Adaptive FPGA Placement Optimization via Reinforcement Learning

Kevin E. Murray and Vaughn Betz
Reinforcement Learning & CAD
CAD Tool Development: Human-in-the-loop

Large solution space → Use heuristics
CAD Tool Development: Human-in-the-loop

Large solution space → Use heuristics
CAD Tool Development: Human-in-the-loop

Large solution space → Use heuristics

Idea
CAD Tool Development: Human-in-the-loop

Large solution space → Use heuristics
CAD Tool Development: Human-in-the-loop

Large solution space → Use heuristics
CAD Tool Development: Human-in-the-loop

Large solution space → Use heuristics
CAD Tool Development: Human-in-the-loop

Large solution space → Use heuristics
CAD Tool Development: Human-in-the-loop

Large solution space → Use heuristics
CAD Tool Development: Human-in-the-loop

Large solution space → Use heuristics

With human-in-the-loop:

• Slow

• Simple heuristics, limited tool parameters (to keep tractable)

• Tune for average case (can’t investigate every benchmark design)
With RL:

- Human *out* of the loop!
- Learn better heuristics: exploit more information, more parameters
- Online adaptation → better than average case
With RL:

• Human out of the loop!

• Learn better heuristics: exploit more information, more parameters

• Online adaptation → better than average case
FPGA Placement
FPGA Placement

FPGA:
- Pre-fabricated programmable blocks and routing
- Can implement wide range of designs

Placement Considerations:
- Key step for timing (no later fix up)
- Routing architecture dependent
- Many legality constraints
- Discrete optimization
- Large designs (millions of netlist primitives)
Simulated Annealing (SA) Placement

- Modify placement by making ‘moves’
- Accept/Reject move based on:
  - Cost change
  - Temperature (hill climbing)
Move Generation
Many possible types of moves!
Move Generation

Many possible types of moves!

Simple: random swap
Move Generation
Many possible types of moves!

Simple: random swap
Move Generation
Many possible types of moves!

**Simple:** random swap

**Smart:** directed move to ‘good’ location (wirelength, timing)
Move Generation
Many possible types of moves!

Simple: random swap

Smart: directed move to ‘good’ location (wirelength, timing)
Move Generation
Many possible types of moves!

Simple: random swap

Smart: directed move to ‘good’ location (wirelength, timing)

Complex: Analytic
Move Generation
Many possible types of moves!

**Simple:** random swap

**Smart:** directed move to ‘good’ location (wirelength, timing)

**Complex:** Analytic
Move Generation
Many possible types of moves!

**Simple:** random swap

**Smart:** directed move to ‘good’ location (wirelength, timing)

**Complex:** Analytic

**Complex:** Assignment
Move Generation
Many possible types of moves!

**Simple:** random swap

**Complex:** Analytic

**Smart:** directed move to ‘good’ location (wirelength, timing)

**Complex:** Assignment
Move Generation
Many possible types of moves!

Simple: random swap

Smart: directed move to 'good' location (wirelength, timing)

Complex: Analytic

Complex: Assignment

Many considerations:
• Frequencies of different moves
• Situation dependent?
• Move ‘strength’ vs run-time
Move Generation
Many possible types of moves!

Simple: random swap

Smart: directed move to ‘good’ location (wirelength, timing)

Complex: Analytic

Complex: Assignment

Many considerations:
• Frequencies of different moves
• Situation dependent?
• Move ‘strength’ vs run-time

Treat as RL Problem!
RL Move Generator

Actions: moved different block types

Reward:
- Accepted: -$\Delta$cost
- Rejected: 0

Agent:
- Estimates value of actions
- Selects action to take
Estimating Action Values

- Values of action are not stationary!
Estimating Action Values

- Values of action are not stationary!
Estimating Action Values

• Values of action are not stationary!
Estimating Action Values

- Values of action are not stationary!

Agent determines ‘good’ move types online!
Action Selection: Exploration vs Exploitation

- $\epsilon$-greedy: Mostly greedy (exploit), occasionally random (explore)
- $\epsilon$: fraction of exploratory moves
Action Selection: Exploration vs Exploitation

- $\varepsilon$-greedy: Mostly greedy (exploit), occasionally random (explore)
- $\varepsilon$: fraction of exploratory moves

Exploit to save run-time
Action Selection: Exploration vs Exploitation

- $\epsilon$-greedy: Mostly greedy (exploit), occasionally random (explore)
- $\epsilon$: fraction of exploratory moves

No exploration harms quality

Exploit to save run-time
Quality/Run-time Comparison

- VTR Benchmarks (10K-165K primitives), 3 seeds
Quality/Run-time Comparison

- VTR Benchmarks (10K-165K primitives), 3 seeds
Quality/Run-time Comparison

- VTR Benchmarks (10K-165K primitives), 3 seeds
Quality/Run-time Comparison

- VTR Benchmarks (10K-165K primitives), 3 seeds
Quality/Run-time Comparison

- VTR Benchmarks (10K-165K primitives), 3 seeds

![Graph showing Quality/Run-time Comparison with VTR 8 (hand tuned), Random Agent, and RL Agent]

Wirelength (Normalized) vs. Place Time (Normalized, log scale)
Quality/Run-time Comparison

- VTR Benchmarks (10K-165K primitives), 3 seeds

Same quality
20% faster
Quality/Run-time Comparison

- VTR Benchmarks (10K-165K primitives), 3 seeds

![Graph showing Quality/Run-time Comparison](graph)

- Same quality: 20% faster
- Better quality: 50% faster
Conclusion

• RL-enhanced Simulated Annealing based FPGA Placer
• RL agent controlled move generator
• Learns on-line what types of moves are productive
• Improves run-time/quality trade-offs
  • Particularly at low run-times
Future Work

• More types of moves
• Other reward formulations (e.g. cost run-time)?

• Agent:
  • Less greedy action selection (soft-max)?
  • Use more state information: Circuit & Optimizer statistics

• Learn:
  • Off-line agent training
  • Other RL algorithms (e.g. Temporal Difference Learning, Policy Gradients)

• Explore RL elsewhere in CAD flow
Backup
Quality/Run-time Comparison: Critical Path

- VTR Benchmarks (10K-165K primitives), 3 seeds
Estimating Action Values: Time Scale

Long Time-scale

Short Time-scale
Reinforcement Learning (RL) for CAD: Challenges

Long CAD Run-times
• Must exploit limited experience

Long delayed rewards
• Core challenge of RL
• CAD has well defined objectives

Nested black-box optimization
• CAD optimization already difficult to interpret/debug
• Nested optimization makes interpretability more challenging