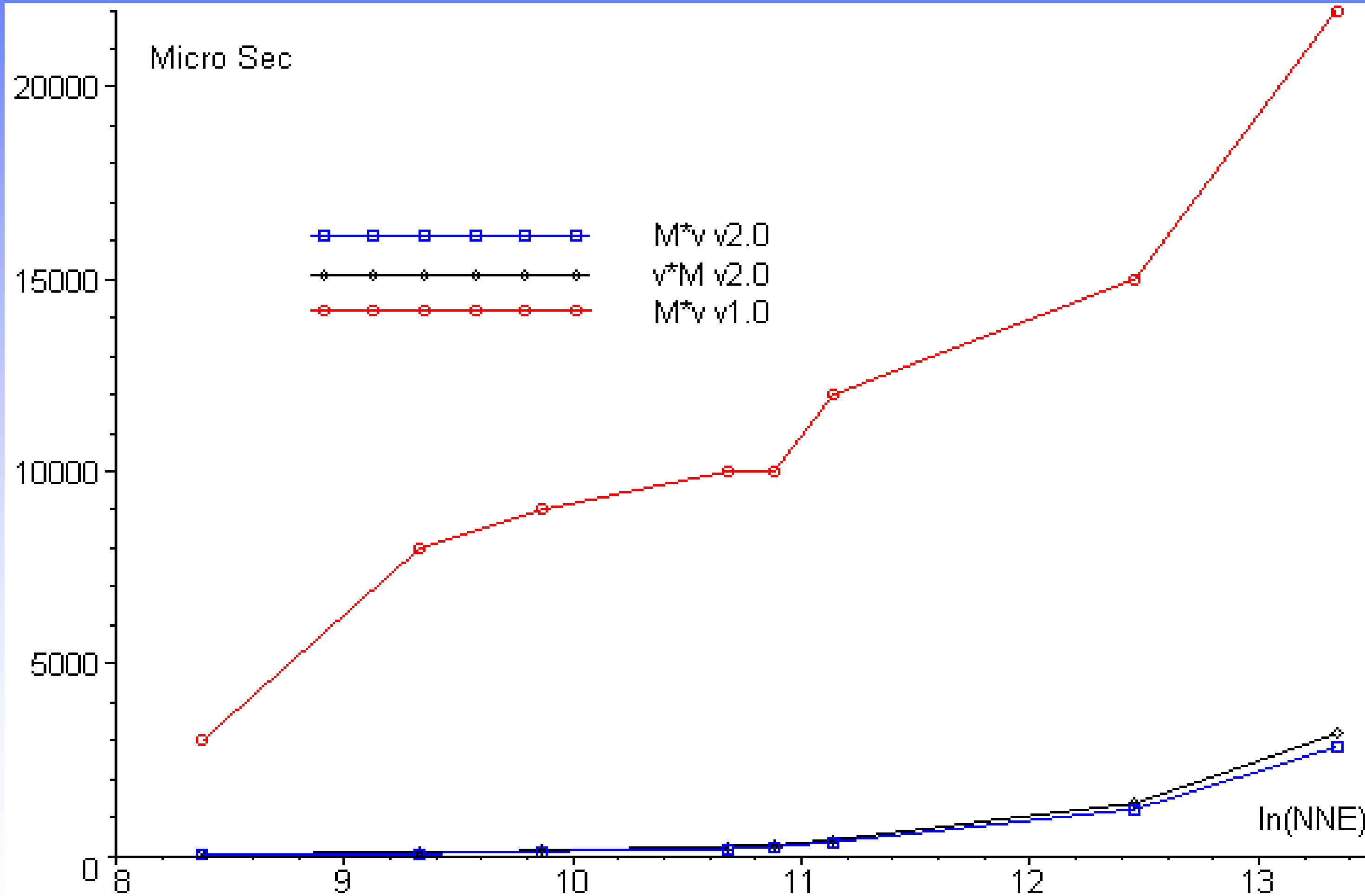

- Better access to the rate matrix

# Outline

- Goals
- Data structures and Algorithms
- PRCTL
- ETMCC v2.0 vs. v1.0
- Extensions
- Conclusions

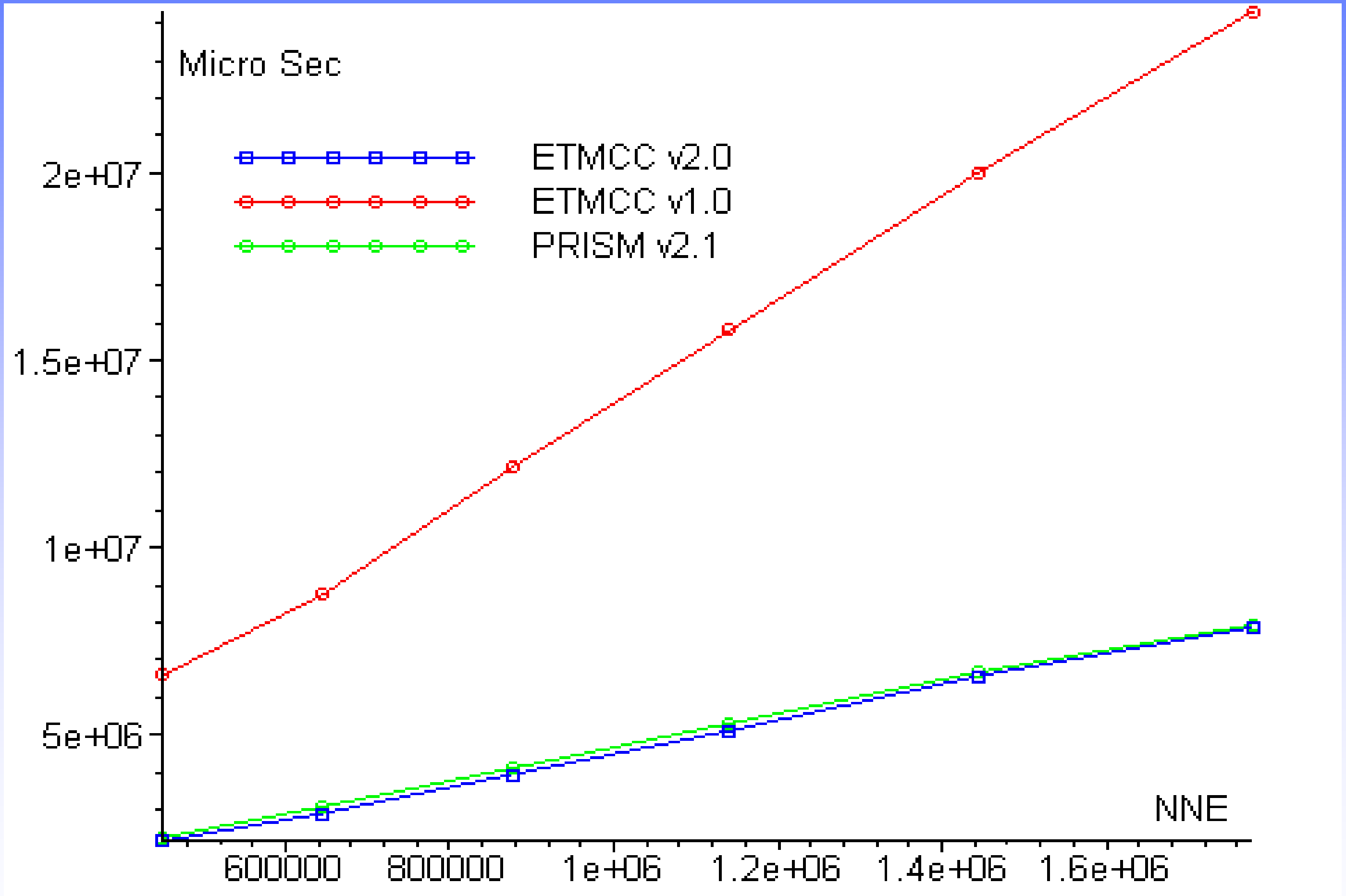
# ETMCC v2.0 vs. v1.0

Matrix vector multiplication: $NNE \in [4374, 623554]$



# ETMCC v2.0 vs. v1.0

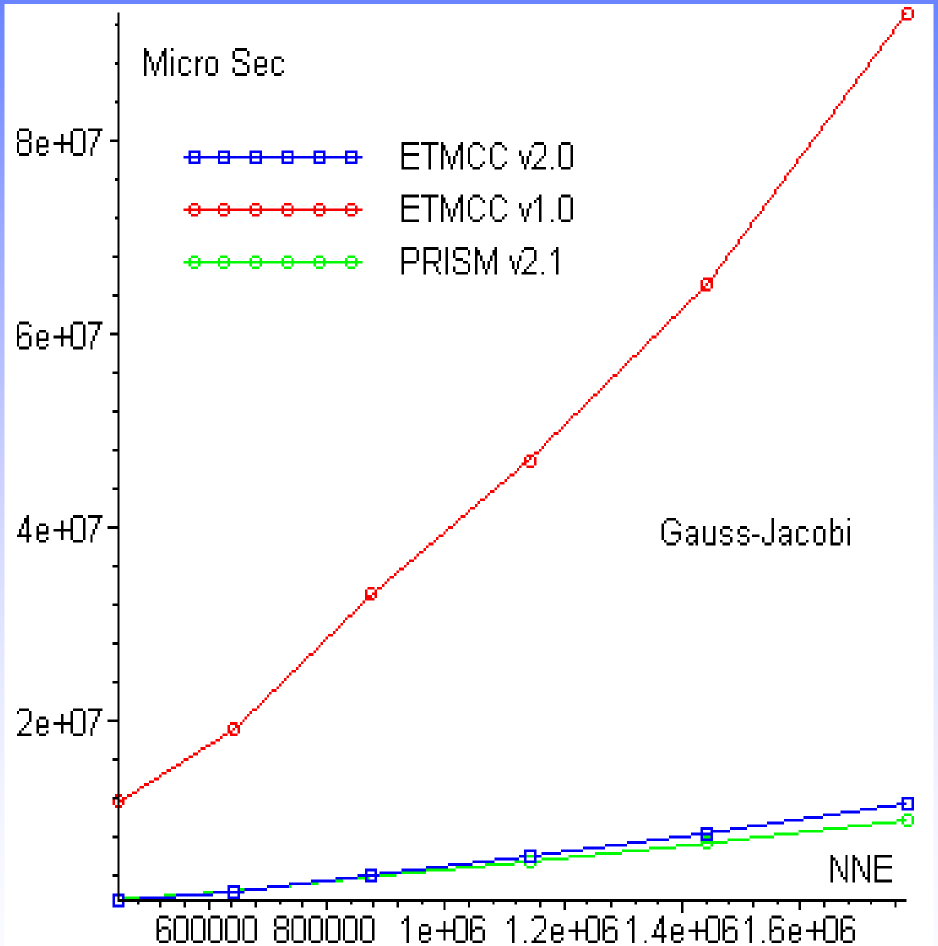
The Cluster Computing example $P_{<0.1}\left(tt\ U^{\leq 15} !minimum\right)$

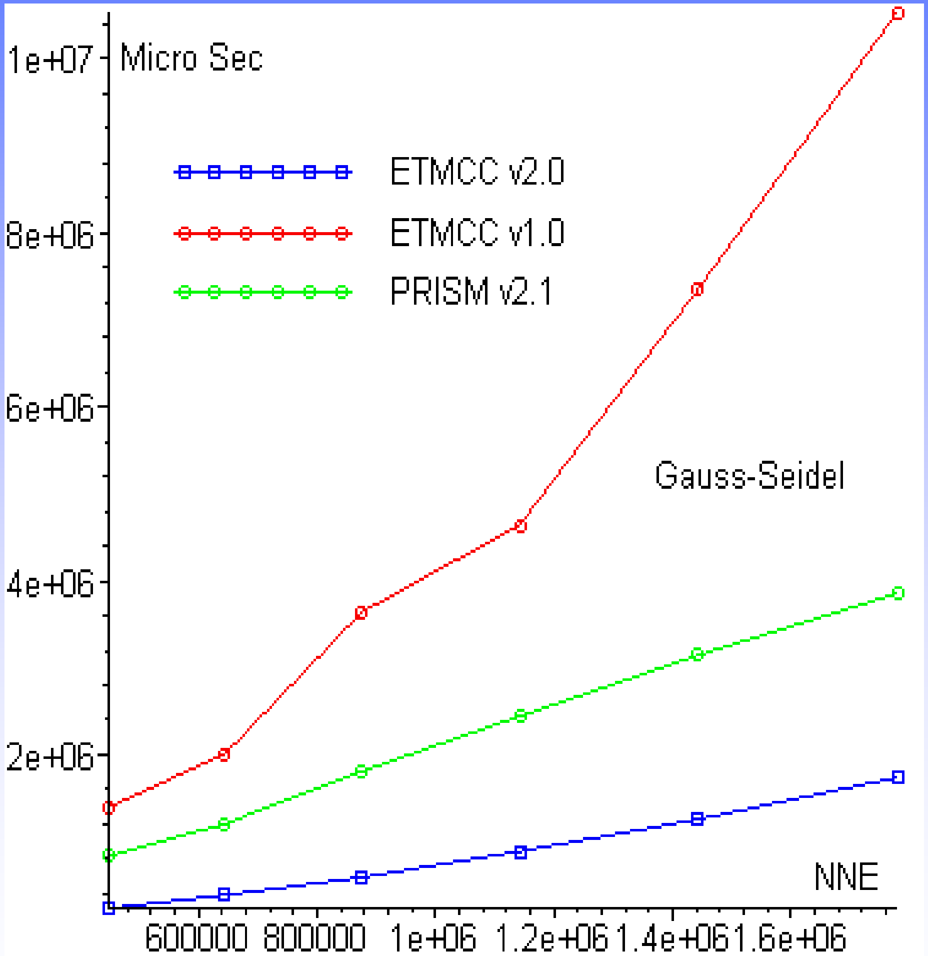



http://www.cs.bham.ac.uk/dxp/prism/cluster.html

# ETMCC v2.0 vs. v1.0

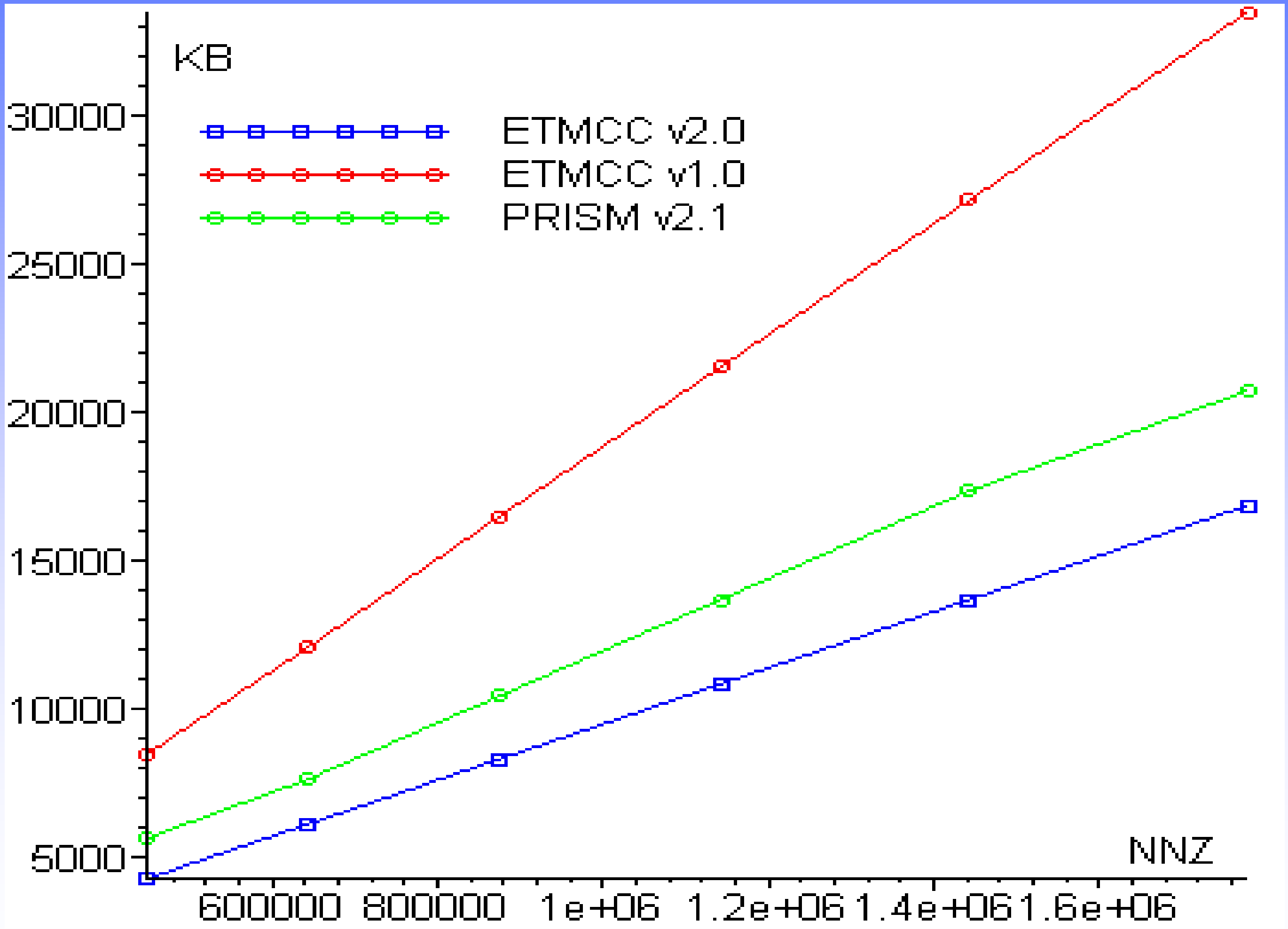
The Cluster Computing example $S_{<0.05}(!minimum)$



# ETMCC v2.0 vs. v1.0

## The Cluster Computing example, space usage



# Outline

- Goals
- Data structures and Algorithms
- PRCTL
- ETMCC v2.0 vs. v1.0
- Extensions
- Conclusions

# Krylov Subspaces

$$\pi^{M[\neg\phi\vee\varphi]}(s, s'', t) = \mathrm{Prob}^{M[\neg\phi\vee\varphi]}\left(s, \Diamond^{[t,t]} at_{s''}\right)$$

- Solution of the linear differential equation $w(t) = e^{t \cdot M[\neg\phi\vee\varphi]} \cdot \vec{v}$
- Typical approach – Uniformisation
  - Makes elements positive
- For $\lambda t > 500$ Krylov-based algorithms work better [R. Sidje].
  - Compute $e^{t \cdot M[\neg\phi\vee\varphi]} \cdot \vec{v}$ at once
  - Map w(t) onto a much smaller subspace

# The $\mathrm{Prob}\left(\phi\, U^{\le t}_{\triangleright\triangleleft p}\, \varphi\right)$ estimate

**Idea 1:** In DTMC consider only some paths to $\varphi$.
Get $Min^k_\varphi \le P^k \cdot \vec{i}_\varphi$ estimate.

**Idea 2:** In DTMC compute $Min^k_{\neg\phi\wedge\neg\varphi}$,
then $Max^k_\varphi = 1 - Min^k_{\neg\phi\wedge\neg\varphi}$.

**Idea 3:** The estimates for CTMC are now obtained with the formula:

$$\mathrm{Prob}\left(\phi\, U^{\le t}_{\triangleright\triangleleft p}\, \varphi\right) = \sum_{k=L_\varepsilon}^{U_\varepsilon} e^{-qt} \frac{(q\cdot t)^k}{k!} P^k \cdot \vec{i}_\varphi$$

# Example for the $\mathrm{Prob}\left(\phi\, U^{\le t}_{\rhd\lhd p}\, \varphi\right)$ estimate

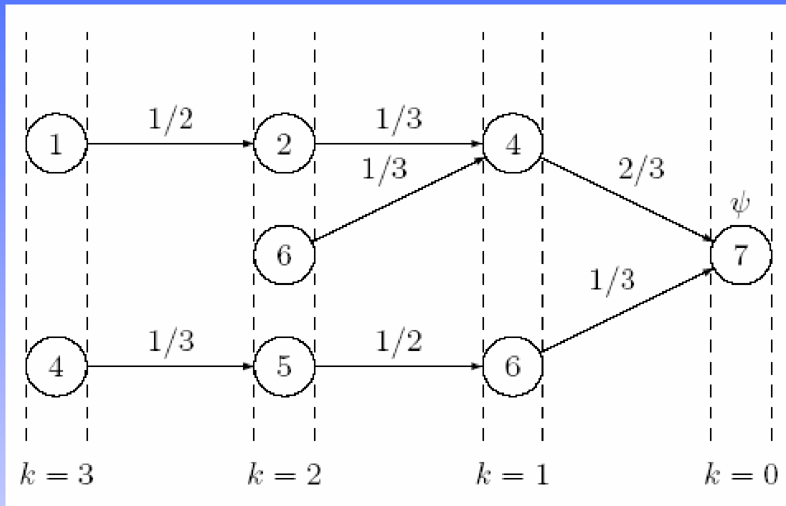



Figure 2: Some of the finite paths.

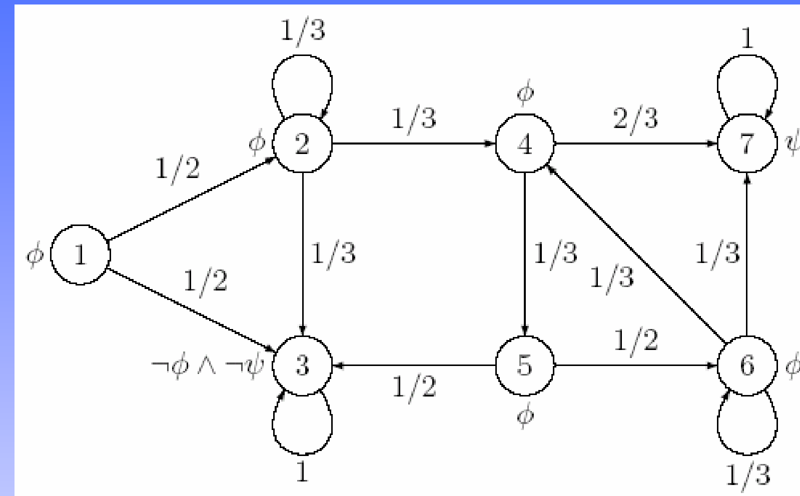



Figure 1: The Initial Example.

k=0:

$P^0_{min}$=(0, 0, 0, 0, 0, 0, 1)

$P^0$ =(0, 0, 0, 0, 0, 0, 1)

k=1:

$P^1_{min}$=(0, 0, 0, 2/3, 0, 1/3, 1)

$P^1$ =(0, 0, 0, 2/3, 0, 1/3, 1)

k=2:

$P^2_{min}$=(0, 2/9, 0, 2/3, 1/6, 5/9, 1)

$P^2$ =(0, 2/9, 0, 2/3, 1/6, 2/3, 1)

k=3:

$P^3_{min}$=(1/9, 2/9, 0, 13/18, 5/18, 5/9, 1)

$P^3$ =(1/9, 8/27, 0, 13/18, 1/3, 7/9, 1)

k=5:

$P^5_{min}$=( 1/9, 2/9, 0, 13/18, 5/18, 5/9, 1)

$P^5$ =(55/324, 181/486, 0, 43/54, 5/12, 47/54, 1)

# Conclusions

- Support PCTL, CSL, PRCTL logics
- The implementation is faster
- Ongoing work:
  - Incorporate CSRL logic
  - Collapse $\neg\phi \vee \varphi$ states
  - Involve bisimulation minimization
  - Further v*M, M*v improvement

# Any further wishes?

- Link to BDD implementation?
- Graphic User Interface?
- etc.