Incremental Timing-Driven Placement

Steve Haynal and Forrest Brewer

Department of Electrical and Computer Engineering
University of California, Santa Barbara

January 1997

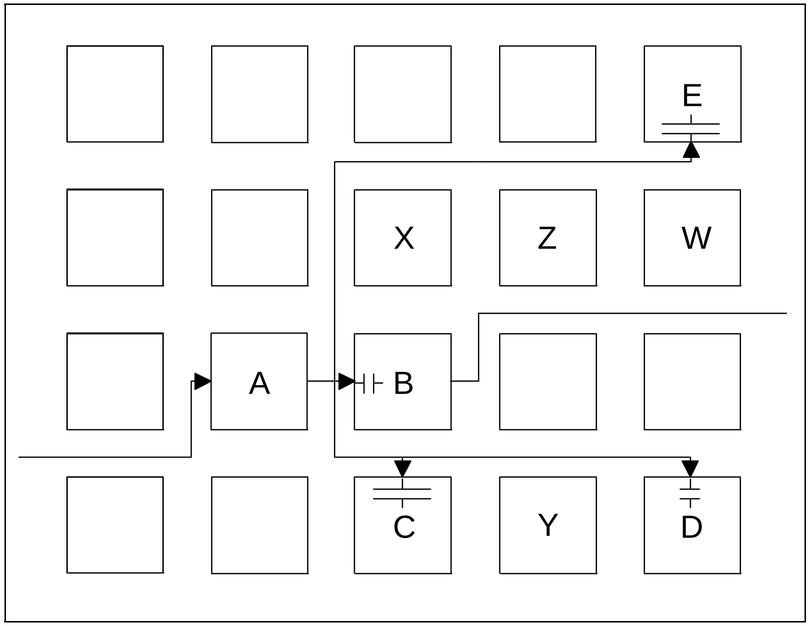
Motivations

- Class project
- Practical project
- Experience with Epoch
- Automate steps used to fix critical delay in Rogue Chip
- Incremental algorithms:
  - Not as many
  - Provides manual intervention
  - Fast
  - Can be complex
  - More focused view

Outline

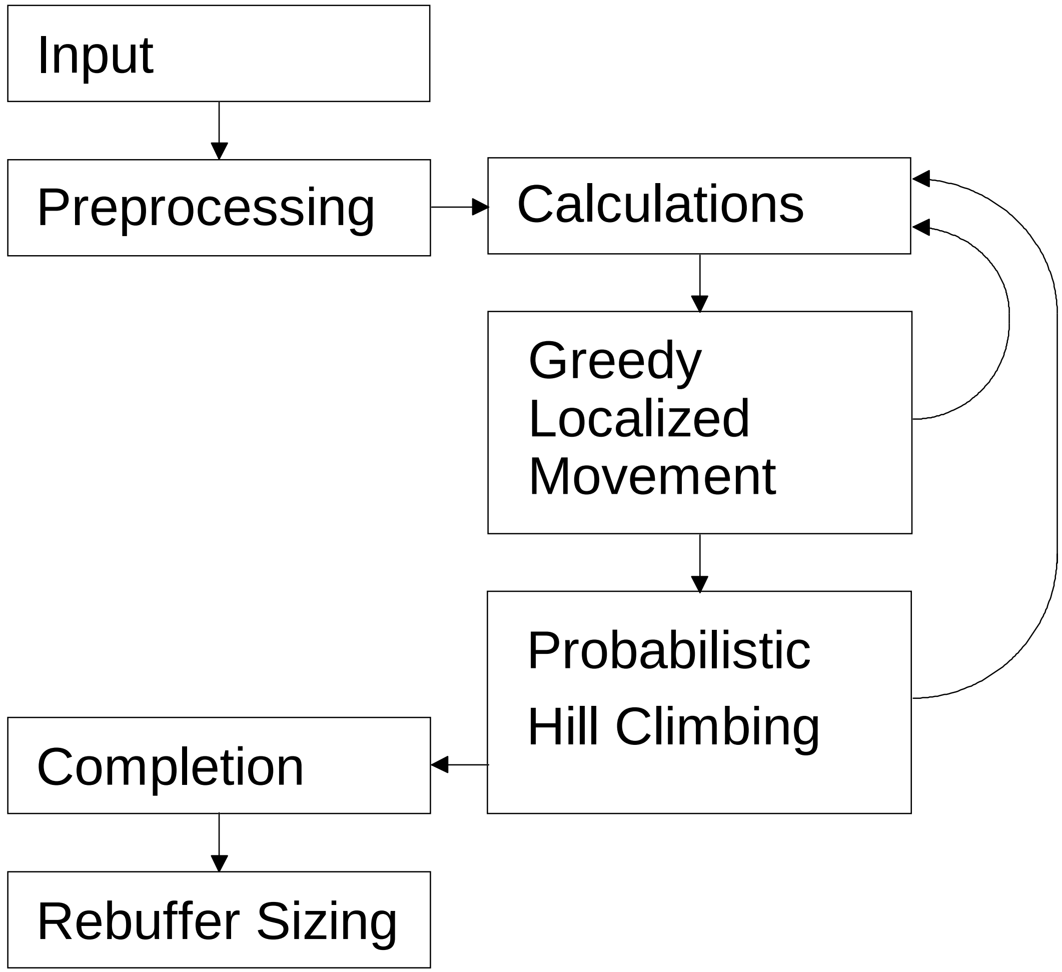
- Underlying Principles
- Algorithm Description
- Implementing and Testing the Algorithm
- Observations and Conclusions

Underlying Principles



- Cluster Critical Cells - First for a critical net, then for a critical path.
- Reduce Gate Sizes - Even of periphery nodes
- Localize Movement - Spread out impact costs
- Probabilistic Hill Climbing - Overcomes local minimums
- Good Delay Estimates - Costly but applicable for incremental changes
- Stop When Goal is Reached - Avoids detrimental changes

Seven Step Algorithm

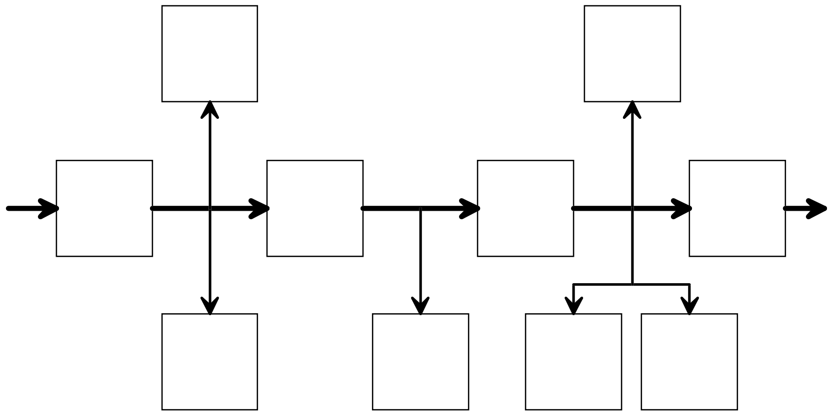


Step 1: Input

- Connectivity and placement data - movable cells and fixed points
- Original buffer sizes
- Penfield-Horowitz timing data
- Critical and near critical paths
- Desired critical path delay reduction
- Delay due to wiring

Step 2: Preprocessing

- Critical paths
- Critical nets
- Critical cells
- Near critical nets
- Near critical cells



Step 3: Calculations

- Net mass center $x_{net} = \frac{1}{n}\sum_{i=1}^{n} x_{cell_i}$ $y_{net} = \frac{1}{n}\sum_{i=1}^{n} y_{cell_i}$ (n=critical cells in net)
- Ideal cell locations $x_{ideal} = \frac{\sum_{j=1}^{m}\left(w_j \times x_{net_j}\right)}{\sum_{j=1}^{m} w_j}$ $y_{ideal} = \frac{\sum_{j=1}^{m}\left(w_j \times y_{net_j}\right)}{\sum_{j=1}^{m} w_j}$ (m=critcal nets to cell)
- Path cost $pathc = \sum_{j=1}^{n}\left(w_j \times \beta_j\right)$ (n=critical nets in path)
- Global cost $globalc = \sum_{j=1}^{m}\left(w_j \times \beta_j\right)$ (m=critical nets in circuit)
  - Initial path goal cost $goalc = \frac{d_{wiring} - d_{reduce}}{d_{wiring}} \times pathc_{initial}$.

Step 4: Greedy Localized Movement

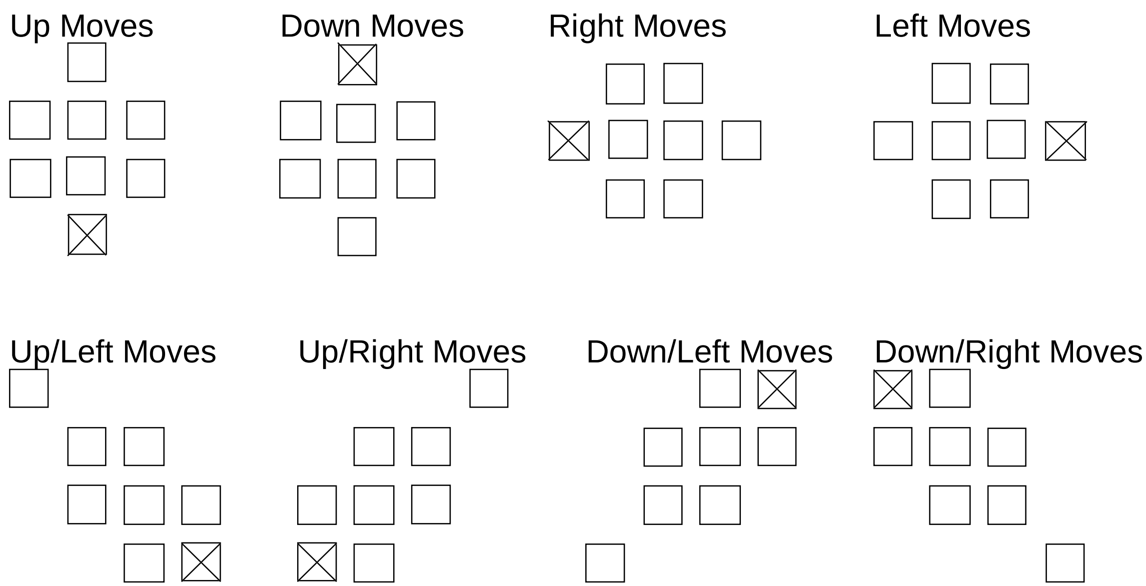


Figure 3: Local move sets. Critical cell to move is marked with an ×.

Step 4: Greedy Localized Movement

- **Types of moves**
  - Jam moves
  - Swap moves

- **Accept move when** $accept = \sum_{i=1}^{n}\left(w_i \times \left(\beta_{old_i} - \beta_{new_i}\right)\right)$ **is maximized**

- **Exceptions to step 4:**
  - Cell is in an ideal location
  - No move is possible

  - Move to critical path mass center $x_{cp} = \dfrac{\sum_{j=1}^{m}\left(w_j \times x_{net_j}\right)}{\sum_{j=1}^{m} w_j}$

Step 5: Probababilistic Hill Climbing

- Kernighan-Lin style backtracking to lowest cost point in move sequence
  - If n moves were made in step 4, n-(k+1) of these moves are undone so that $\sum_{i=1}^{k} accept$ is maximized and greater than or equal to zero.
- The new global cost is set to $g_{new} = g_{old} - \sum_{i=1}^{k} accept$

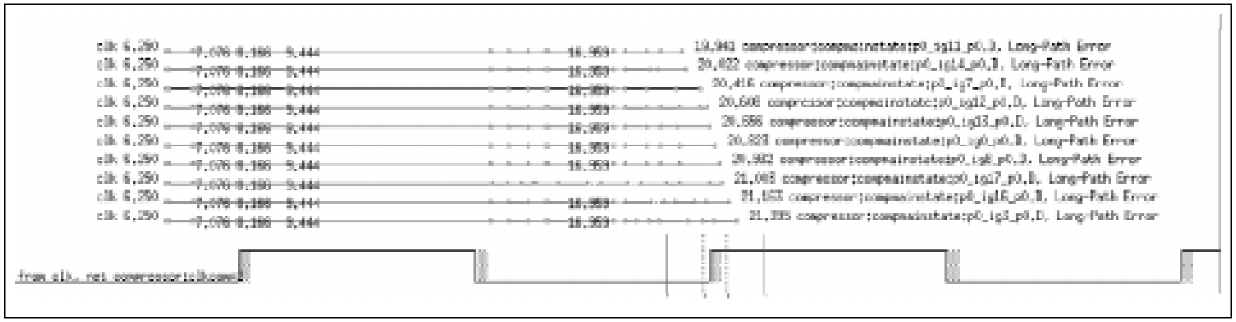
Step 6: Completion.

- Random reordering of lists:
  - A net's critical cells
  - A path's critical nets
  - Critical paths
- Exit when all goal costs met or no change in global cost

Step 7: Rebuffer Sizing

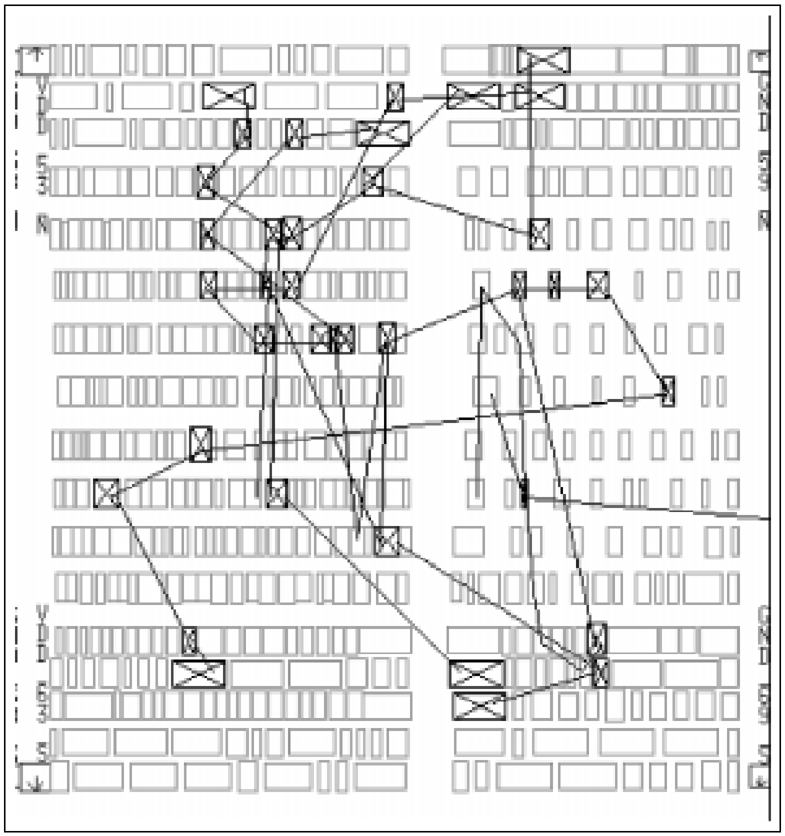
- All critical cells set to minimum size
- Timing-driven buffer sizing where needed

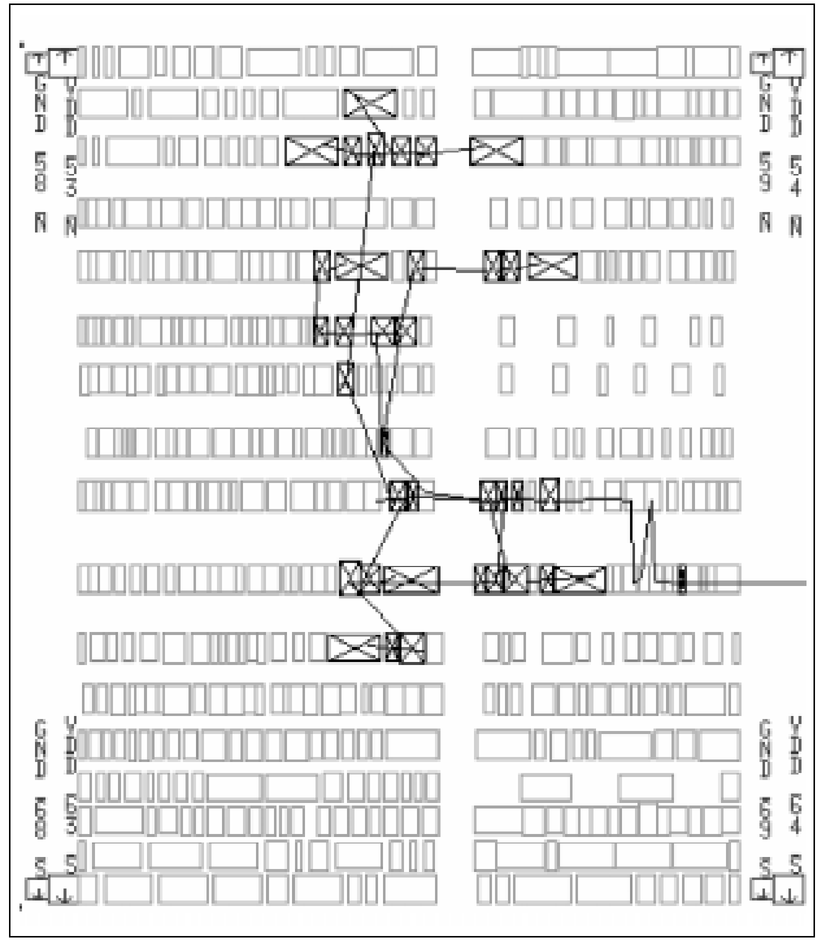
Implementing and Testing the Algorithm



- Roughly 2200 lines of C++
- Interface with Cascade's Epoch
- 512 cell HP26G 0.8 μm technology standard cell group test circuit
- 12.5 ns target clock period
- All Epoch's optimization options employed
- Worst critical path 900 ps too long

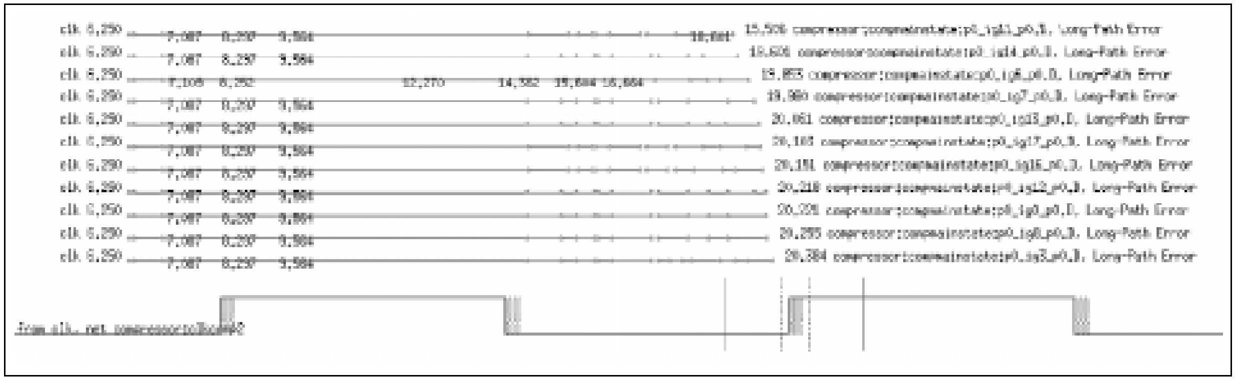
Maximum Net Reduction Trial





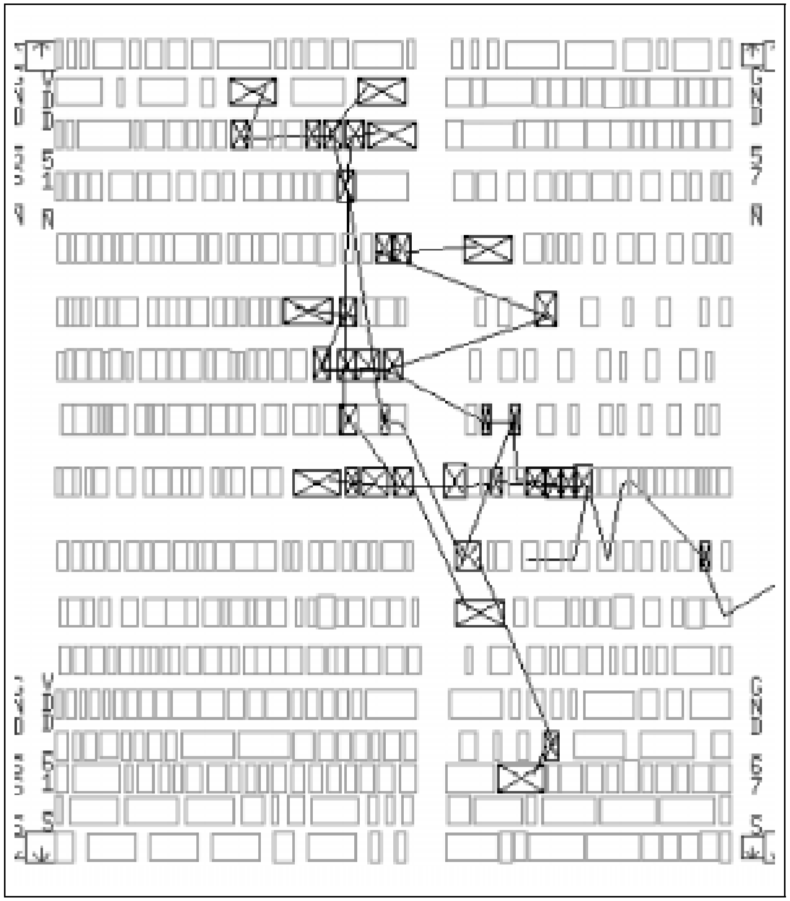
Before: $1.94\times10^{6}\ \mu m^{2}$ 9 critical paths **After** - $2.06\times10^{6}\ \mu m^{2}$ 1 critical path

Appropriate Net Reduction Results



- Appropriate d_{reduce} for each critical path
- 9905 moves
- 14.5 seconds Ultra Sparc 1 user time
- 2.02×10^6 μm^2 - 4% increase
- 12.5 ns cycle time - 11% speedup
- Worst critical path reduced by 1 ns
- Remaining near critical paths more balanced

Appropriate Net Reduction Trial



- Clumping of critical cells
- 4% area increase
- 11% speedup

Observations and Conclusions

- **Observations**
  - Timing analysis in the main loop
  - K-for-one swap moves
  - Compare with benchmarks
  - Compare with standard placement algorithms

- **Conclusions**
  - Exploitable freedoms in Epoch’s optimized standard cell placements
    - 11% speedup and 4% area increase in 512 standard cell test circuit
  - Approximately $O(n^3)$ complexity where n is number of critical cells
    - Fast - only handful of critical cells