Design and Implementation of the Fiduccia-Mattheyses Heuristic for VLSI Netlist Partitioning

Andrew E. Caldwell, Andrew B. Kahng, Igor L. Markov

UCLA Computer Science Department

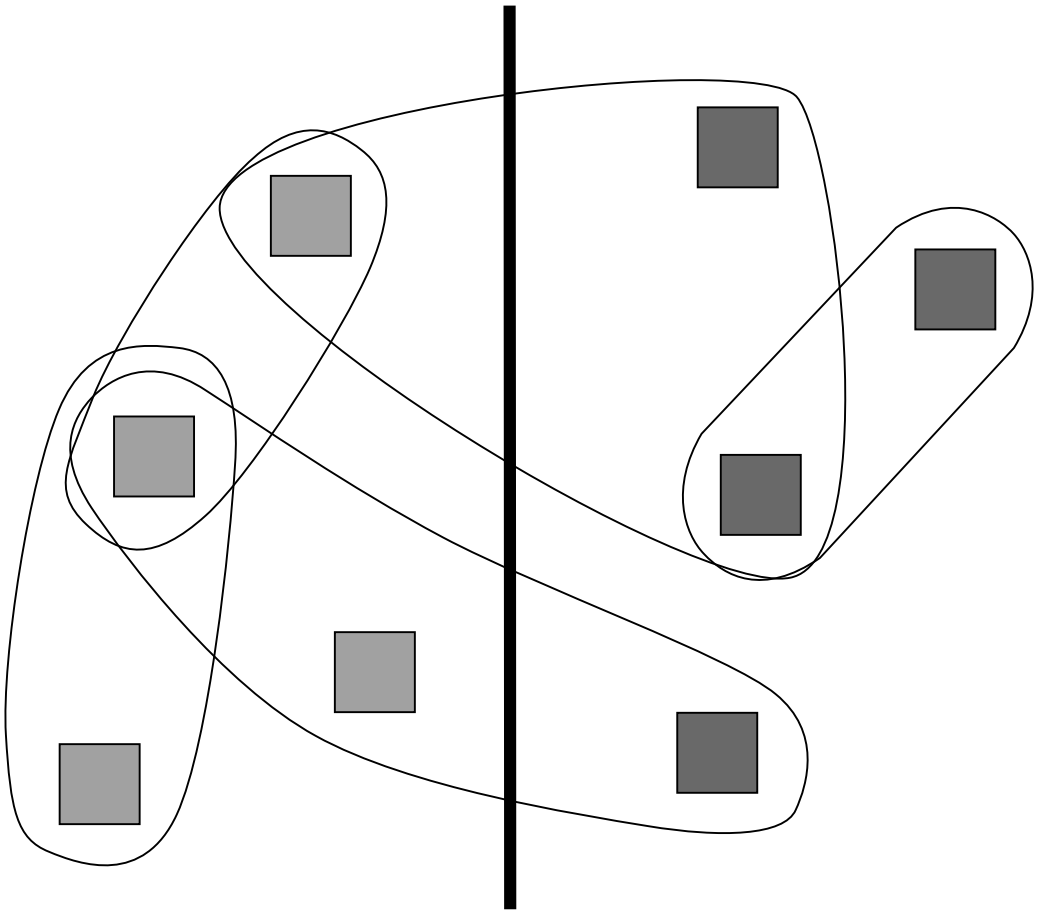
{caldwell,abk,imarkov}@cs.ucla.edu

http://vlsicad.cs.ucla.edu/

Overview

- Complex discrete optimization heuristics
  - increasingly sophisticated algorithms
  - growing variety of applications and contexts
  - implementation issues may dominate algorithmic improvements
- Component-based architecture
  - reduces complexity and facilitates flexibility
  - instrumental in evaluating algorithms and improvements
  - essential to achieving leading edge results
- Example: hypergraph partitioning in VLSI context

Balanced Hypergraph Partitioning



Cut = 2

Outline of Talk

- Hypergraph partitioning in VLSI design
- The Fiduccia-Mattheyses (FM) approach
- Flexible, efficient component-based design
- Benefits of a partitioning testbench

VLSI Partitioning Context

- Top-down placement (“divide and conquer”)
- Instances
  - directed/undirected hypergraphs
  - weighted/unweighted vertices, hyperedges
  - real IC designs (randomized benchmarks not accepted)
- Characteristics
  - up to 1,000,000 vertices, all sizes equally important
  - very uneven cell areas
  - sparse (avg vertex degree 3-5, avg hyperedge degree 3-5)
  - several very large hyperedges (clock, reset, …)

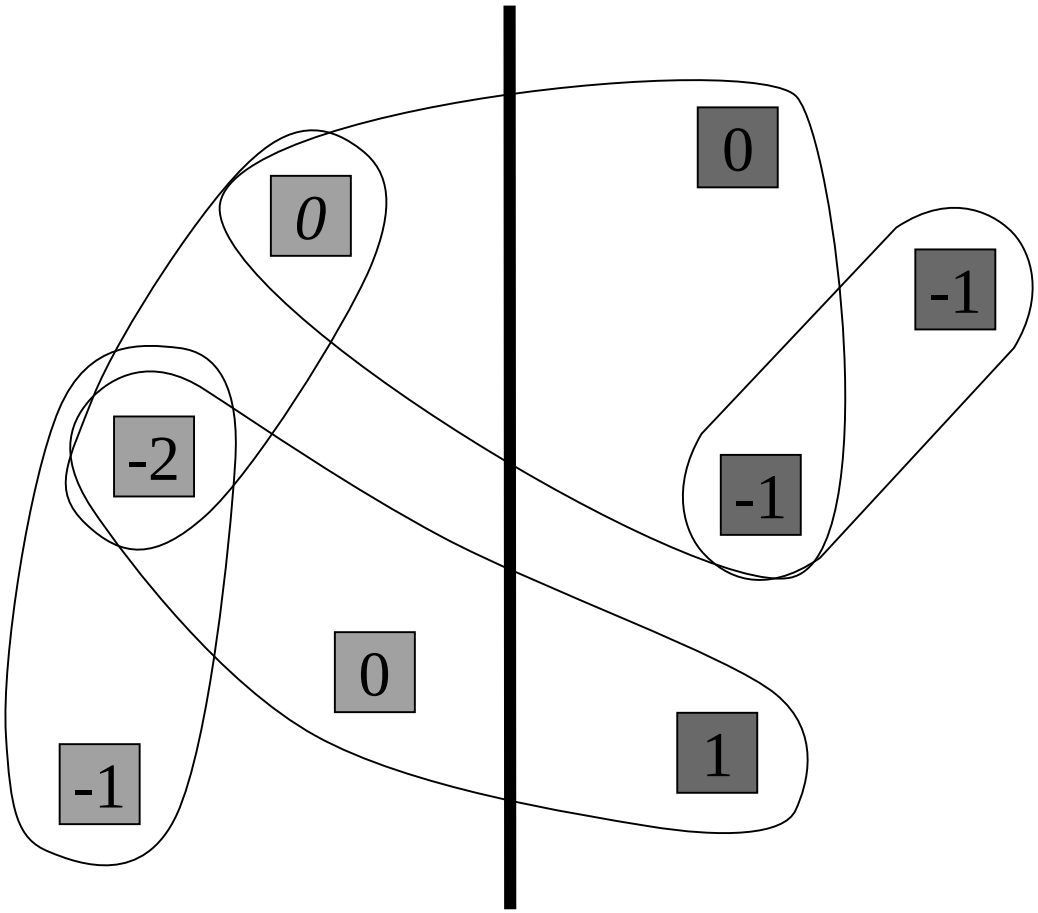
VLSI Partitioning Context (cont.)

- Speed: 6,000 cells/minute to final placement
  - implied partitioning runtime: 2 seconds for 25,000 cells, 1 minute for 750,000 cells
- Objectives: net cut, sum-of-degrees, ratio cut, …
- Constraints
  - tight partition area balance requirement
  - fixed vertices (I/O pads, terminals, etc.)
- Well-studied problem, many approaches attempted
  - spectral, network flow, metaheuristic, …
- Efficiency, flexibility: KLFM style preferred

Fiduccia-Mattheyses Heuristic

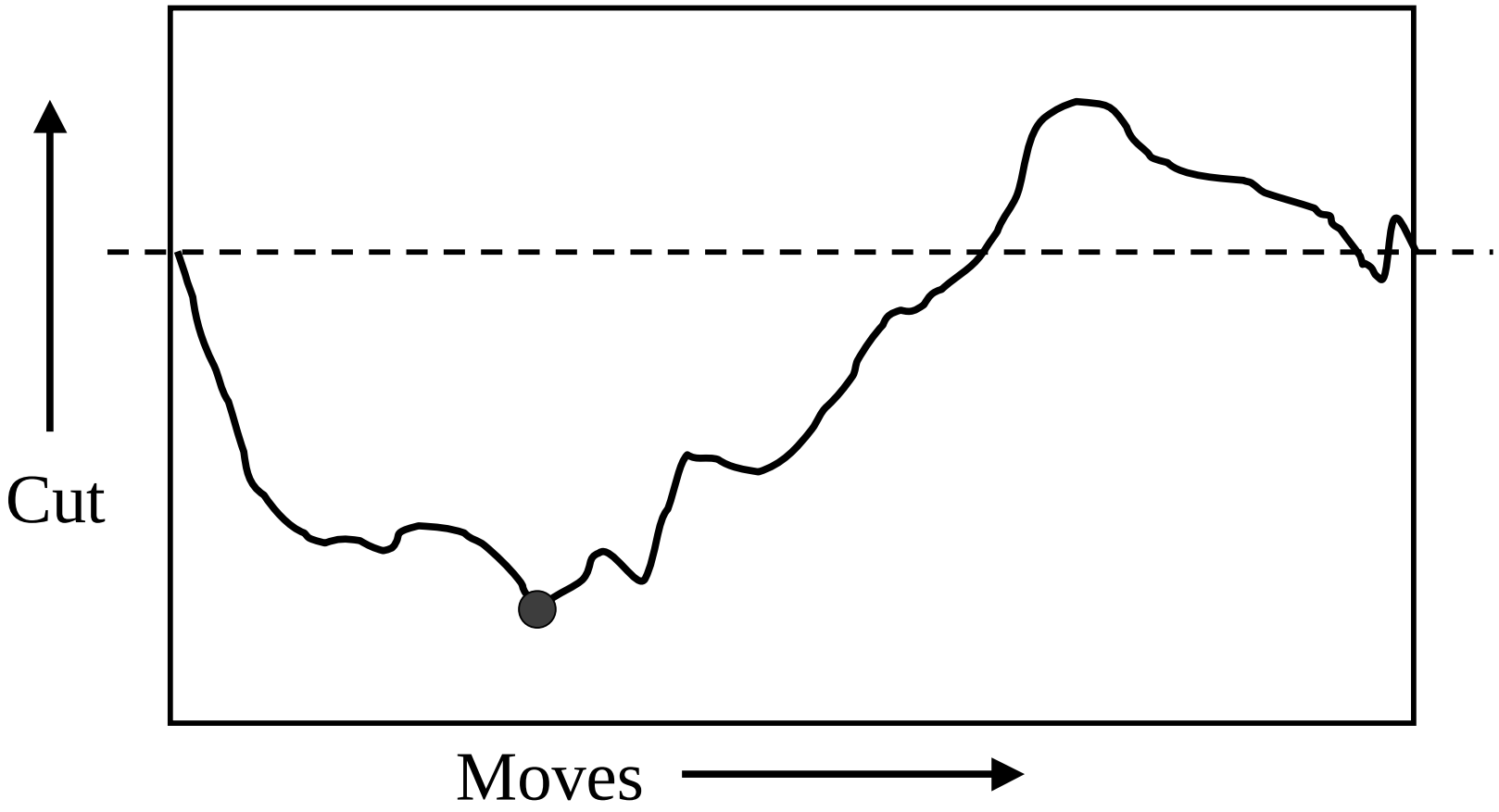
- **Pass:**
  - start with all vertices free to move (*unlocked*)
  - label each move with immediate change in cost that it causes (*gain*)
  - iteratively:
    - select and execute a move with highest gain
    - *lock* the moving vertex (i.e., cannot move again during the pass)
    - update affected gains (adjacent unlocked vertices)
  - best solution seen during pass is adopted as starting solution for next pass
- **FM:**
  - start with some initial solution
  - perform passes until a pass fails to improve solution quality

Balanced Hypergraph Partitioning



Cut = 2

Cut During an FM Pass



Key Elements of FM

- Three main operations
  - computation of initial gain values at beginning of pass
  - retrieval of the best-gain (feasible) move
  - update of all affected gain values after a move is made

- Observations of Fiduccia and Mattheyses:
  - circuit hypergraphs are sparse
  - move gain is bounded by +/- 2 * max vertex degree
  - can hash moves by gains (*gain bucket* structure)
  - each affected gain is updated in constant time
  - giving linear time complexity per pass

Motivations for This Work

- FM: a classic heuristic for a classic problem
  - basis for leading-edge multilevel heuristics and industry placement tools
  - dozens of “improvements” reported in literature since 1982
- But are we confident of “improvements” ?
  - implementation descriptions often inadequate
  - small implementation differences can have a large effect
  - comparisons are difficult at best
  - large and unnecessary “barrier to entry” in research
- FM literature is representative of other literatures

Example Implementation Differences

- Comparison of two ‘98 (LIFO) FM implementations
- Min and Ave cutsizes from 100 single-start trials

Tolerance	LIFO-FM		Ibm01	Ibm02	Ibm03	Ibm04	Ibm05	Ibm06
2%	Paper1	Min	450	648	2459	3201	2397	1436
		Ave	**2701**	**12253**	**16944**	**20281**	**3420**	**16578**
	Paper2	Min	366	301	1588	1014	2640	1008
		Ave	**594**	**542**	**2688**	**1802**	**3382**	**1746**
10%	Paper1	Min	270	313	1624	544	1874	1479
		Ave	**486**	**3872**	**12348**	**2383**	**3063**	**14007**
	Paper2	Min	244	266	1057	561	2347	821
		Ave	**445**	**405**	**1993**	**1290**	**3222**	**1640**

Canonical Partitioning Testbench Needed

- Flexibility - a single implementation supports
  - different objectives and constraints
  - different formulations
    - multi-way, with replication, path timing driven
- Necessary to verify improvements
  - what improves weak algorithms may worsen strong ones
  - fair comparison against competitive implementations

Component-Based Partitioner

- Initial Solution Generator
- Incremental Cost Evaluator
  - represents objective function
- Legality Checker
  - represents feasibility constraints
- Gain Container
  - represents neighborhood structure
- Move Manager
  - selects and executes best feasible move
- Pass-Based Partitioner (proper)

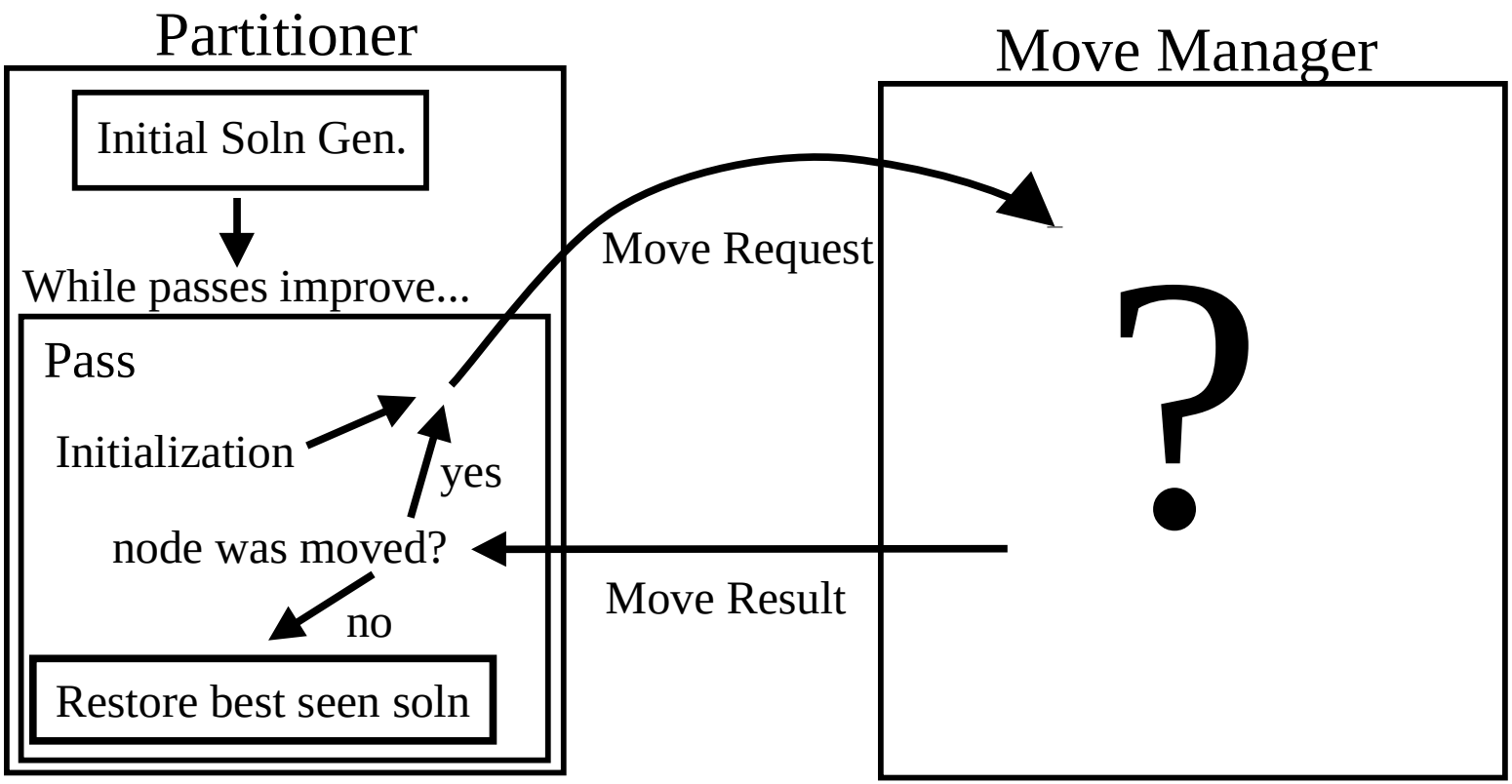
Components (cont.)

- Incremental Cost Evaluator
  - evaluates the cost of a partitioning
  - dynamically maintains costs as moves are applied
- Gain Container
  - general container for available moves
  - moves can be retrieved by, e.g. vertex id and destination partition
  - prioritizes moves by gains
    - O(1) retrieval of the highest gain move
  - supports O(1) gain updates

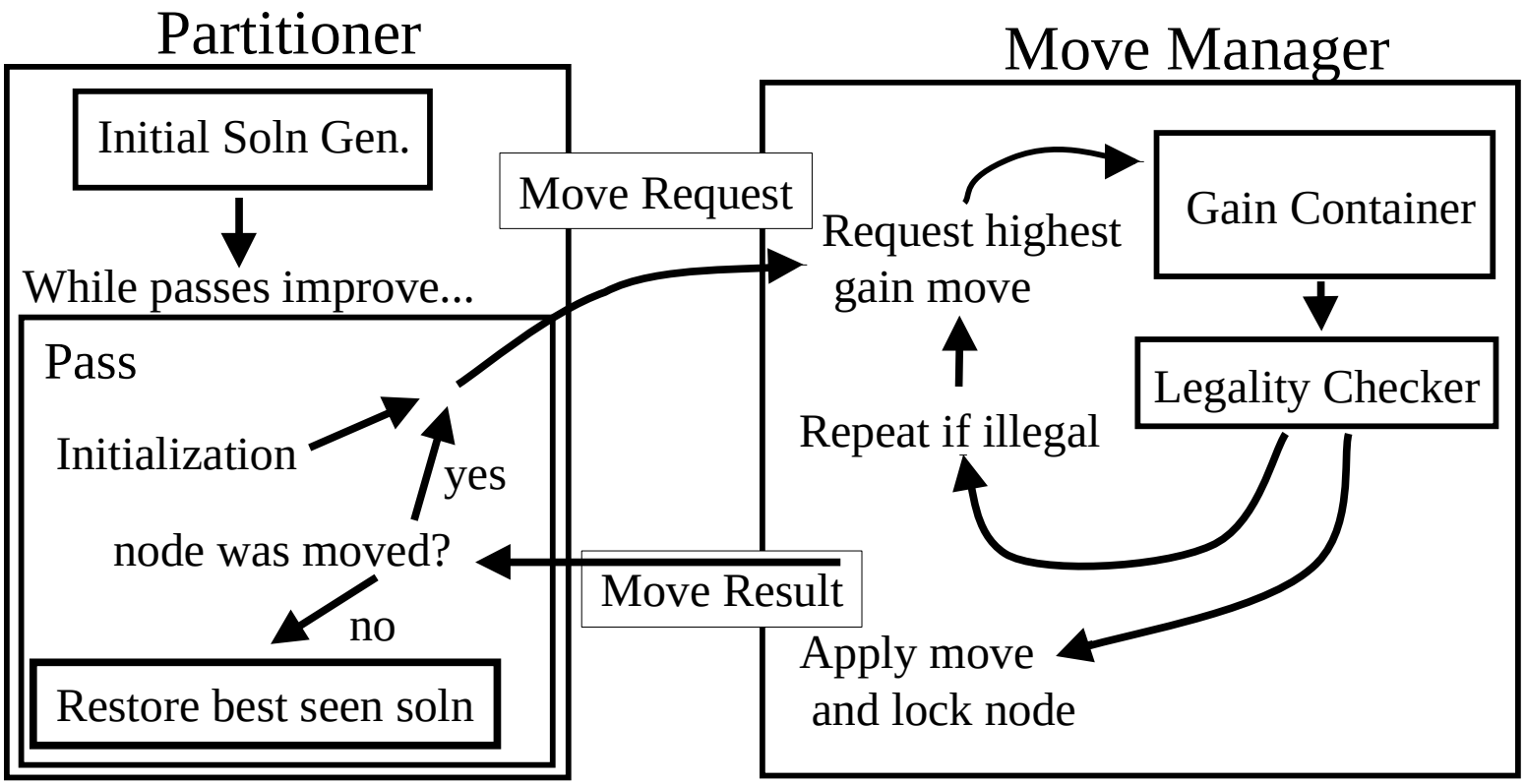
Components (cont.)

- Move Manager
  - contains
    - Legality Checker (one or more)
    - Incremental Cost Evaluator
  - populates a Gain Container with moves and initial gains
  - requests the highest gain move from Gain Container
  - notifies Legality Checker and Cost Evaluator of moves
- Pass-Based Partitioner Proper
  - implements a common partitioner interface
  - contains
    - Initial Solution Generator
    - Move Manager (one or more)

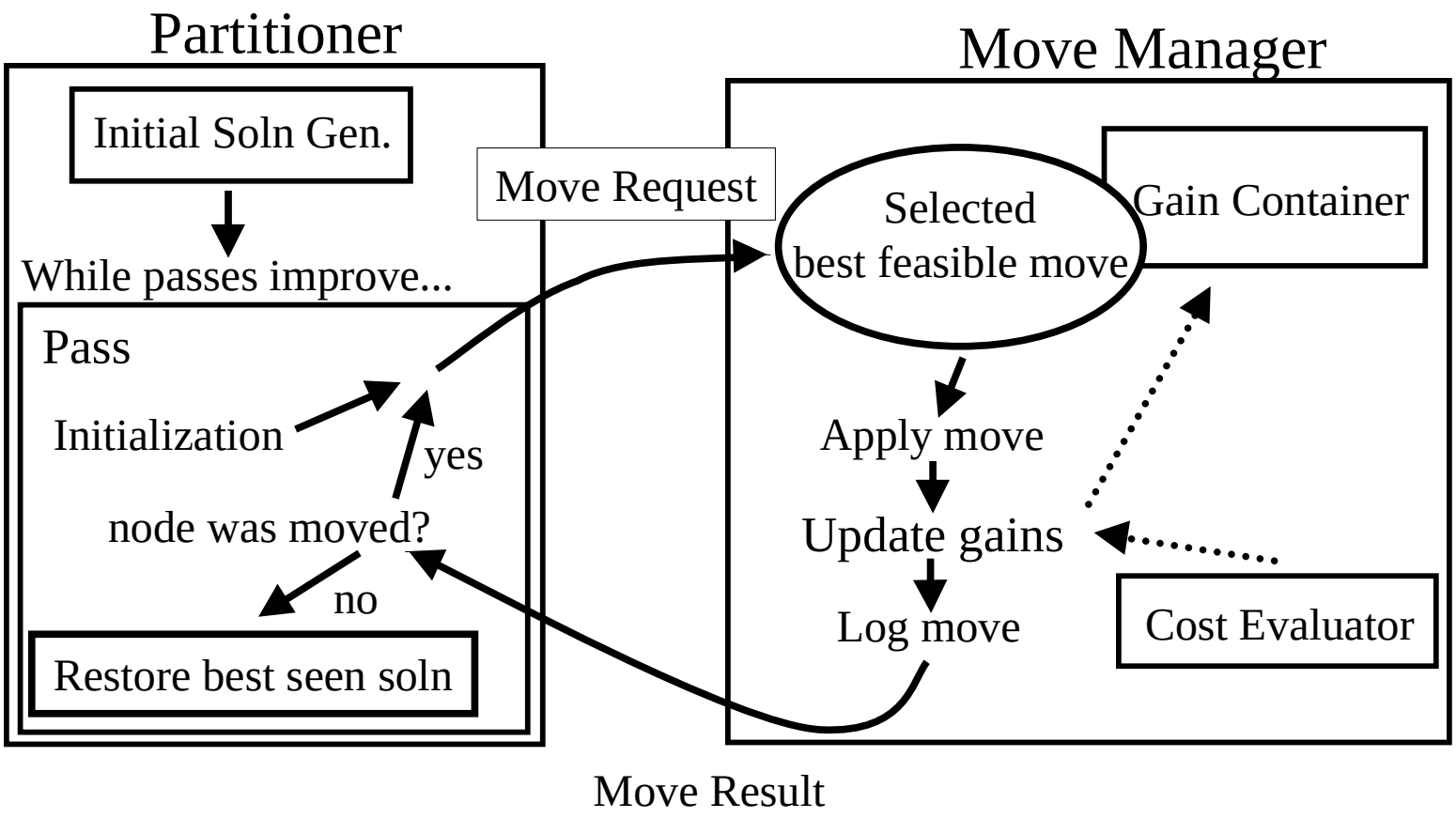
Component-Based Partitioner



Component-Based Partitioner



Component-Based Partitioner



Our Use of the Canonical Testbench

- Study of tie-breaking decisions and their effects
  - stunning differences in solution quality due to “implicit”, i.e., underspecified, implementation decisions
  - 400% solution quality differences far outweigh claimed improvements due to heuristic innovations
- Testbench consistent with leading-edge partitioning
- Affords flexibility and rapid exploration
  - e.g. partition variety of cost functions with no other change

Multilevel Partitioner Results

- Average cut over 100 runs
- Average CPU seconds per run given in parentheses
- Results are competitive with other leading-edge partitioners

Test Case	2% unit area	10% unit area	2% actual area	10% actual area
primary1	57.1(0.418)	52.1(0.412)	61.3(0.389)	54.1(0.421)
primary2	164.7(1.3)	143.6(1.3)	193.1(1.41)	177.8(1.07)
biomed	132.8(2.24)	117.0(2.16)	137.2(2.07)	125.0(2.23)
ibm01	275.0(5.7)	247.9(5.82)	284.5(6.63)	244.4(5.36)
ibm02	372.1(10.6)	268.8(10.3)	421.3(14.6)	275.7(14.7)
ibm03	1031.4(11.6)	820.9(11.8)	1079.8(15.8)	1062.1(16.4)

Conclusions

- Canonical implementation testbenches critical to reliable, principled heuristic innovation
- Component-based architecture for FM-based iterative hypergraph partitioners
- Demonstrate effects of underspecified, “implicit” implementation decisions
- Achieve leading-edge solution qualities, runtimes using our testbench