Generating Counter-examples through Randomized Guided Search

Neha Rungta and Eric G. Mercer

Verification and Validation Lab

Computer Science Department

Brigham Young University

Provo, UT - 84602

Acknowledgements

Ira & Mary Lou Fulton for the Fulton Supercomputing Lab at BYU

Marylou4, cluster of 618 nodes
Among top 50 supercomputers

Acknowledgements

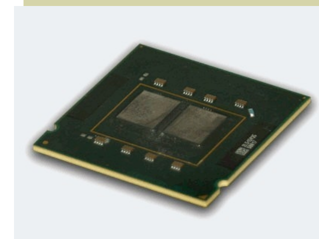
Matt Dwyer at UNL

Suzette Person at UNL

Shmuel Ur at

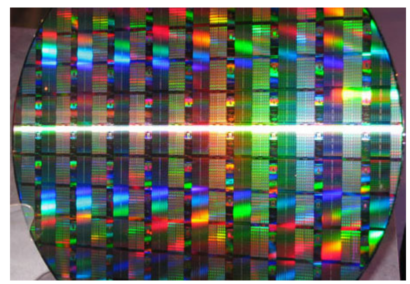
IBM Research Lab, Haifa

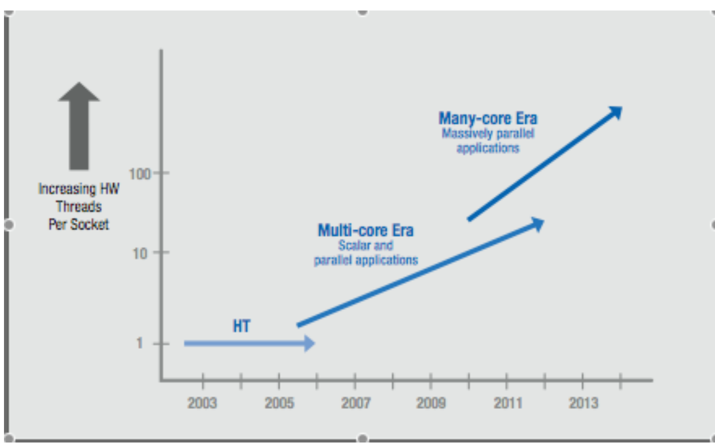
Current Trend



Dual and Quad Core Processors are becoming increasingly common

Intel's 80 core prototype





More processors on a single die *

* Image courtesy Intel white paper

Distributed/Parallel Model Checking

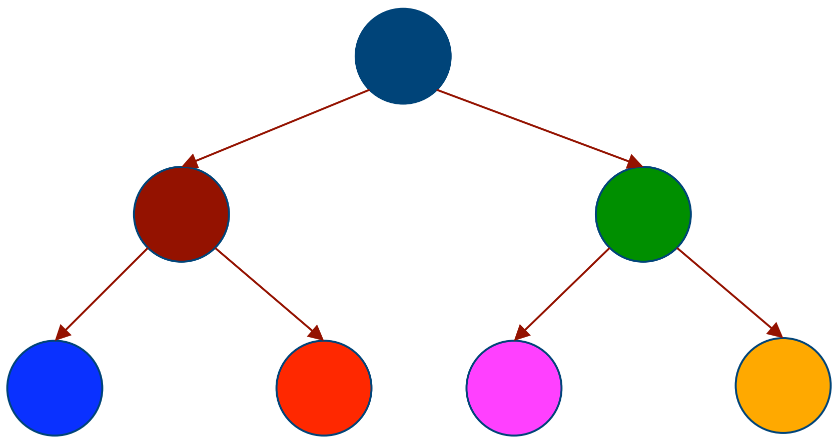
- Exhaustive proof is the heart of model checking
- Enumerate entire behavior space
- Complexity of system limits practical application
- Parallel model checking shows limited promise
- Shift focus to bug-finding (counter-examples)
- Parallel search for bugs using randomization

Contributions

- Low-overhead randomized greedy best-first search
- Empirical study over a very large characterized Java benchmark suite using JPF 4.0
- Empirical study in Estes

Default Search Order in DFS

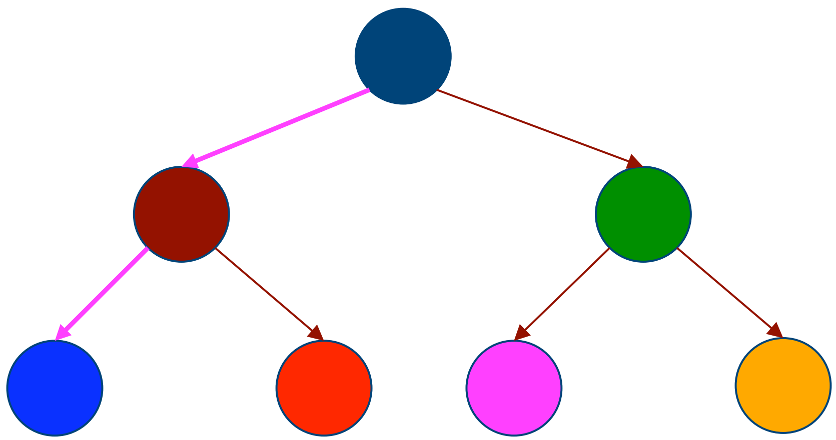
Dwyer et al. (FSE '06)



- **Search follows a deterministic order**

Default Search Order in DFS

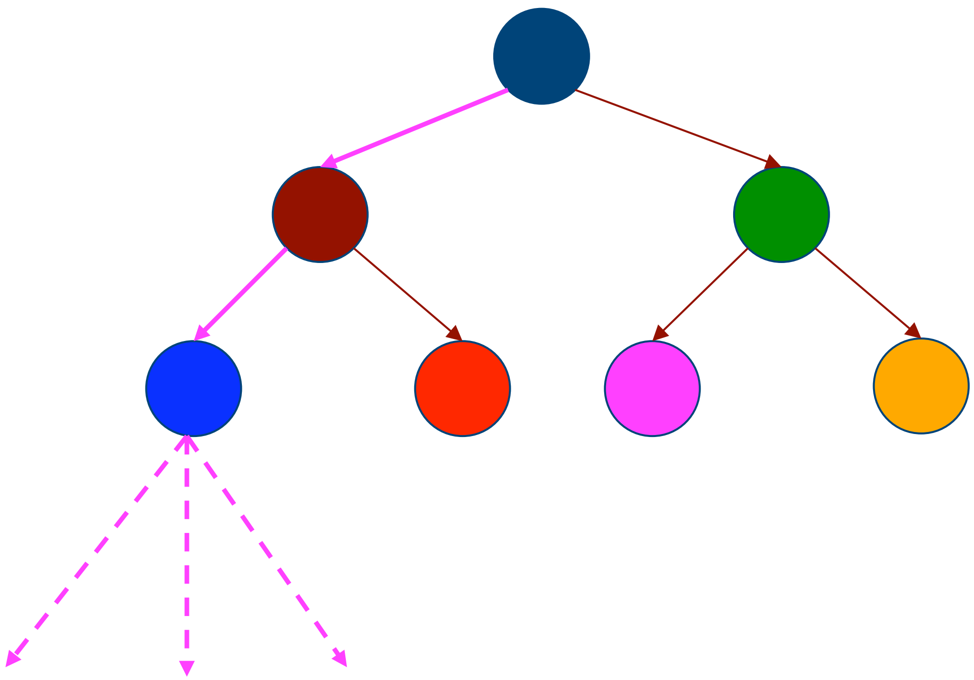
Dwyer et al. (FSE '06)



- **Order depends on model checker implementation**

Default Search Order in DFS

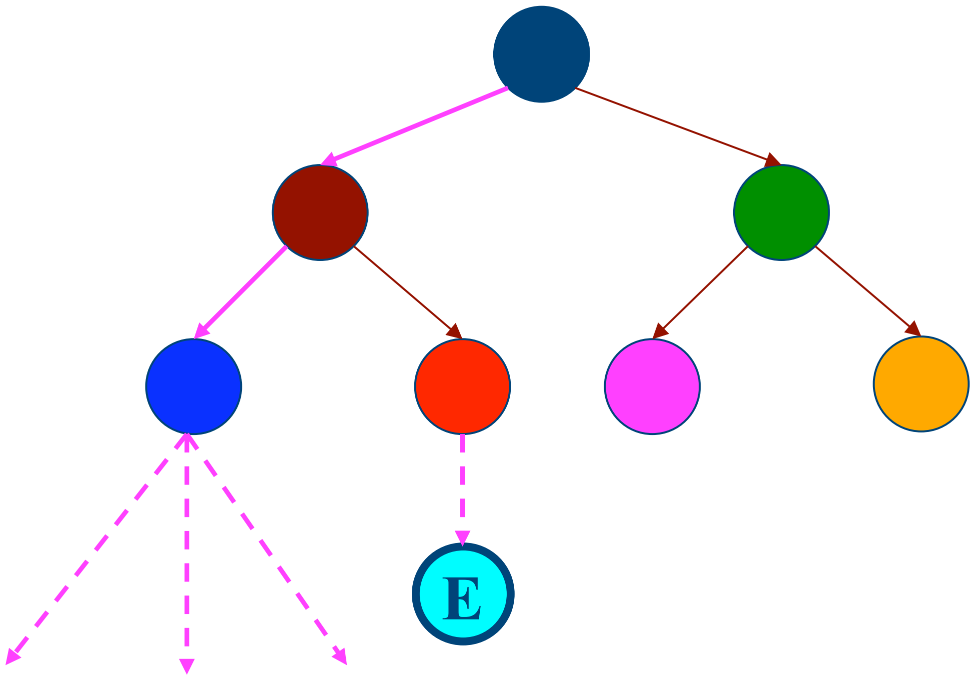
Dwyer et al. (FSE '06)



- **Spend all the time in one portion of state graph**

Default Search Order in DFS

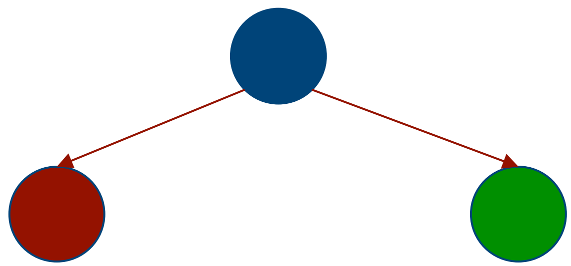
Dwyer et al. (FSE '06)



- **The error may lie along a different path**

Parallel Randomized DFS

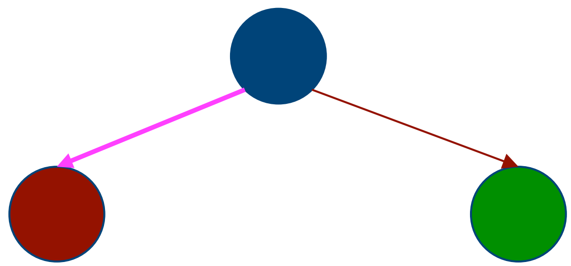
Dwyer et al. (ICSE '07)



- **Randomly picks a successor to explore**

Parallel Randomized DFS

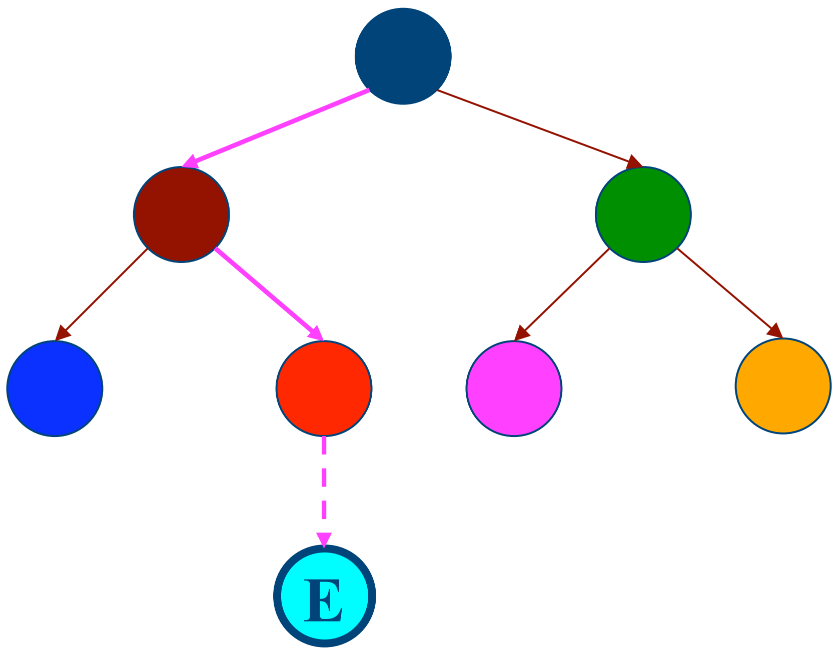
Dwyer et al. (ICSE '07)



- **Randomly picks a successor to explore**

Parallel Randomized DFS

Dwyer et al. (ICSE '07)



- Embarrassingly parallel
- Aim is to find a counterexample

Guided Search Basics

- Order state by priority using heuristic
- Replace stack with priority queue in search
- Heuristic type determines type of search:
  - greedy best-first: ignores path cost
  - best-first: includes current path cost
  - A*: includes current path and heuristic is admissible
- We focus on greedy best-first search
- GDS stands for GuiDed Search (greedy best-first)

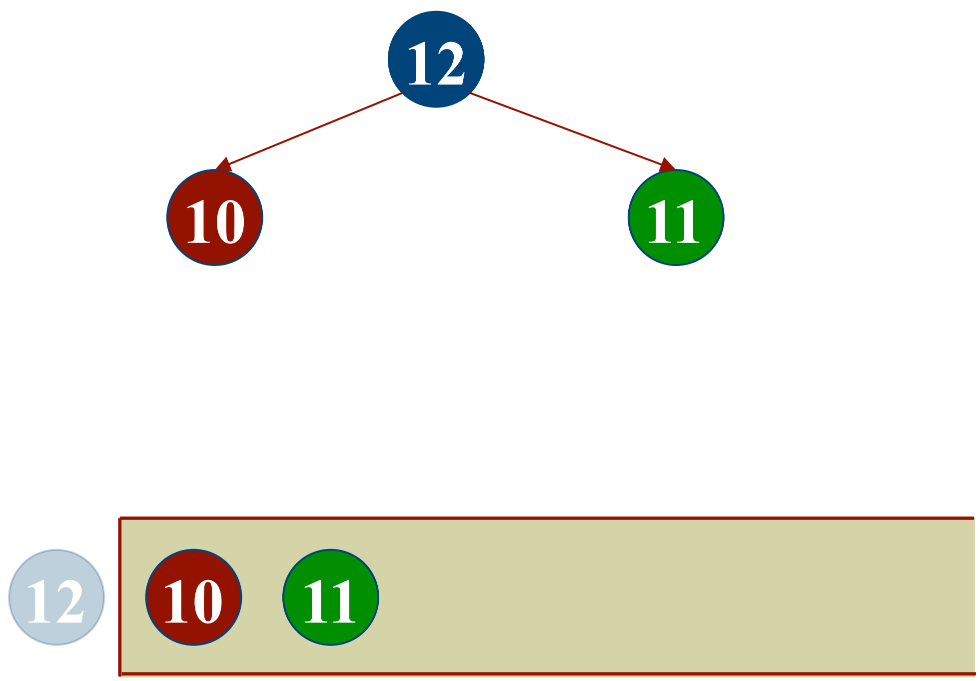
Default Search order in GDS

12

12

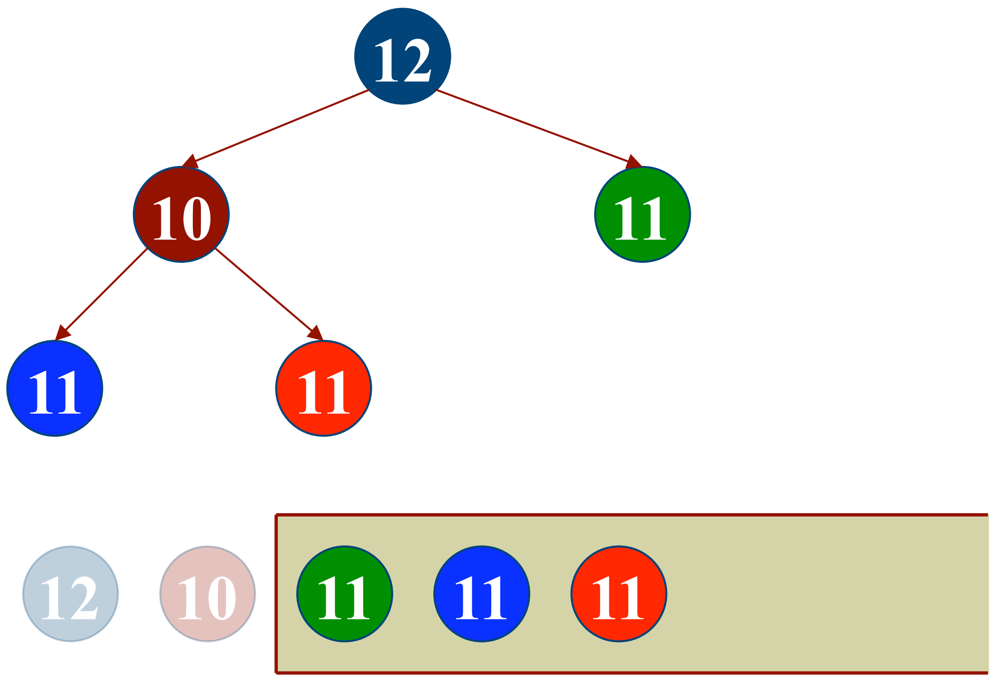
- Greedy best-first search
- Uses the heuristic estimate to guide the search

Default Search order in GDS



- **Orders states in a PQ based on the rank**

Default Search order in GDS



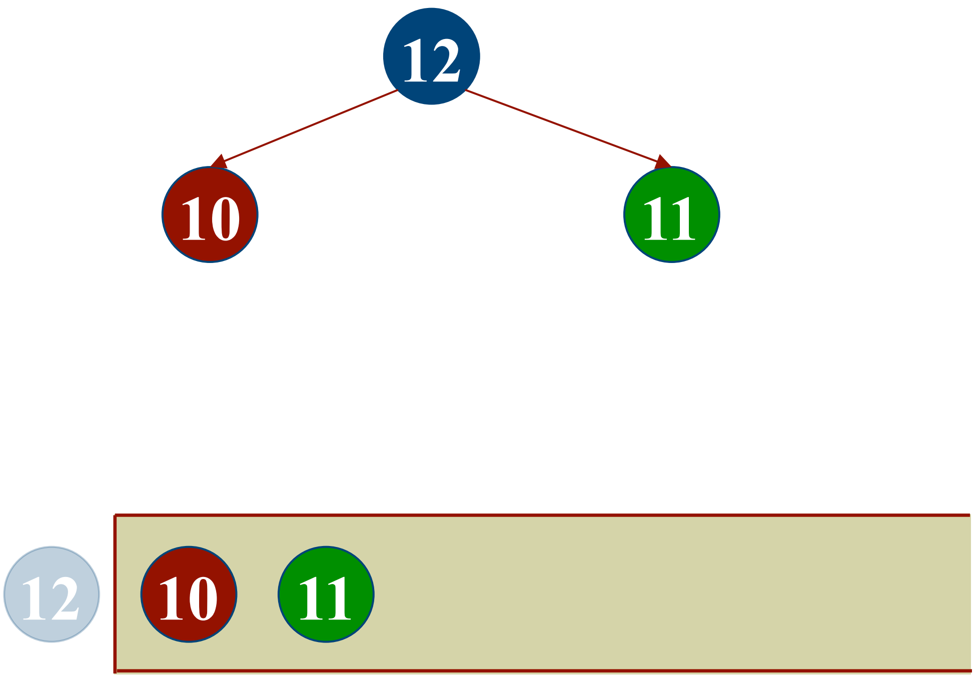
- **Priority queue determines ordering**

Randomized GDS in JPF

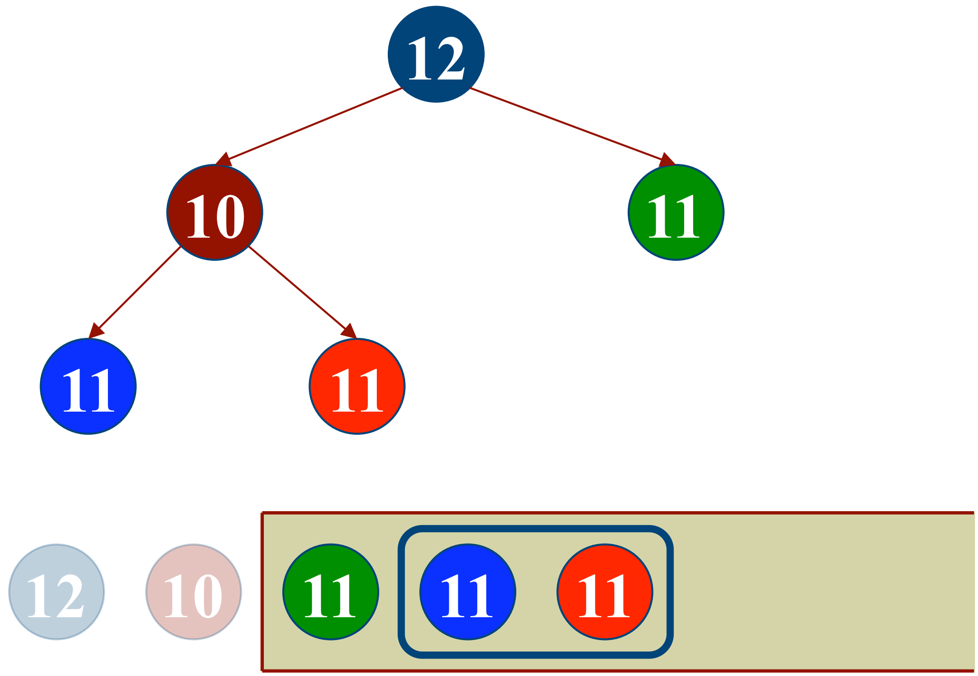
12

12

Randomized GDS in JPF

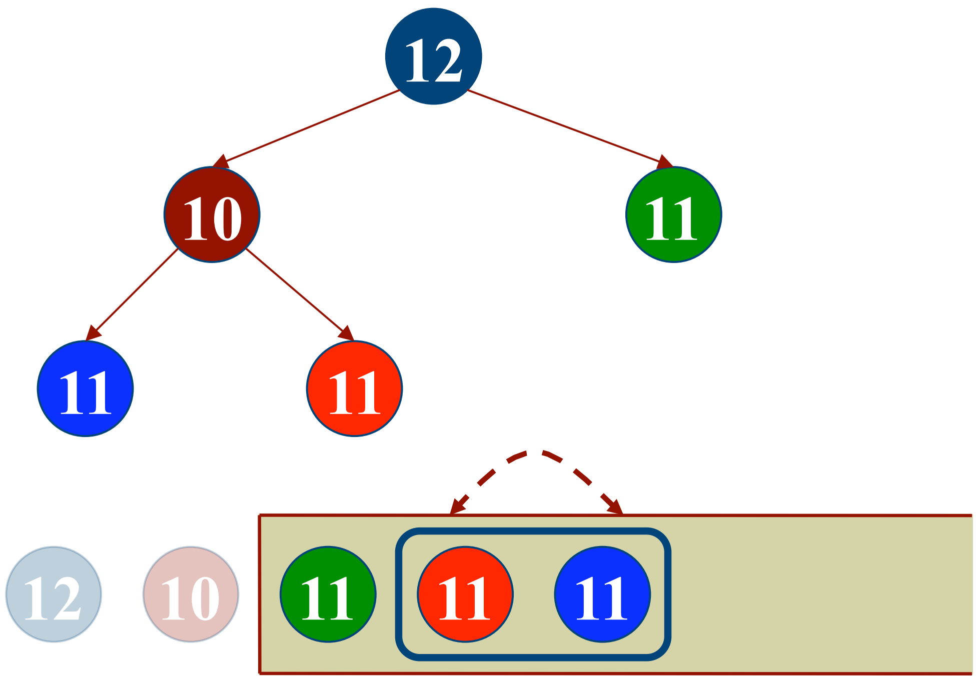


Randomized GDS in JPF



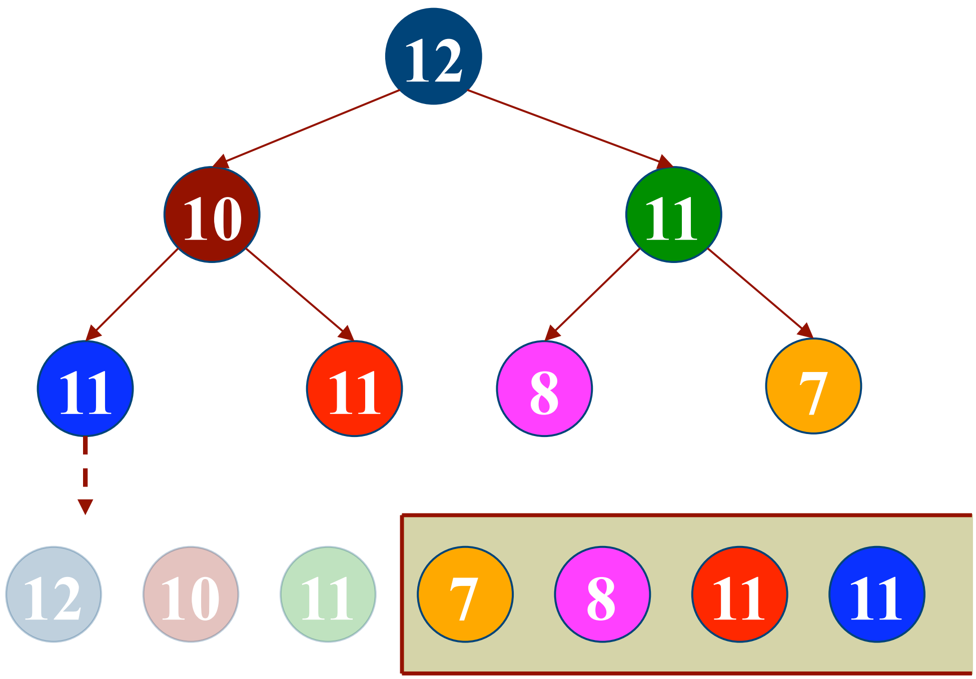
- JPF has an option to randomize successors
- The priority queue resolves ties

Randomized GDS in JPF



- Controls for default order in siblings
- Does not control for common heuristic values
- Not effective in randomizing default order

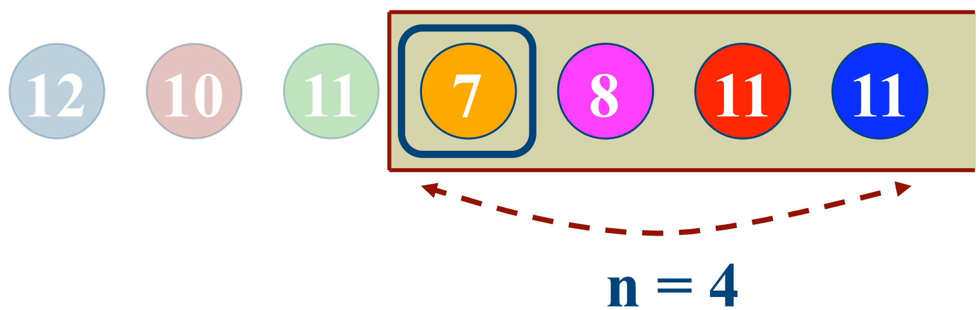
Randomized GDS in JPF



- **The error may be along a different path**

Randomly pick from n-best

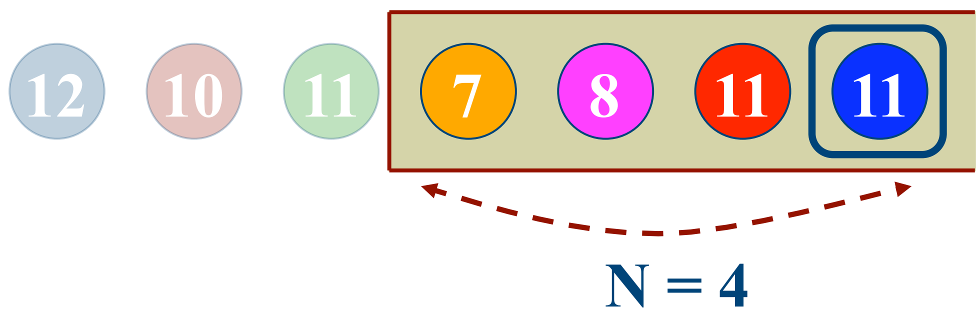
Jones and Mercer (SPIN '04)



- **Picks one of *n* candidates**
- **Does not consider ranking**
- **Moderately effective in error discovery**

Randomly pick from n-best

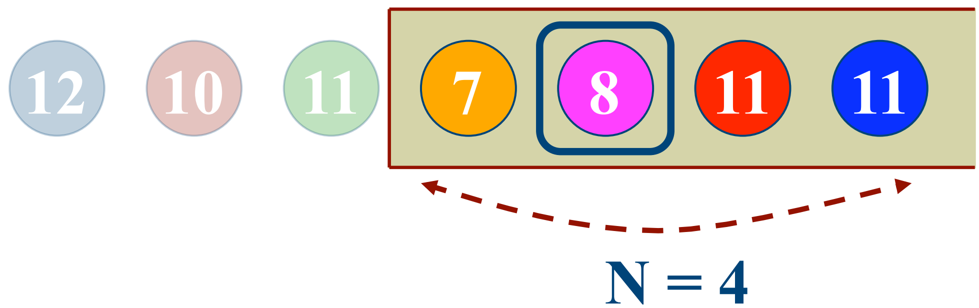
Jones and Mercer (SPIN '04)



- **Disregards the heuristic ranking**

Randomly pick from n-best

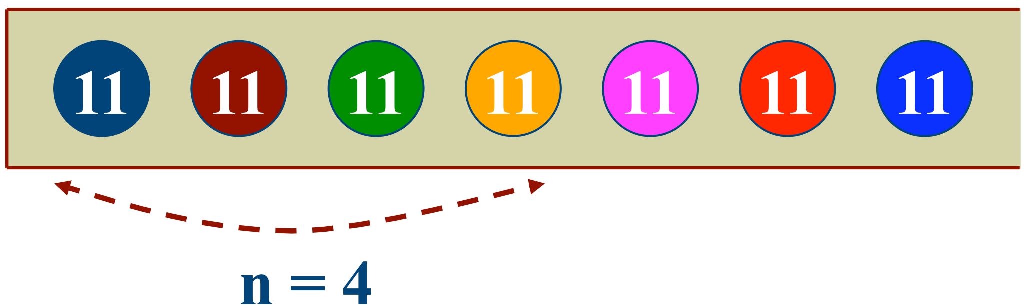
Jones and Mercer (SPIN ‘04)



- **Disregards the heuristic ranking**

Randomly pick from n-best

Jones and Mercer (SPIN '04)



- Does not randomize all heuristic ties
- Not effective in Java benchmarks in JPF

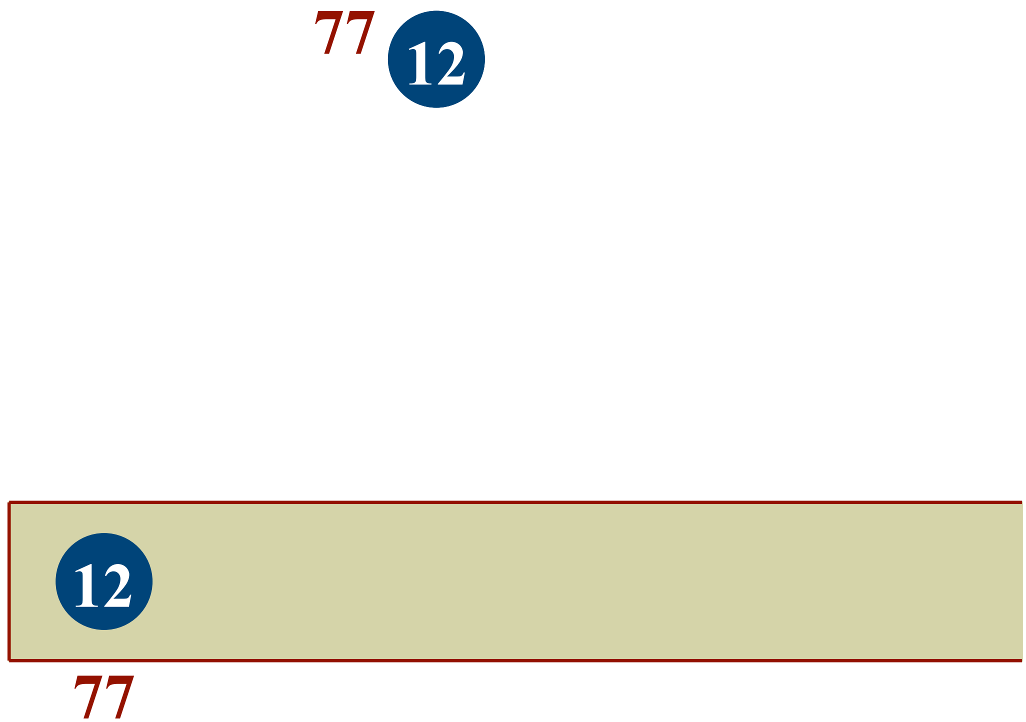
Unable to counter default order

- Both techniques are insufficient
- Comparison to GDS with default order
- Empirical analysis of the RGDS techniques
- No statistical difference for most examples
- Results for existing RGDS omitted

Scope of this work

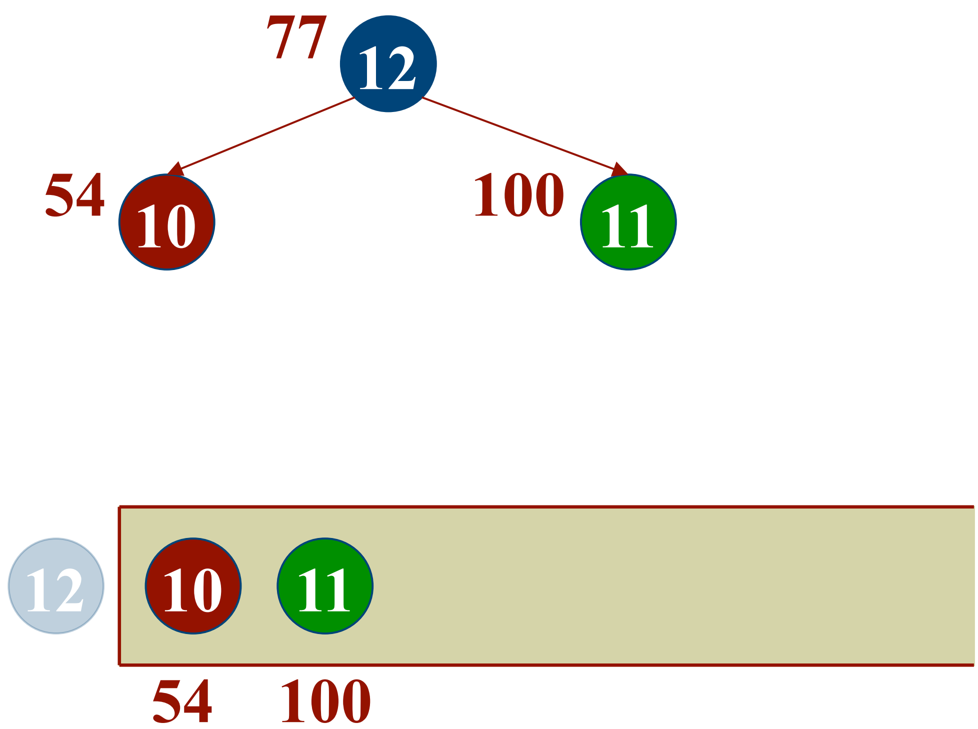
- Focuses on a greedy best-first search
- Best-first search with inadmissible heuristics
- Results of A^* not significantly affected

Our Randomized GDS

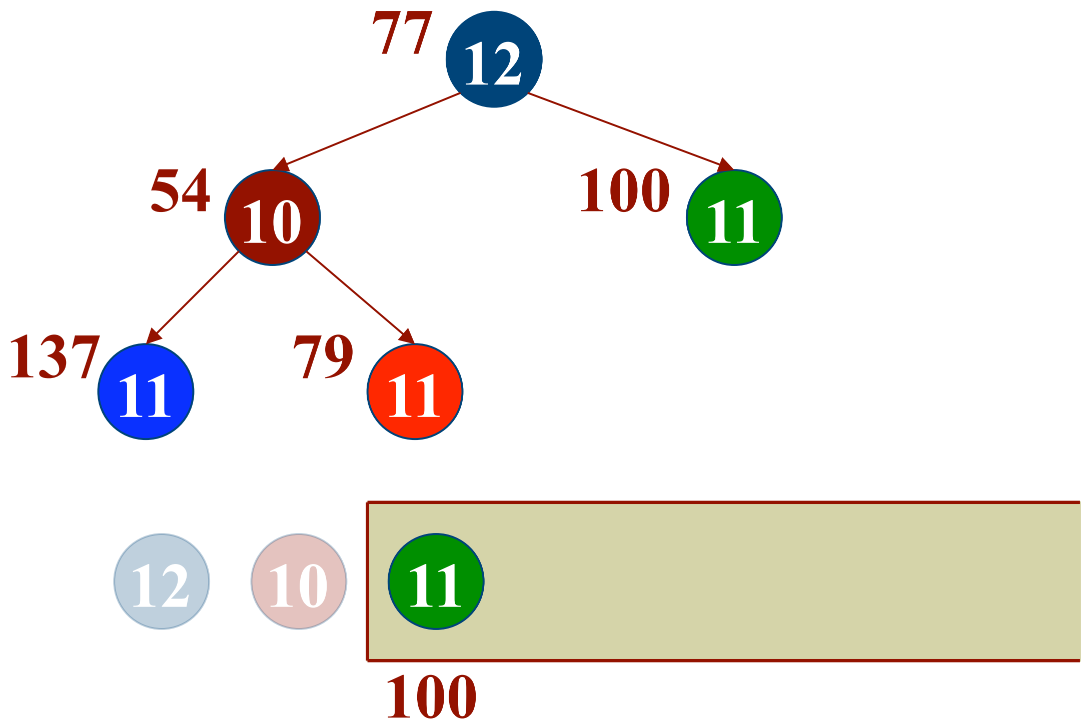


- **Add a random value as a secondary key**

Our Randomized GDS

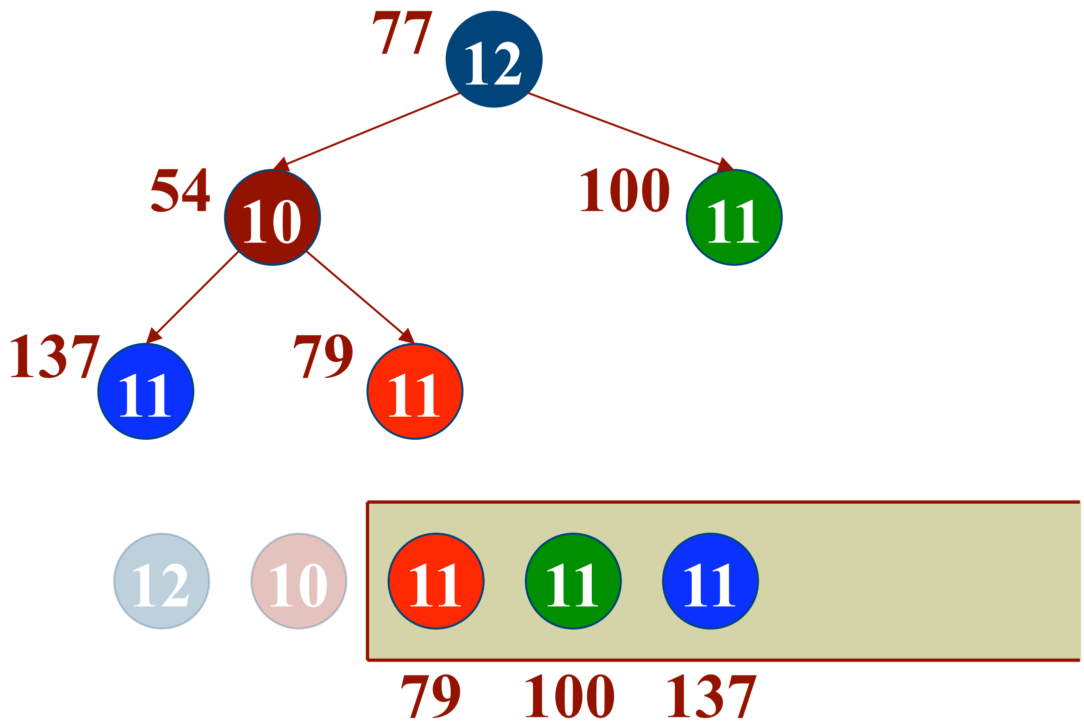


Our Randomized GDS



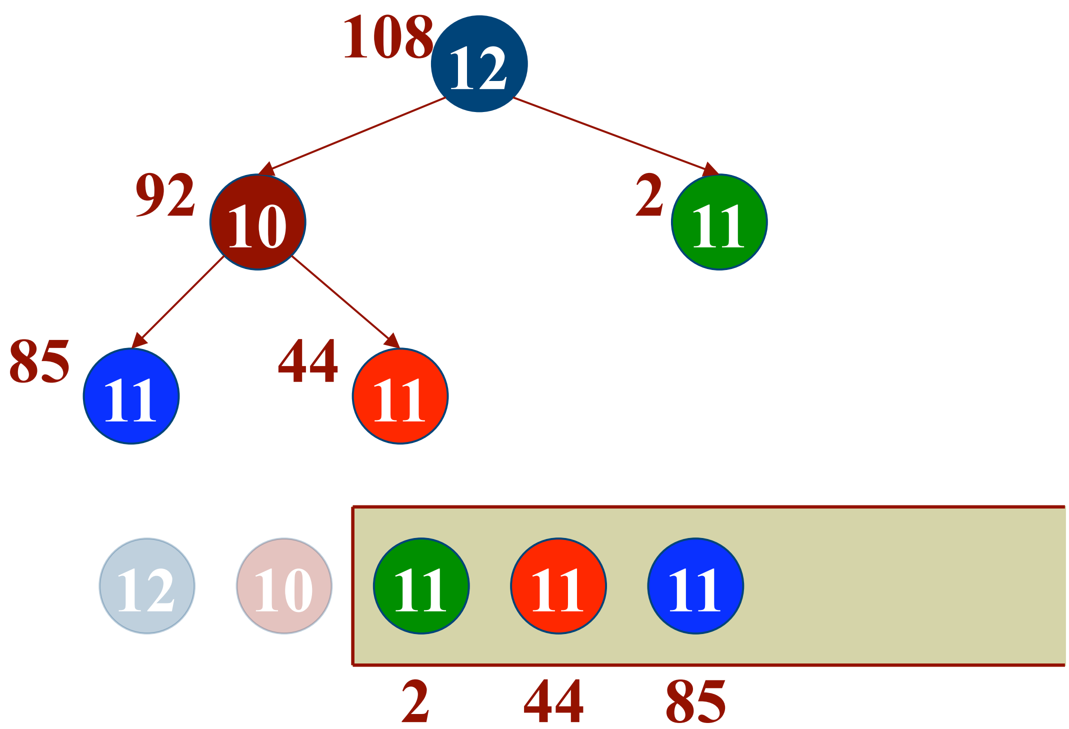
- Secondary key used to break heuristic ties

Our Randomized GDS



- Secondary key used to break heuristic ties

Our Randomized GDS



- Embarrassingly Parallel
- Scales to Arbitrary Number of Nodes

Research Question

- Does the randomized GDS perform better than guided search with default search order?

- Compare default order to randomized GDS
- Published Java heuristics and models in JPF v4.0
- Distance heuristics in Estes on Barbershop model
- 100 trials of randomized GDS on each model
- One hour time bound
- 7 GB RAM for JPF and 2 GB for Estes

Empirical study

- Marylou4: Cluster of 618 nodes
- Two dual core processors per node (2.6 GHz)
- Intel EM64T processors
- JPF v4.0 for Java Benchmarks
- Estes model checker for C models

Empirical Study

- 100 trails of randomized GDS in parallel
- Time bounded for 1 hour
- 7 GB of RAM for the trials in JPF
- 2 GB of RAM for the trials in Estes

Independent Variables

- Heuristics in JPF
- Distance heuristics in Estes
- Subjects with concurrency errors
- Used in extensive benchmarking studies
  (*Dwyer et al. FSE ‘06*)
  (*Rungta and Mercer SEFM ‘07*)

Dependent Variables

- Path Error Density, the ratio of error finding RGDS trials over total number of trials
- Number of states generated

JPF Results

Model	PED	RGDS Avg. States	GDS States
RaxExtended(4,3)	1.00	20,774	1,225,743*
Twostage(6,1)	0.94	486,830	716,413
Piper(2,4,4)	0.87	1,229,530	2,478,360*
Reorder(10,1)	0.00	-	1,727,521

JPF Results

Model	PED	RGDS Avg. States	GDS States
RaxExtended(4,3)	1.00	20,774	1,225,743*
Twostage(6,1)	0.94	486,830	716,413
Piper(2,4,4)	0.87	1,229,530	2,478,360*
Reorder(10,1)	0.00	-	1,727,521

JPF Results

Model	PED	RGDS Avg. States	GDS States
RaxExtended(4,3)	1.00	20,774	1,225,743*
Twostage(6,1)	0.94	486,830	716,413
Piper(2,4,4)	0.87	1,229,530	2,478,360*
Reorder(10,1)	0.00	-	1,727,521

Estes Results for Barbershop(9)

Heuristic	PED	RGDS Avg. States	GDS States
FSM (*Edelkamp and Mehler MoChart '04*)	0.59	785,698	492,166
EFSM (*Rungta and Mercer ASE '05*)	0.65	816,848	17,537
E-FCA (*Rungta and Mercer FMCAD '06*)	1.00	1,692	814

Evaluation

- RDFS and RGDS overcome default order
- RDFS provides a good lower bound on hardness (*Rungta and Mercer, SEFM '07*)
- Heuristics are restricted to a class of subjects
- RDFS ideal comparison for RGDS

Research Question

- How does randomized GDS compare with randomized DFS?

- Published Java heuristics and models in JPF v4.0
- Distance heuristics in Estes and C versions of select models
- 100 trials of randomized GDS and Randomized DFS
- One hour time bound
- 7 GB RAM for JPF and 2 GB for Estes
- Bounded queue of 100,000 states (*arbitrary choice*)

Empirical Study

- Similar set up as previous study
- 100 trials of RDFS and RGDS
- 1 hour time bounded
- Size of the frontier in RGDS prohibitive
- Bounded the Queue at 100,000 states (*arbitrary choice*)

Possibly Prune the Bug?

- Yes! But…
- Otherwise run out of memory (10 to 30 mins)

Independent Variables

- Pick subjects characterized as “hard” (*Rungta and Mercer SEFM ‘07*)
- Models where RDFS struggles

Dependent Variables

- Path Error Density
- Number of states generated
- Time Taken before Error Discovery
- Length of the Counterexample
- Total Memory usage
- Minimum, Average, and Maximum values

Normalization

- Min, Avg, and Max normalized to 0 and 1
- Minimum is normalized to 1.00
- Maximum is normalized to 0.00
- All other values are in between
- Process conducted for each metric separately
- Allows better understanding on same scale

PED for Prefer-thread

Model	RDFS	RGDS
ProdCons(1,16,4)	0.67	0.87
TwoStage(7,1)	0.41	0.73
WrongLock(1,20)	0.28	0.81
Reorder(10,1)	0.00	0.34

PED for Prefer-thread

Model	RDFS	RGDS
ProdCons(1,16,4)	0.67	0.87
TwoStage(7,1)	0.41	0.73
WrongLock(1,20)	0.28	0.81
Reorder(10,1)	0.00	0.34

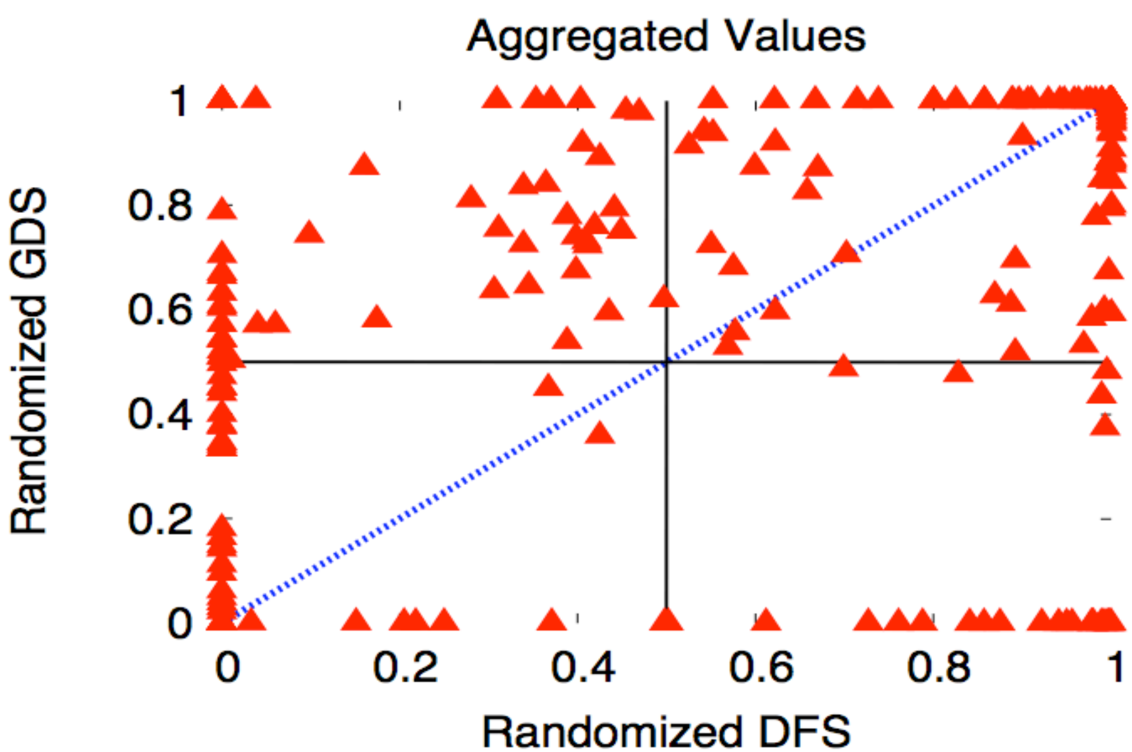
PED for Prefer-thread

Model	RDFS	RGDS
ProdCons(1,16,4)	0.67	0.87
TwoStage(7,1)	0.41	0.73
WrongLock(1,20)	0.28	0.81
Reorder(10,1)	0.00	0.34

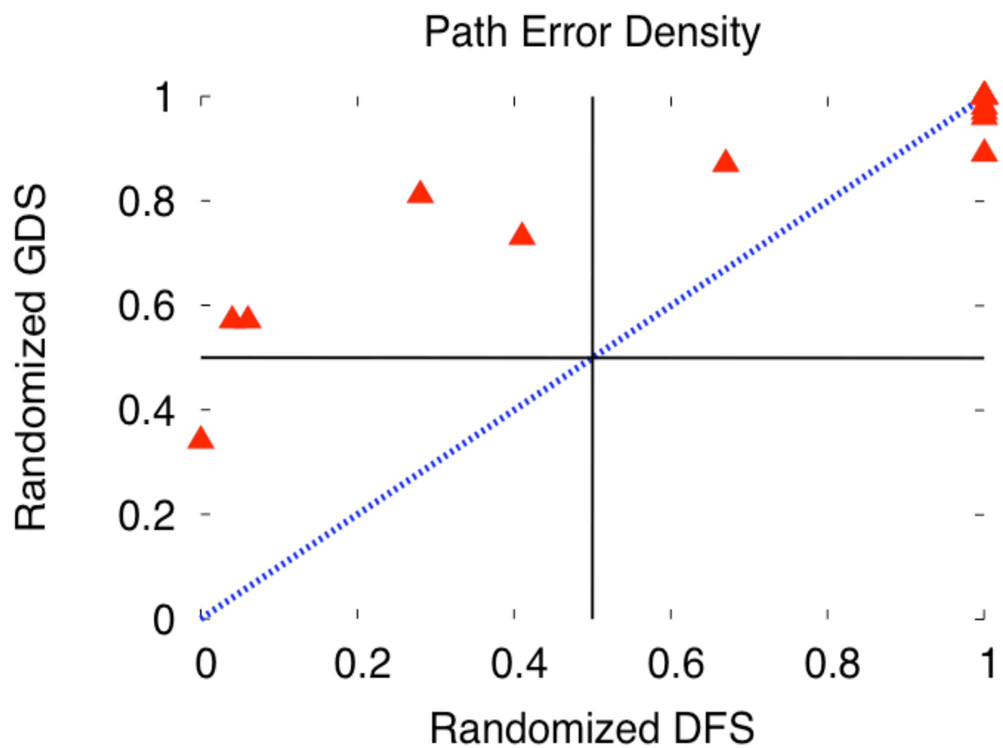
PED for Prefer-thread

Model	RDFS	RGDS
ProdCons(1,16,4)	0.67	0.87
TwoStage(7,1)	0.41	0.73
WrongLock(1,20)	0.28	0.81
Reorder(10,1)	0.00	0.34

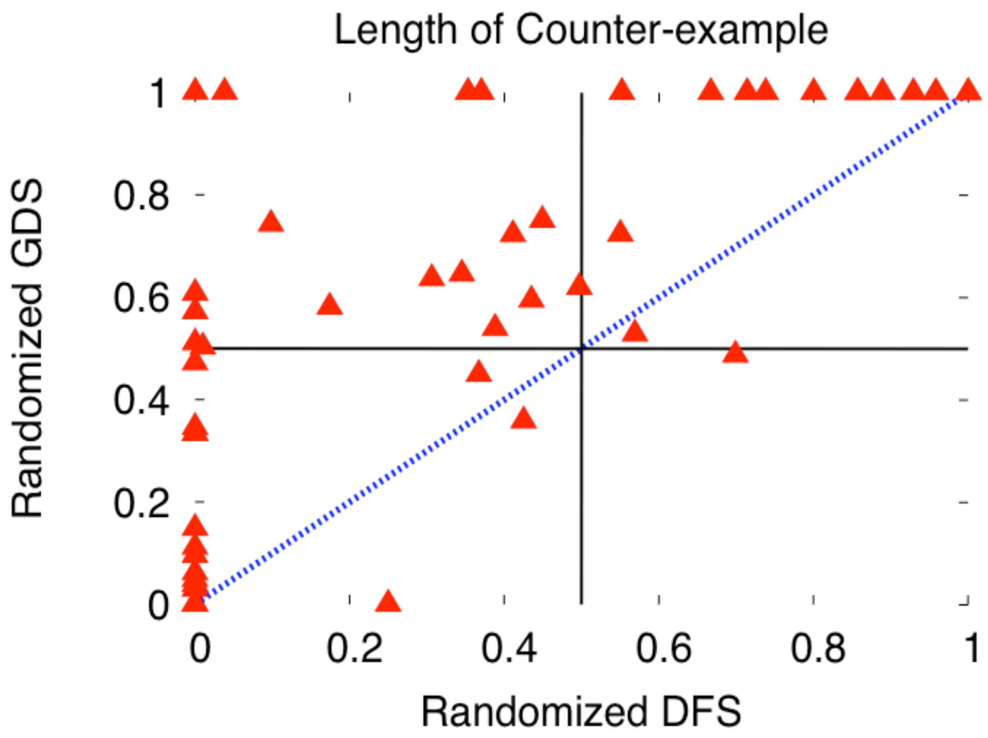
Scatter Plot on All Dependents



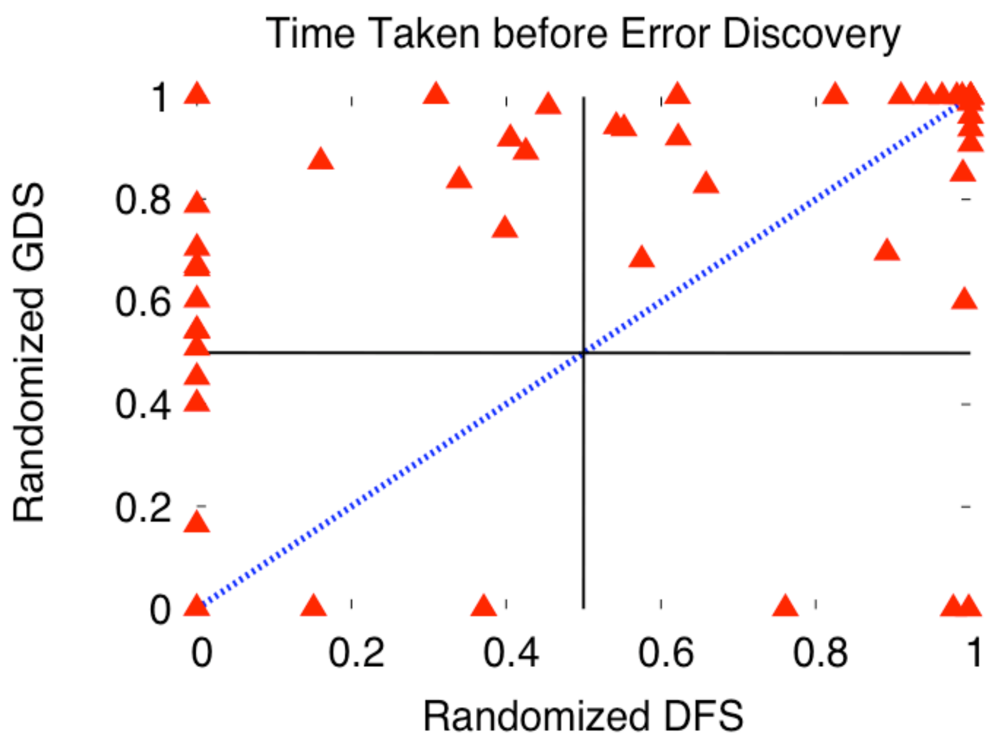
Path Error Density Scatter Plot



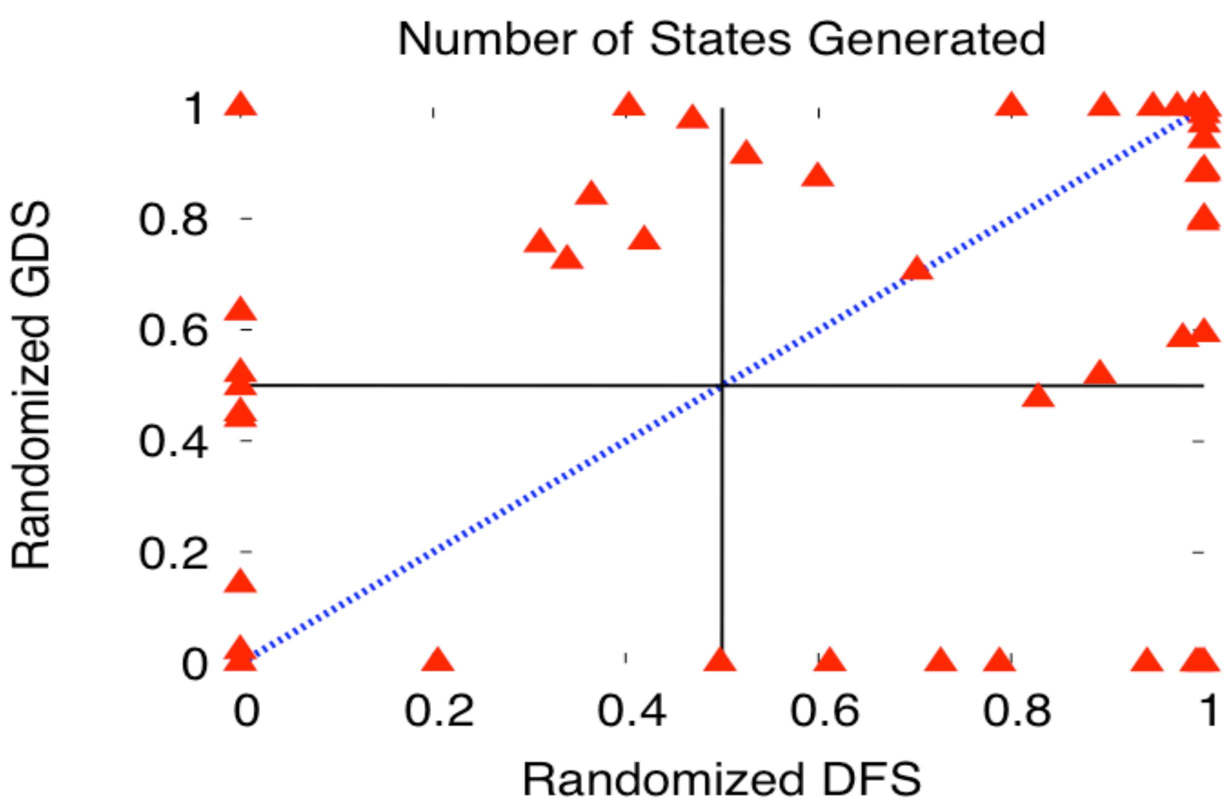
Length of Counter Example



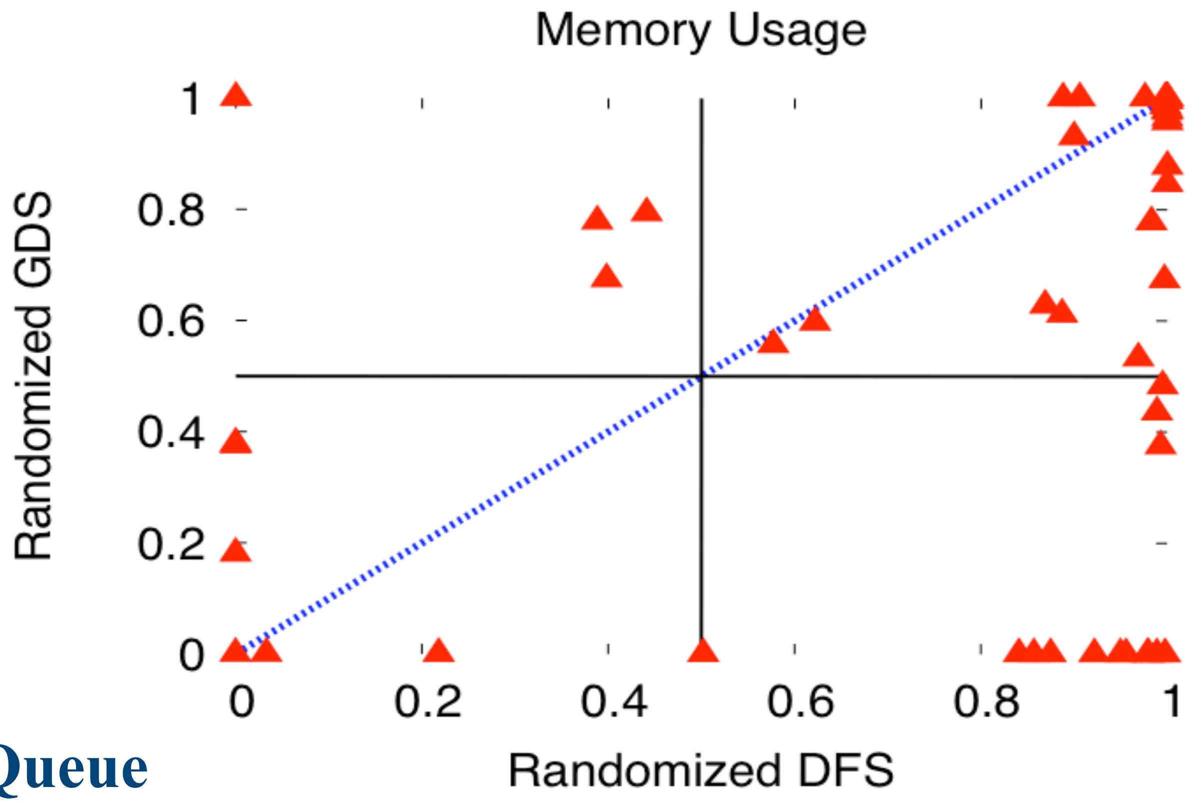
Time Taken for error discovery



Number of States Generated



Memory Usage



Priority Queue uses memory!

PED for Most-Blocked

Model	RDFS	RGDS
Piper(2,4,,4)	1.00	1.00
Piper(2,8,4)	0.96	0.00
Clean(10,10,1)	0.96	0.00
Piper(2,16,8)	0.00	0.00

PED for Most-Blocked

Model	RDFS	RGDS
Piper(2,4,,4)	1.00	1.00
Piper(2,8,4)	0.96	0.00
Clean(10,10,1)	0.96	0.00
Piper(2,16,8)	0.00	0.00

PED for Most-Blocked

Model	RDFS	RGDS
Piper(2,4,,4)	1.00	1.00
Piper(2,8,4)	0.96	0.00
Clean(10,10,1)	0.96	0.00
Piper(2,16,8)	0.00	0.00

Estes results

- E-FCA distance heuristic (*Rungta and Mercer FMCAD '06*)

Model	RDFS	RGDS
Airline(10,1)	0.18	1.00
Piper(2,8,4)	0.07	1.00

Conclusions

- Randomization is a good thing
- Embarrassingly parallel
- Helps models well matched to heuristics
- Generally better than RDFS
- Uses the computation resources effectively

Future Work

- Better characterization of benchmarks: syntactic measures with low cost computation
- Static analysis to match heuristics to models
- Better use of randomness to improve error discovery
- BEEM (DiViNE models) characterization and Java/C implemenations

Future Work

- Converting Java benchmarks in C models
- Creating hard C models for RDFS in Estes
- Comparing FSM with JPF heuristics
- Models where JPF heuristics perform poorly
- Coverage obtained by RGDS
- Heuristics that work well with randomization
- Characterize heuristics for specific domains

Questions

Verification and Validation Lab
Computer Science Department
Brigham Young University
Provo, Utah

Neha Rungta: neha@cs.byu.edu
Eric G. Mercer: eric.mercer@byu.edu

http://vv.cs.byu.edu