

Tree Optimisation for Performance Tree Evaluation

Harini Kulatunga

24 January 2008 / AESOP Meeting

Summary

- ▶ Performance Trees
- ▶ Tree Data Structure
- ▶ Hashing
- ▶ Dynamic Programming Algorithm
- ▶ Dependency and Parallelism
- ▶ **Grid Scheduling**
- ▶ Memory Management
- ▶ Conclusions

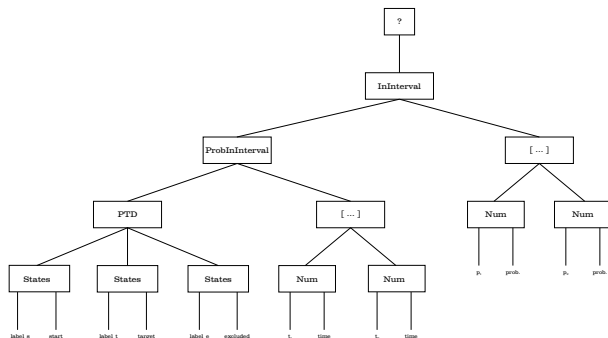
Motivation

- ▶ Dynamic programming for optimal code generation - Aho and Johnson(1976), Bernstein et.al.(1987)
- ▶ Hashing of data structures - Sedgewick(1983), C++ Hash Table Memoization

Basic passage-time query

- ▶ Measures the probability that a state machine reaches one of a set of target states provided,
 - ▶ the time duration is between t_1 and t_2
 - ▶ the path does not include any one of the excluded set of states
 - ▶ the start state is one of a specified set

Performance Tree Representation - Suto et.al.

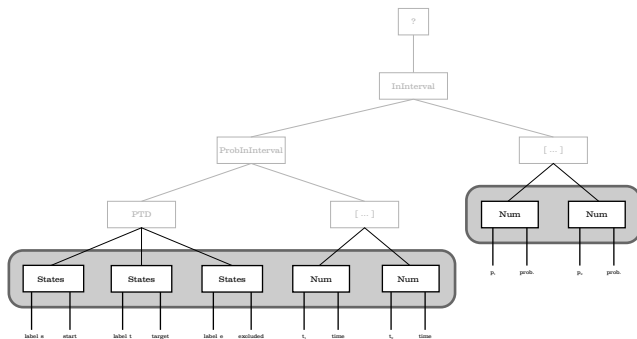


PTD - Passage Time Density function,

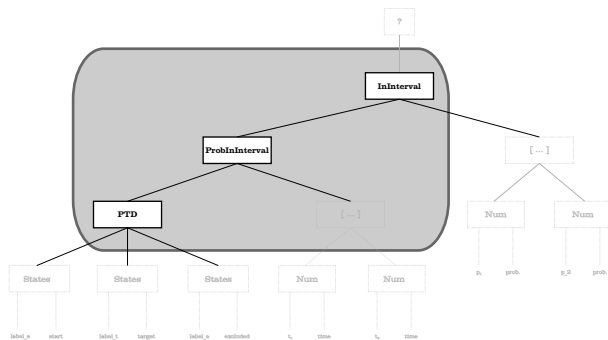
ProbInInterval - Gives the probability of the first passage time being within a given time interval

InInterval - A boolean operator which determines whether a given value falls within a given interval

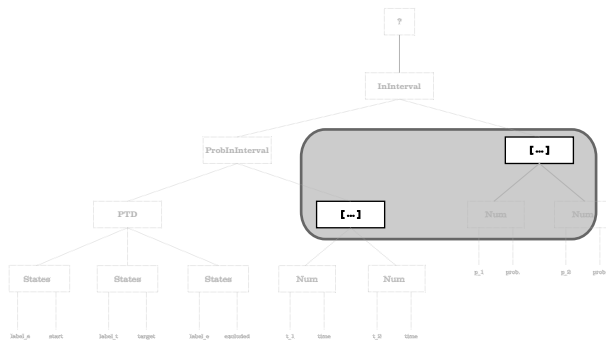
Leaf Nodes - Data Nodes



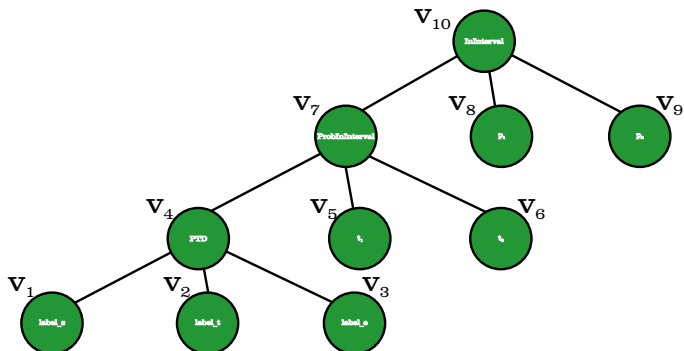
Root Nodes I - Computation Nodes



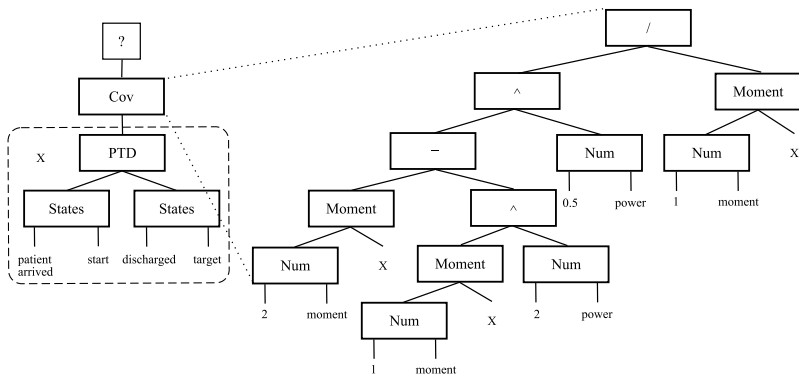
Root Nodes II - Conceptual Nodes



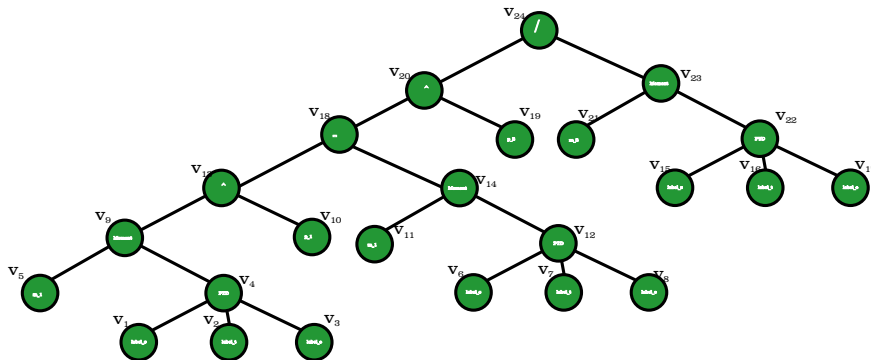
Tree Structure for Example 1



Simple tree structure that can be evaluated sequentially with a postorder traversal sequence, $V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10}$

Complex Performance Query : **Cov**

Tree Structure for Macro Cov



Hashing - Leaf Nodes

- ▶ Each Leaf Node's Hash Key is defined using 3 distinct values,
 - ▶ 4th integer storing the string "**DATA**".
 - ▶ 2 – 3rd integers storing the Mean and Variance of the array of data at each Leaf Node.
 - ▶ 1st integer stores "-1".
- ▶ (Hash Key, ptr_data, Semaphore=TRUE, depth_of_node) are stored in the Hash Table.
- ▶ Collisions are determined by comparing the Hash Table Index and the data.

Hashing - Computation Nodes

- ▶ Each Computation Node is assigned an $OP_CODE = \text{"PT Syntax"}$, e.g. $OP_CODE = \text{"PTD"}$
- ▶ Each computation node's Hash Key is defined using up to 4 integers with,
 - ▶ 4th integer storing the OP_CODE of the Computation Node
 - ▶ 1 – 3rd integers will store the Hash Table Indices (TI) of the Child Nodes
- ▶ Every node will have a unique TI that will be stored in the tree itself.

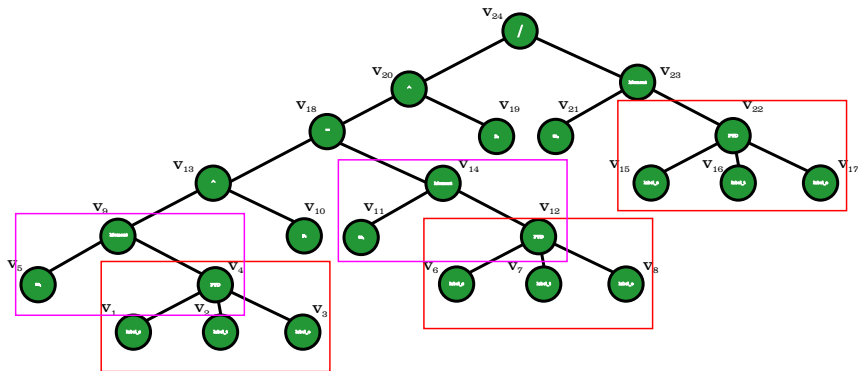
Dynamic Programming with Hashing Constraints

1. Assign vertex numbers in postorder maintaining precedence, i.e., for $i > j$, vertex $v_i \notin T_{v_j}$
2. Assign all Leaf nodes to set L and all C. Nodes to set C
3. Set $i = 1, j = 1$
4. **While** $i \leq |L|$ **or** $j \leq |C|$ **do**
5. **IF** $i > |L|$ **then**
6. « add the remaining C. nodes to the Hash Table and Job Queue »
7. **ELSEIF** $j > |C|$ **then**
8. « add the remaining Leaf nodes to the Hash Table »

Dynamic Programming with Hashing Constraints

9. **ELSEIF** $l_i \in T_{c_j}$
10. « IF l_i is not in Hash Table then add l_i to the Hash Table »
11. $i = i + 1$
12. **ELSEIF** l_i and c_j are not in Hash Table **then**
13. « add l_i, c_j to the Hash Table and add c_j to Job Queue »
14. $i = i + 1, j = j + 1$
15. **ELSEIF** l_i is in Hash Table **then**
16. $i = i + 1$
17. **ELSEIF** c_j is in Hash Table **then**
18. $j = j + 1$
19. **Return** Job Queue

Tree Traversal for Example 2: Cov

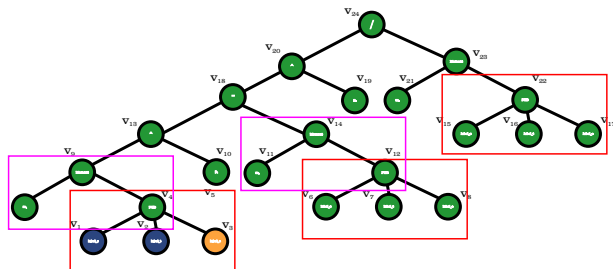


$$L = (v_1 \ v_2 \ v_3 \ v_5 \ v_6 \ v_7 \ v_8 \ v_{10} \ v_{11} \ v_{15} \ v_{16} \ v_{17} \ v_{19} \ v_{21}); \quad i = 1 - 14$$

$$C = (v_4 \ v_9 \ v_{12} \ v_{13} \ v_{14} \ v_{18} \ v_{22} \ v_{20} \ v_{23} \ v_{24}); \quad j = 1 - 10$$

Breadth first postorder sequences for Leaf nodes
and Computation Nodes

Tree Traversal for Example 2: Cov

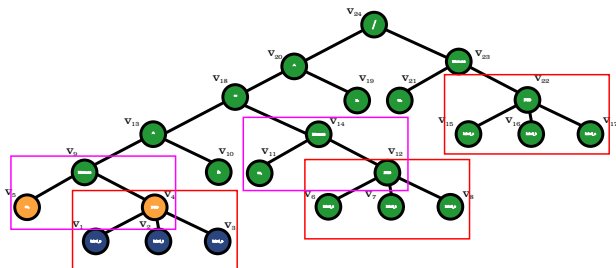


Keys	Data	Semaphore	Depth
-1 ** ** ** DATA	PTR_DATA	TRUE	7
-1 ** ** ** DATA	PTR_DATA	TRUE	7
-1 ** ** ** DATA	PTR_DATA	TRUE	7

Job Queue

--	--	--	--

Tree Traversal for Example 2: Cov

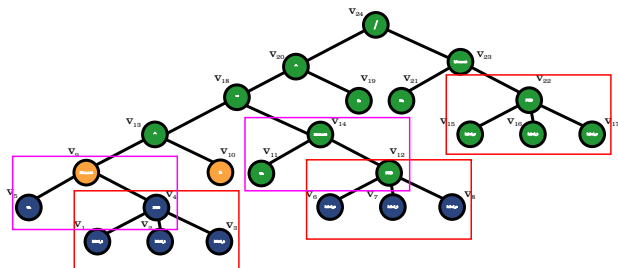


Keys	Data	Semaphore	Depth
-1 ## ## DATA	PTR_DATA	TRUE	7
-1 ## ## DATA	PTR_DATA	TRUE	7
-1 ## ## DATA	PTR_DATA	TRUE	7
TI 1 ## TI 2 ## TI 3 ## PTD		FALSE	6
-1 ## ## DATA	PTR_DATA	TRUE	6

Job Queue

V4			
----	--	--	--

Tree Traversal for Example 2: Cov

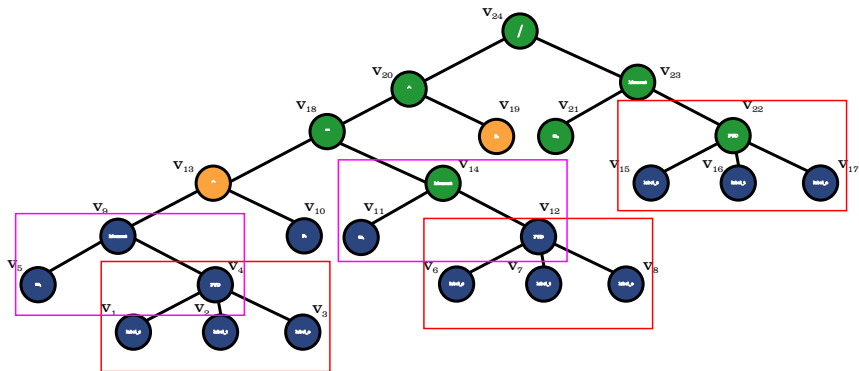


Keys	Data	Semaphore	Depth
-1 ## ## DATA	PTR_DATA	TRUE	6
-1 ## ## DATA	PTR_DATA	TRUE	6
-1 ## ## DATA	PTR_DATA	TRUE	6
TI_1 ## TI_2 ## TI_3 ## PTD	PTR_DATA	FALSE	6
-1 ## ## DATA	PTR_DATA	TRUE	6
TI_1 ## TI_2 ## TI_3 ## MOBA	PTR_DATA	FALSE	5
-1 ## ## DATA	PTR_DATA	TRUE	5

Job Queue

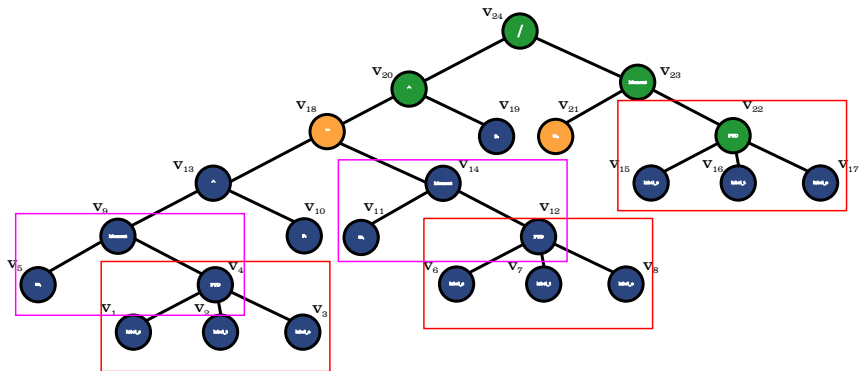
V ₄	V ₉		
----------------	----------------	--	--

Tree Traversal for Example 2: Cov



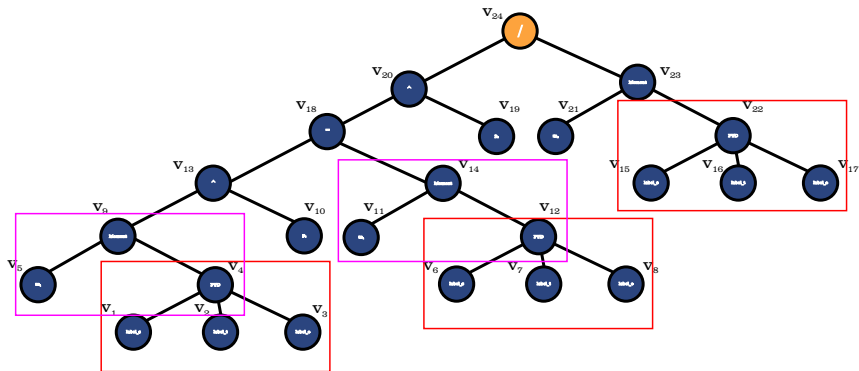
		1	2	3	4	5	6
PQ	l	v ₁	v ₂	v ₃	v ₅	v ₁₀	v ₁₉
	c			v ₄	v ₉	v ₁₃	

Tree Traversal for Example 2: Cov



		1	2	3	4	5	6	7
PQ	l	v ₁	v ₂	v ₃	v ₅	v ₁₀	v ₁₉	v ₂₁
	c			v ₄	v ₉	v ₁₃	v ₁₈	

Tree Traversal for Example 2: Cov



		1	2	3	4	5	6	7	8	9	10
PQ	l	v ₁	v ₂	v ₃	v ₅	v ₁₀	v ₁₉	v ₂₁			
	c			v ₄	v ₉	v ₁₃	v ₁₈	v ₂₀	v ₂₃	v ₂₄	

Dependency between Computation Nodes

- ▶ The Hash Table will store four parameters
 - ▶ H_Key
 - ▶ PTR_DATA
 - ▶ SEMAPHORE - is always signalled for Leaf nodes
 - ▶ DEPTH
- ▶ A SEMAPHORE (mutex) in the Hash Table blocks access to the DATA until the node finishes computation
- ▶ Dependent computation nodes in the queue go to SLEEP while the SEMAPHORE is locked.

Parallelism

- ▶ All the Jobs in the Queue are submitted at the same time to the Sun GridEngine (SGE).
- ▶ Start of computation is determined by the dependency graph created using Semaphores.
- ▶ Unrelated nodes will start computation.
- ▶ Dependent nodes will go to sleep until the Semaphores of their child nodes are signalled.
- ▶ They will then start computation when resources become available.

Scheduling of Computation Nodes

- ▶ Resource management and scheduling jobs will be done by the SGE
- ▶ Sun GridEngine 6.0 contains scheduling algorithms which allow policy based resource allocation, Job Checkpointing and support DRMAA resource management

Memory Management

- ▶ Node **DEPTH** is the highest depth in the tree of that node.
- ▶ Node **DEPTH** is stored in the Hash Table.
- ▶ Both input and intermediate data will be released from memory once all the dependent nodes have finished computation.

Future Considerations ...

- ▶ Implementation of the tree-based optimisation and SGE scheduling.
- ▶ Developing more efficient scheduling algorithm, e.g., scheduling subtree computations at different processors.
- ▶ Graph-based optimisation methods.

The End.