Implementing Counters with Decay

Christian Schulte
KTH Royal Institute of Technology, Sweden
Decaying Counters

- Counters for counting events used in search heuristics
  - AFC (accumulated failure count = weighted degree)
    count how often a constraint has failed
    sum up over all constraints of a variable
    [Boussemart et al., ECAI 2004]
  - activity
    count how often a variable has been pruned
    [Michel, Van Hentenryck, CPAIOR 2012]

- Values should decay as search progresses
  - random restarts: must decay

- Decay: if counter not incremented, scale with decay factor $\gamma$
  $(0 < \gamma \leq 1)$
Problem

• Data structures
  • counters $c_1, \ldots, c_n$ (floating point numbers, C++: `double`)

• Implementation
  
  
  ```
  proc inc(i) =
  \text{ } c_i \leftarrow c_i + 1; \forall j \neq i \text{ do } c_j \leftarrow c_j \cdot \gamma;
  
  fun val(i) =
  \text{ } \text{return } c_i;
  ```
  
  • complexity of inc($i$): $O(n)$

• **But:** counter set possibly not known when doing inc($i$)
  • no `forall` possible

• **But:** inc($i$) might suffer from contention with parallel search
  • all counters must be locked and $O(n)$ operations!
Solution

• Data structures
  • counters $c_i = \langle n_i, t_i \rangle$  “$\langle \text{value, timestamp} \rangle$”
  • global timestamp $t$

• Implementation
  \[
  \text{proc } \text{inc}(i) = n_i \leftarrow n_i \cdot \text{pow}(\gamma, t - t_i) + 1; \ t \leftarrow t + 1; \ t_i \leftarrow t;
  \]
  \[
  \text{fun } \text{val}(i) = n_i \leftarrow n_i \cdot \text{pow}(\gamma, t - t_i); \ t_i \leftarrow t; \ \text{return } n_i;
  \]
  • complexity of inc($i$): $O(1)$
  • AFC: often $O(n^2)$ calls to val($i$) for each call to inc($i$)

• Optimizations
  • cache for likely exponents of pow (which is expensive to compute)
  • do not do anything for $\gamma=1$