Experiments on Union-Find Algorithms for the Disjoint-Set Data Structure

Md. Mostofa Ali Patwary* Jean Blair** Fredrik Manne*

*Department of Informatics, University of Bergen, Norway

**Department of EE and CS,United States Military Academy, USA

May 20-22, 2010
SEA 2010, Italy.

Overview

- Extensive experimental study comparing 55 different variations of UNION-FIND algorithm.
- The study includes:
  - All the classical algorithms.
  - Several recently suggested enhancements.
  - Different combinations and optimizations of these.
- Main Result: A somewhat forgotten simple algorithm developed by Martin Rem in 1976 is the fastest algorithm.

Related Experimental Studies

Reference	Application	Computing	# of Algorithms
[Liu, 1990]	Sparse matrix	Factorization	2
[Gilbert et al., 1994]	Sparse matrix	Factorization	6
[Wassenberg et al., 2008]	Image processing	Labeling	8
[Wu et al., 2009]	Image processing	Labeling	3
[Hynes, 1998]	Graphs	Connected components	18
[Osipov et al., 2009]	Graphs	Minimum spanning tree	2

Outline

Disjoint-Set Data Structure: Definitions

- $U \Rightarrow$ set of n elements and $S_i \Rightarrow$ a subset of U.
- S_1 and S_2 are disjoint if $S_1 \cap S_2 = \emptyset$.
- Maintains a dynamic collection $S_1, S_2, \ldots S_k$ of disjoint sets which together cover U.
- Each set is identified by a representative x.
- A set of algorithms that operate on this data structure is often referred to as a UNION-FIND algorithm.

Main Operations

- Each set is represented by a rooted tree, pointer towards root.
- The element in the root node is the representative of the set.
- Parent pointer $p(x)$ denotes the parent of node x.
- Two main operations.
  - $\textsc{Find}(x)$.
  - $\textsc{Union}(x, y)$.

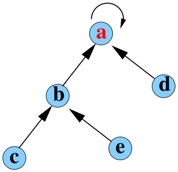


Figure: $S_i = \{a, b, c, d, e\}$.

FIND(x)

- To which set does a given element x belong ⇒ FIND(x).
- Returns the root (representative) of the set that contain x.

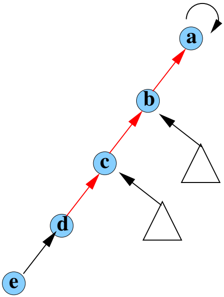


Figure: Find(d).

$\textsc{Union}(x, y)$

- Create a new set from the union of two existing sets containing x and $y \Rightarrow \textsc{Union}(x, y)$.
- Change the parent pointer of one root to the other one.

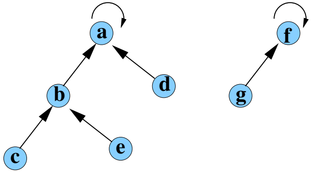


Figure: $\textsc{Union}(c, g)$.

UNION(x, y)

- Create a new set from the union of two existing sets containing x and $y \Rightarrow$ UNION(x, y).
- Change the parent pointer of one root to the other one.

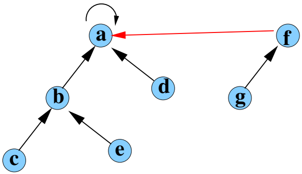


Figure: UNION(c, g).

Use of UNION-FIND for Computing Connected Components: $G = (V, E)$

1: $S \leftarrow \emptyset$
2: **for** each $x \in V$ **do**
3: MAKESET(x)
4: **for** each $(x, y) \in E$ **do**
5: **if** FIND(x) $\neq$ FIND(y) **then**
6: UNION(x, y)
7: $S \leftarrow S \cup \{(x, y)\}$

- Note that if the edges are processed by increasing weight then this algorithm is Kruskal's algorithm.

Union Techniques

- Naive-Link (nl)
- Link-by-Size (ls)
- Link-by-Rank (lr)

Union Techniques : Link-by-Rank (lr)

- Each set maintains a rank value, intially 0.
- Lowest ranked root ⇒ higher ranked root.
- Equal ranked roots ⇒ root of the combined tree is increased by 1.

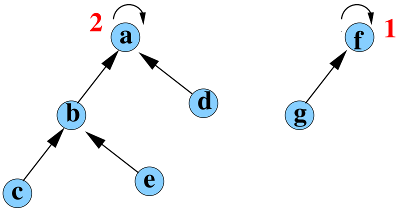


Figure: Union.

Union Techniques : Link-by-Rank (lr)

- Each set maintains a rank value, intially 0.
- Lowest ranked root ⇒ higher ranked root.
- Equal ranked roots ⇒ root of the combined tree is increased by 1.

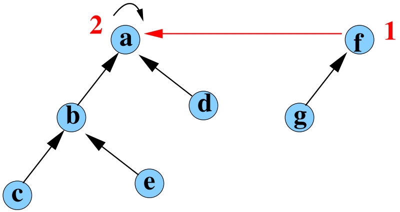


Figure: Union.

Union Techniques : Link-by-Rank (lr)

- Each set maintains a rank value, intially 0.
- Lowest ranked root ⇒ higher ranked root.
- Equal ranked roots ⇒ root of the combined tree is increased by 1.

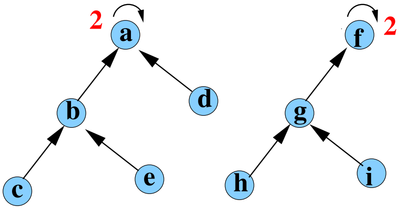


Figure: Union.

Union Techniques : Link-by-Rank (lr)

- Each set maintains a rank value, intially 0.
- Lowest ranked root ⇒ higher ranked root.
- Equal ranked roots ⇒ root of the combined tree is increased by 1.

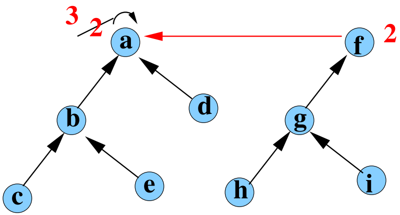


Figure: Union.

Compression Techniques

- Naive-Find (nf)
- Path-Compression (pc)
- Path-Splitting (ps)
- Path-Halving (ph)
- Type-0-Reversal (r0)
- Type-1-Reversal (r1)
- Collapsing (co)

Compression Techniques

- Reduce the height of the tree during the FIND operation.
- Subsequent FIND operations require less time.
- *Find-path* of a node x is the path of parent pointers from x upto the root of the tree.

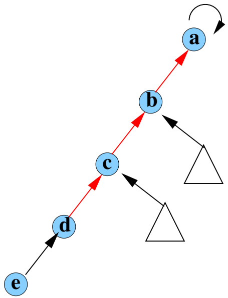


Figure: Find-path(d).

Compression Techniques : Path-Compression (PC)

- Set the parent pointers of all nodes in the find path to the root.
- Need to traverse the find-path twice.

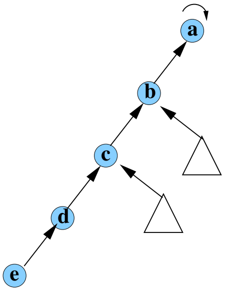


Figure: Find(e) with PC

Compression Techniques : Path-Compression (PC)

- Set the parent pointers of all nodes in the find path to the root.
- Need to traverse the find-path twice.

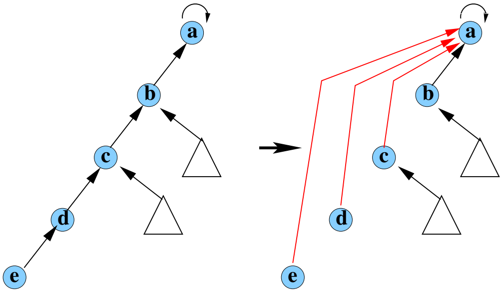


Figure: FIND(e) with PC

Compression Techniques : Path-Halving (PH)

- Set the parent pointers of every other nodes in the find-path to its grandparent.
- Traverse the find-path once.

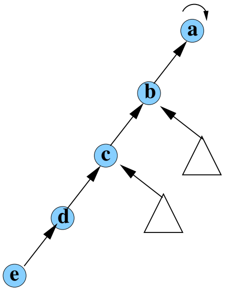


Figure: FIND(e) with PH

Compression Techniques : Path-Halving (PH)

- Set the parent pointers of every other nodes in the find-path to its grandparent.
- Traverse the find-path once.

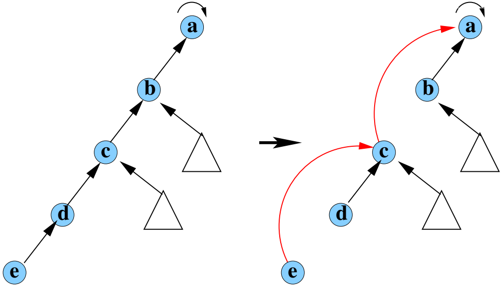


Figure: FIND(e) with PH

Compression Techniques : Collapsing (co)

- Every node points directly to the root.
- Find operation takes constant time.
- In a Union operation, all nodes of one tree point to the root of other tree.

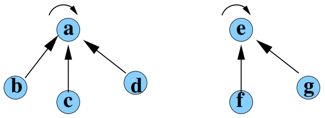


Figure: co

Compression Techniques : Collapsing (co)

- Every node points directly to the root.
- Find operation takes constant time.
- In a Union operation, all nodes of one tree point to the root of other tree.

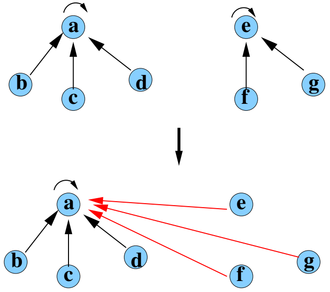


Figure: co

Worst Case COMPLEXITY

- For any combination of m MAKESET, UNION and FIND operations on n elements.

UNION	COMPRESSION	COMPLEXITY
NL	NF	$O(mn)$
NL	PC, PH, PS	$O(m \log_{(1+m/n)} n)$
NL	CO	$O(m + n^2)$
NL, LR, LS	R0, R1	$O(n + m \log n)$
LR, LS	CO	$O(m + n \log n)$
LR, LS	PC, PH, PS	$O(m \cdot \alpha(m, n))$

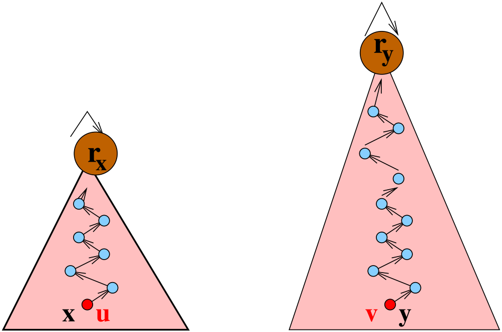
INTERLEAVED (INT) Algorithm

- During a UNION operation, the two FIND are performed as a single interleaved operation.
- The first INT algorithm is Rem's algorithm [Dijkstra, 1976].

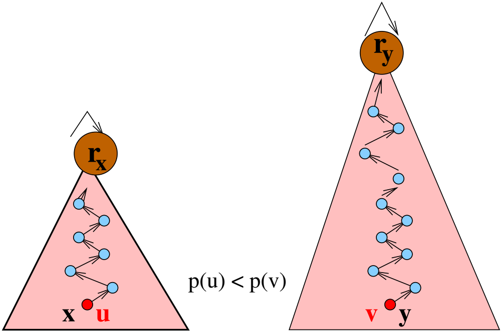
The Rem Algorithm [Dijkstra, 1976]

- Each node has a unique identifier $\Rightarrow$ index of the node.
- Lowered numbered node points to higher numbered node or to itself (if it is a root).

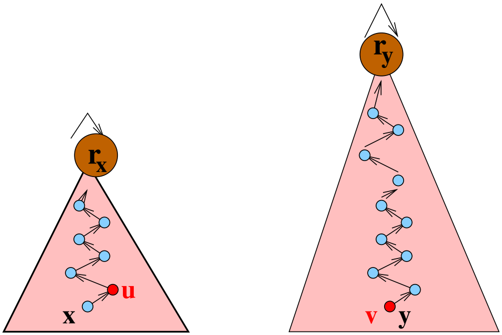
The Rem Algorithm: Example - Edge (x, y)



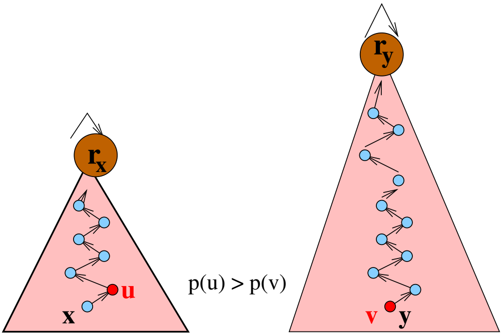
The Rem Algorithm: Different Sets - Compare ID of Parents



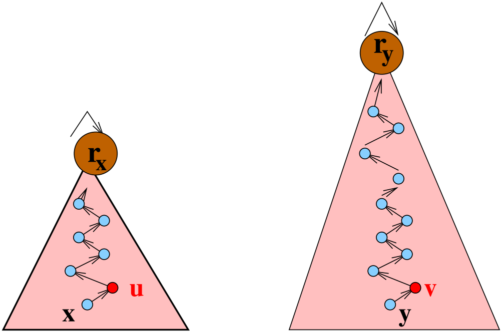
The Rem Algorithm: Different Sets - Compare ID of Parents



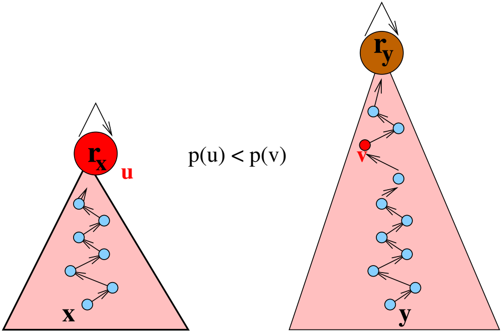
The Rem Algorithm: Different Sets - Compare ID of Parents



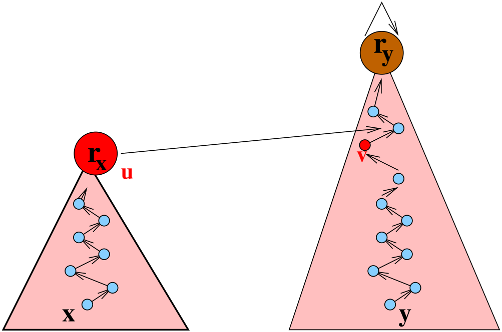
The Rem Algorithm: Different Sets - Compare ID of Parents



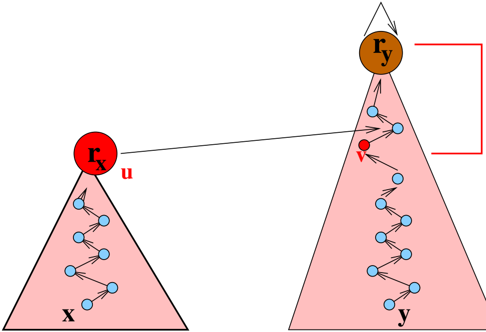
The Rem Algorithm: Different Sets - Compare ID of Parents



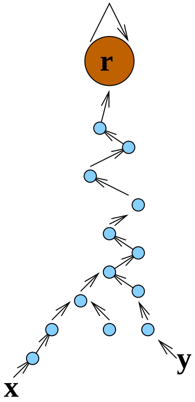
The Rem Algorithm: Different Sets - Compare ID of Parents



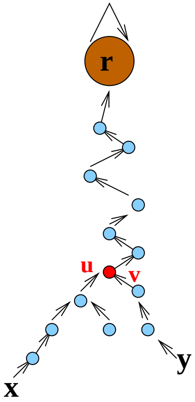
The Rem Algorithm: Different Sets - Compare ID of Parents



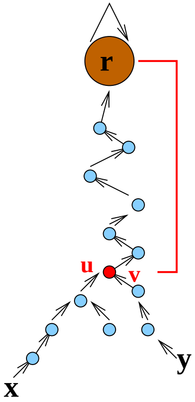
The Rem Algorithm: Same Set - Edge (x, y)



The Rem Algorithm: Same Set - Edge (x, y)



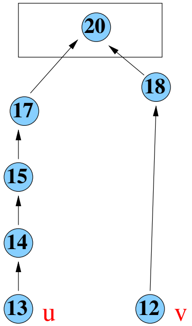
The Rem Algorithm: Same Set - Edge (x, y)



The Rem Algorithm: Compression

Splicing (sp)

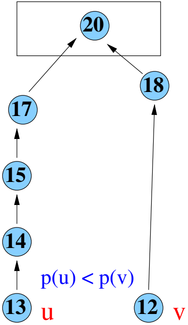
- Set parent pointer to a higher valued node ⇒ compressing the tree.
- Intuition: Higher valued node should be closer to the root.
- The running time of RemSP is $O(m \log_{(2+m/n)} n)$.



The Rem Algorithm: Compression

SPLICING (SP)

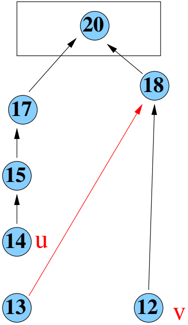
- Set parent pointer to a higher valued node ⇒ compressing the tree.
- Intuition: Higher valued node should be closer to the root.
- The running time of RemSP is $O(m \log_{(2+m/n)} n)$.



The Rem Algorithm: Compression

SPLICING (SP)

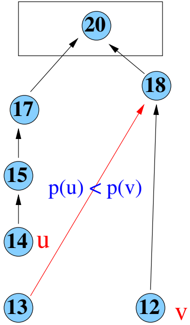
- Set parent pointer to a higher valued node ⇒ compressing the tree.
- Intuition: Higher valued node should be closer to the root.
- The running time of RemSP is $O(m \log_{(2+m/n)} n)$.



The Rem Algorithm: Compression

Splicing (sp)

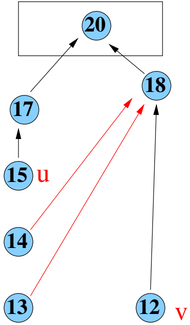
- Set parent pointer to a higher valued node ⇒ compressing the tree.
- Intuition: Higher valued node should be closer to the root.
- The running time of RemSP is $O(m \log_{(2+m/n)} n)$.



The Rem Algorithm: Compression

Splicing (sp)

- Set parent pointer to a higher valued node ⇒ compressing the tree.
- Intuition: Higher valued node should be closer to the root.
- The running time of RemSP is $O(m \log_{(2+m/n)} n)$.



A Variation of Rem: "TvL" [Tarjan and van Leeuwen, 1984]

- Uses ranks rather than identifier.
- This algorithms is slightly more complicated than Rem.

Test Sets and Experimental Setup

- Dell computer, Intel Core 2 CPU (2.40 GHz), Fedora 10, C++ and GCC (-O3).
- Three test sets.
  1. rw: 9 real world graphs.
     - Linear programming, Medical science.
     - Structural engineering, Civil engineering.
     - Automotive industry.
  2. sw: 5 synthetic small world graphs .
  3. er: 6 synthetic Erdös-Rényi random graphs.
- For each graph: 5 runs with 5 different random orderings of edges.

Structural Properties of the Input Graphs

Graph	$\|V\|$	$\|E\|$	Comp	Max Deg	Avg Deg	# Edges Processed
rw1 (m_t1)	97,578	4,827,996	1	236	99	692,208
rw2 (crankseg_2)	63,838	7,042,510	1	3,422	221	803,719
rw3 (inline_1)	503,712	18,156,315	1	842	72	5,526,149
rw4 (ldoor)	952,203	22,785,136	1	76	48	7,442,413
rw5 (af_shell10)	1,508,065	25,582,130	1	34	34	9,160,083
rw6 (boneS10)	914,898	27,276,762	1	80	60	11,393,426
rw7 (bone010)	986,703	35,339,811	2	80	72	35,339,811
rw8 (audikw_1)	943,695	38,354,076	1	344	81	10,816,880
rw9 (spal_004)	321,696	45,429,789	1	6,140	282	28,262,657
sw1	50,000	6,897,769	17,233	6,241	276	6,897,769
sw2	75,000	12,039,043	9,467	8,624	321	12,039,043
sw3	100,000	16,539,557	34,465	10,470	331	16,539,557
sw4	175,000	26,985,391	43,931	14,216	308	26,985,391
sw5	200,000	34,014,275	68,930	16,462	340	34,014,275
er1	100,000	453,803	24	25	9	453,803
er2	100,000	1,650,872	1	61	33	603,141
er3	500,000	2,904,660	8	30	12	2,904,660
er4	1,000,000	5,645,880	31	31	11	5,645,880
er5	500,000	9,468,353	1	70	38	3,476,740
er6	1,000,000	20,287,048	1	76	41	7,347,376

Relative performance of the classical algorithms

CL	Compression Technique							
UNION	NF	PC	PH	PS	CO	R0	R1	SP
NL								×
LR		❶	❷					×
LS								×
Rem			×		×	×	×	❸
TvL			×		×	×	×	

Table: 29 variations of classical algorithms. Each cell is an algorithm.

X dominates Y

- An algorithm X dominates another algorithm Y if X performs at least as well as Y.
- Since LRPC and LRPH are generally accepted as best, we begin by examining these.

Relative performance of the classical algorithms

CL	Compression Technique							
Union	NF	PC	PH	PS	CO	R0	R1	SP
NL								X
LR		❶	❷					X
LS								X
Rem			X		X	X	X	❸
TvL			X		X	X	X	

Table: 29 variations of classical algorithms.

Relative performance of the classical algorithms

CL	Compression Technique							
UNION	NF	PC	PH	PS	CO	R0	R1	SP
NL	LRPC_1				LRPC_1	LRPC_1	LRPC_1	X
LR	LRPC_1	❶	❷			LRPC_1	LRPC_1	X
LS	LRPC_1	LRPC_1				LRPC_1	LRPC_1	X
Rem	LRPC_1		X		X	X	X	❸
TvL	LRPC_1	LRPC_1	X		X	X	X	

Table: LRPC dominates 14 algorithms.

Relative performance of the classical algorithms

CL	Compression Technique							
UNION	NF	PC	PH	PS	CO	R0	R1	SP
NL	$LRPC_1$	$LRPH_2$			$LRPC_1$	$LRPC_1$	$LRPC_1$	X
LR	$LRPC_1$	❶ $LRPH_2$	❷			$LRPC_1$	$LRPC_1$	X
LS	$LRPC_1$	$LRPC_1$				$LRPC_1$	$LRPC_1$	X
Rem	$LRPC_1$		X		X	X	X	❸
TvL	$LRPC_1$	$LRPC_1$	X		X	X	X	$LRPH_2$

Table: LRPH dominates 3 additional, including LRPC - Total 17.

Relative performance of the classical algorithms

CL	Compression Technique							
UNION	NF	PC	PH	PS	CO	R0	R1	SP
NL	LRPC$_1$	LRPH$_2$	RemSP$_3$	RemSP$_3$	LRPC$_1$	LRPC$_1$	LRPC$_1$	X
LR	LRPC$_1$	❶ LRPH$_2$	❷ RemSP$_3$	RemSP$_3$	RemSP$_3$	LRPC$_1$	LRPC$_1$	X
LS	LRPC$_1$	LRPC$_1$	RemSP$_3$	RemSP$_3$	RemSP$_3$	LRPC$_1$	LRPC$_1$	X
Rem	LRPC$_1$	RemSP$_3$	X		X	X	X	❸
TvL	LRPC$_1$	LRPC$_1$	X	RemSP$_3$	X	X	X	LRPH$_2$

Table: RemSP dominates 10 of remaining, including LRPH - Total 27.

Relative performance of the classical algorithms

▸ Figure

CL	Compression Technique							
UNION	NF	PC	PH	PS	CO	R0	R1	SP
NL	$LRPC_1$	$LRPH_2$	$RemSP_3$	$RemSP_3$	$LRPC_1$	$LRPC_1$	$LRPC_1$	X
LR	$LRPC_1$	❶ $LRPH_2$	❷ $RemSP_3$	$RemSP_3$	$RemSP_3$	$LRPC_1$	$LRPC_1$	X
LS	$LRPC_1$	$LRPC_1$	$RemSP_3$	$RemSP_3$	$RemSP_3$	$LRPC_1$	$LRPC_1$	X
Rem	$LRPC_1$	$RemSP_3$	X	**undom.**	X	X	X	❸ **undom.**
TvL	$LRPC_1$	$LRPC_1$	X	$RemSP_3$	X	X	X	$LRPH_2$

Table: Only 2 algorithms are **undominated**.

Implementation Enhancements

- Ways to make the classical algorithms faster.
- Enhancements:
  1. Immediate parent check [Osipov et al., 2009].
  2. Better interleaved algorithms [Manne and Patwary, 2009].
  3. Memory efficient algorithms, Reduce memory usage.

Enhancement 1: Immediate Parent Check (IPC) [Osipov et al., 2009]

- IPC applies to any classical algorithm (Rem already implements IPC).
- {IPC} X {LR, LS} X {PC, PH, PS}
- {IPC} X {TvL} X {PC, PS, SP}
- 9 more variations.
- RemSP dominates all 9 variations.

Enhancement 2: Better Interleaved Algorithms (INT) [Manne and Patwary, 2009]

- Better interleaved algorithms than TvL ⇒ eTvL, ZZ.
- {eTvL} X {PC, PS, SP}
- {ZZ} X {PC, PS}
- 5 more variations.
- RemSP dominates all 5 variations.

Enhancement 3: Memory Efficient (MS) Algorithms

- Reduce memory used by each algorithm.
- {MS} X {NL} X {PC, PS}.
- {MS} X {LR, LS} X {PC, PS, CO}.
- {MS} X {IPC} X {LR, LS} X {PC, PS}.
- 12 more variations.
- Note that Rem does not use size or rank, so it is automatically an MS algorithm.

Enhancement 3: MS relative performance

MS-UNION Method	Compression Technique			
	PC	PS	CO	SP
NL			X	X
LR				X
LS				X
IPC-LR			X	X
IPC-LS			X	X
Rem	RemSP$_3$ ≡	**undom.** ≡	X	❸ **undom.** ≡

Table: 12 more variations. Shaded row has already been considered.

Enhancement 3: MS relative performance

MS-UNION Method	Compression Technique			
	PC	PS	CO	SP
NL	RemSP_3	RemSP_3	X	X
LR	$\text{RemSP}_3\uparrow$			X
LS	$\text{RemSP}_3\uparrow$			X
IPC-LR			X	X
IPC-LS	RemSP_3	RemSP_3	X	X
Rem	$\text{RemSP}_3 \equiv$	**undom.** $\equiv$	X	❸ **undom.** $\equiv$

Table: RemSP dominates 6 algorithms - Total 47 of 55.

Enhancement 3: MS relative performance ▸ Figure

MS-UNION Method	Compression Technique			
	PC	PS	CO	SP
NL	RemSP_3	RemSP_3	×	×
LR	$\text{RemSP}_3 \uparrow$	**undom.** $\uparrow$	**undom.** $\uparrow$	×
LS	$\text{RemSP}_3 \uparrow$	**undom.** $\uparrow$	**undom.** $\uparrow$	×
IPC-LR	**undom.** $\uparrow$	**undom.**	×	×
IPC-LS	RemSP_3	RemSP_3	×	×
Rem	$\text{RemSP}_3 \equiv$	**undom.** $\equiv$	×	❸ **undom.** $\equiv$

Table: 6 algorithms are undominated - Total 8 undominated of 55.

The Fastest Algorithms

- Different metric than the dominates technique.
- Fictious algorithm (GLOBAL-MIN) ⇒ run-time equal to the best of any algorithm for each graph.
- For each algorithm
  1. Compute average relative time for each graph.
  2. Compute average relative time for each type of graph (rw, sw and er).
  3. Compute average of the three types ⇒ final average.
- Rank the order of the algorithms based on final averages.

Rank order of the fastest algorithms ‣ Figure

Algorithm	Rank based on graphs of type			
	All graphs	Real-World	Small-World	Erdős-Rényi
RemSP	1	1	1	1
RemPS	2	5	2	4
MS-IPC-LRPC	3	6	3	7
MS-LSPS	4	2	10	2
MS-IPC-LSPC	5	8	4	8
MS-LRPS	6	4	13	3
MS-IPC-LRPS	7	3	11	5
MS-LSCO	8	9	6	9
MS-LRCO	9	10	5	10
MS-IPC-LSPS	10	7	15	6

The Fastest Algorithms: Observations

- All the top 10 algorithms use MS enhancement.
- 8 out of top 10 are one pass algorithms.
- Out of the top 5 algorithms, 2 uses PC.
- LRPC, LRPH are not in top 10.

1. RemSP
2. RemPS
3. MS-IPC-LRPC
4. MS-LSPS
5. MS-IPC-LSPC
6. MS-LRPS
7. MS-IPC-LRPS
8. MS-LSCO
9. MS-LRCO
10. MS-IPC-LSPS

Related Experimental Studies: Improvement

Reference	# of Algorithms	Recommended Algorithm	RemSP improves by
[Liu, 1990]	2	NLPC	56%
[Gilbert et al., 1994]	6	NLPH	45%
[Wassenberg et al., 2008]	8	LRCO	24%
[Wu et al., 2009]	3	LIPC	48%
[Hynes, 1998]	18	LICO, LSCO	28%, 24%
[Osipov et al., 2009]	2	IPC-LRPC	29%
-	-	LRPC	52%
-	-	LRPH	28%

Future Works

- Extend to other application areas.
- Consider arbitrary sequences of intermixed MAKESET, UNION, and FIND operations.
- More formal profiling including cache misses, pointer jumps, number of comparisons etc.

Thank you.

Best Enhanced Classical Algorithm

- Best enhanced classical algorithm is MS-IPC-LRPC.
- RemSP improved over MS-IPC-LRPC : 12% (-3%-18%)

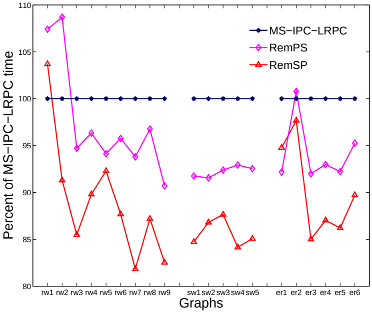


Figure: Improvement over MS-IPC-LRPC.

How much improvement

- RemSP substantially outperforms LRPC even though theoritically inferior.
- LRPC $\Rightarrow$ real world, algorithm courses, libraries.
- RemSP improved over LRPC: 52% (38%-66%)
- RemSP improved over LRPH: 28% (15%-45%)

Bibliography I

Dijkstra, E. W. (1976).
A Discipline of Programming.
Prentice-Hall.

Gilbert, J. R., Ng, E. G., and Peyton, B. W. (1994).
An efficient algorithm to compute row and column counts for sparse Cholesky factorization.
SIAM J. Matrix Anal. Appl., 15(4):1075–1091.

Hynes, R. (1998).
A new class of set union algorithms.
Master's thesis, Department of Computer Science, University of Toronto, Canada.

Bibliography II

Liu, J. W. H. (1990).
The role of elimination trees in sparse factorization.
SIAM J. Matrix Anal. Appl., 11:134–172.

Manne, F. and Patwary, M. M. A. (2009).
A scalable parallel union-find algorithm for distributed memory computers.
To appear in the proceedings of PPAM 2009, LNCS.

Osipov, V., Sanders, P., and Singler, J. (2009).
The filter-Kruskal minimum spanning tree algorithm.
In *ALENEX*, pages 52–61.

Bibliography III

Tarjan, R. E. and van Leeuwen, J. (1984).
Worst-case analysis of set union algorithms.
J. ACM, 31(2):245–281.

Wassenberg, J., Bulatov, D., Middelmann, W., and Sanders, P. (2008).
Determination of maximally stable extremal regions in large images.
In *From Proceeding of Signal Processing, Pattern Recognition, and Applications (SPPRA)*. Acta Press.

Bibliography IV

Wu, K., Otoo, E., and Suzuki, K. (2009).
Optimizing two-pass connected-component labeling algorithms.
Pattern Anal. Appl., 12(2):117–135.