On the Quality and Quantity of Random Decisions in Stochastic Local Search for SAT

(page 146)

Dave A. D. Tompkins and Holger H. Hoos
University of British Columbia

AI 2006. Québec City. June 9

Bioinformatics, Empirical & Theoretical Algorithmics Laboratory
The University of British Columbia – All Materials © 2006.

Overview

- Motivation
  - Stochastic Local Search for SAT
- Quality
  - Random Number Generators (RNGs)
  - PAC Property
- Quantity
  - De-randomization
  - Number of random decisions
- Conclusions & Future Work

Stochastic Local Search (SLS)

9	4	9	1	1	5	1	7	7
7	1	4	1	1	3	2	8	1
6	3	6	8	1	4	9	4	9
2	5	5	1	5	1	1	3	1
3	7	1	4	5	3	8	1	5
1	9	6	1	5	1	8	1	9
5	2	9	1	2	1	1	7	3
6	6	4	7	3	1	2	4	3
9	2	2	5	7	9	6	1	7

- Large combinatorial problems
- Start with a full (random) variable assignment
- Move to neighbouring (adjacent) solutions
- Typically incomplete

SATisfiability Problem

literal

- (a v b v ¬c) (a v ¬b v d) (¬a v d v e)…

clause

variables

OBJECTIVE:

- Find an assignment of variables (A=T,B=F,…) so that all clauses are SATisfied

CRWALK

- a.k.a. Papamaditrou's algorithm
  [Papadimitriou 1991]
- Nice theoretical bounds:
  - Schöning's algorithm is avg. case $O(1.334^n)$
    [Schöning 1999]
- Conflict-Directed Random Walk
  - Randomly select an unsatisfied clause
  - Flip a random variable from that clause

CRWALK

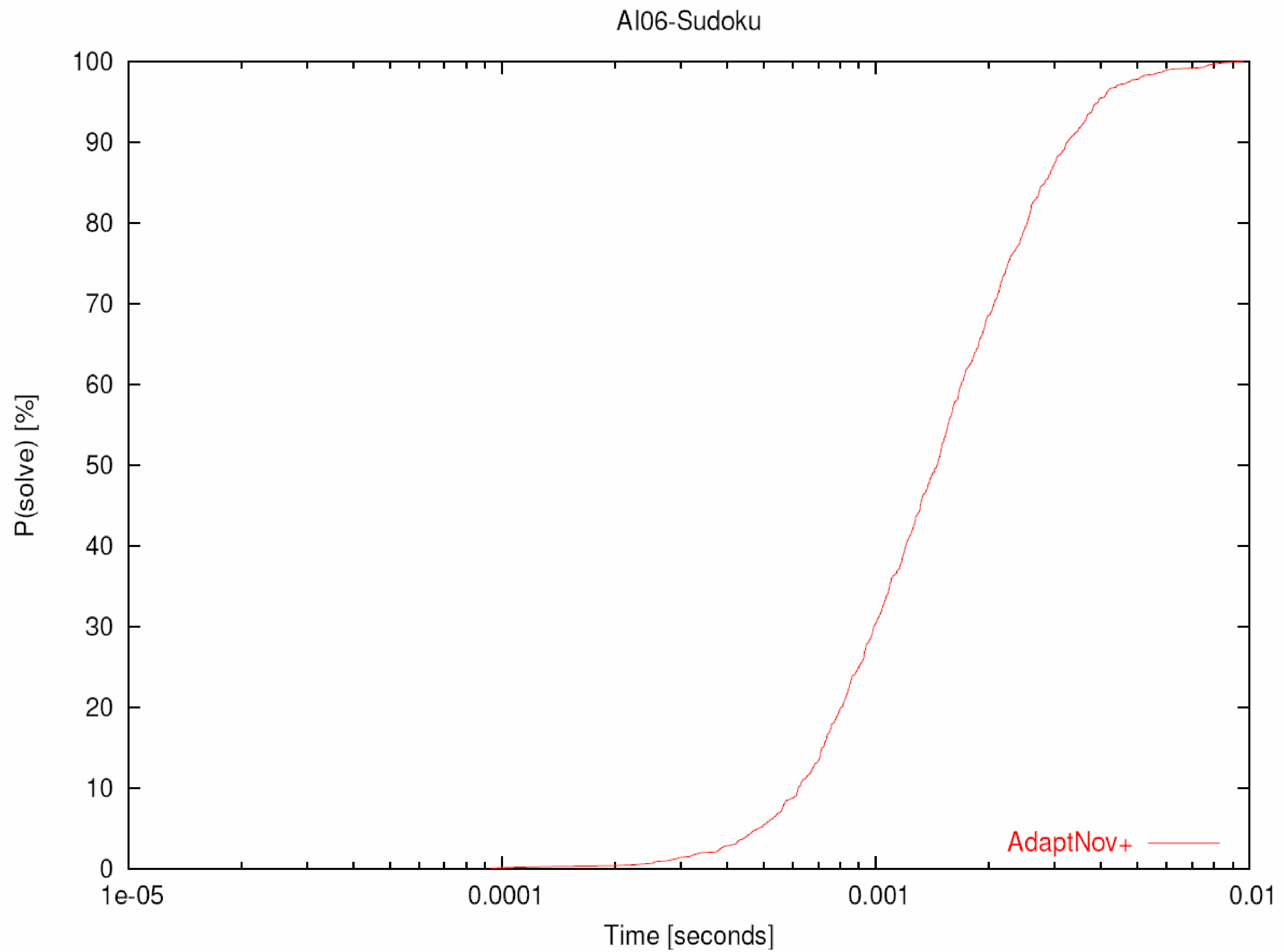
- a=T, b=T, c=T, d=F, e=F…
- (a v b v ¬c) (a v ¬b v d) (¬a v d v e)…

- a=F, b=T, c=T, d=F, e=F…
- (a v b v ¬c) (a v ¬b v d) (¬a v d v e)…

- a=F, b=F, c=T, d=F, e=F…
- (a v b v ¬c) (a v ¬b v d) (¬a v d v e)…

Adaptive Novelty+

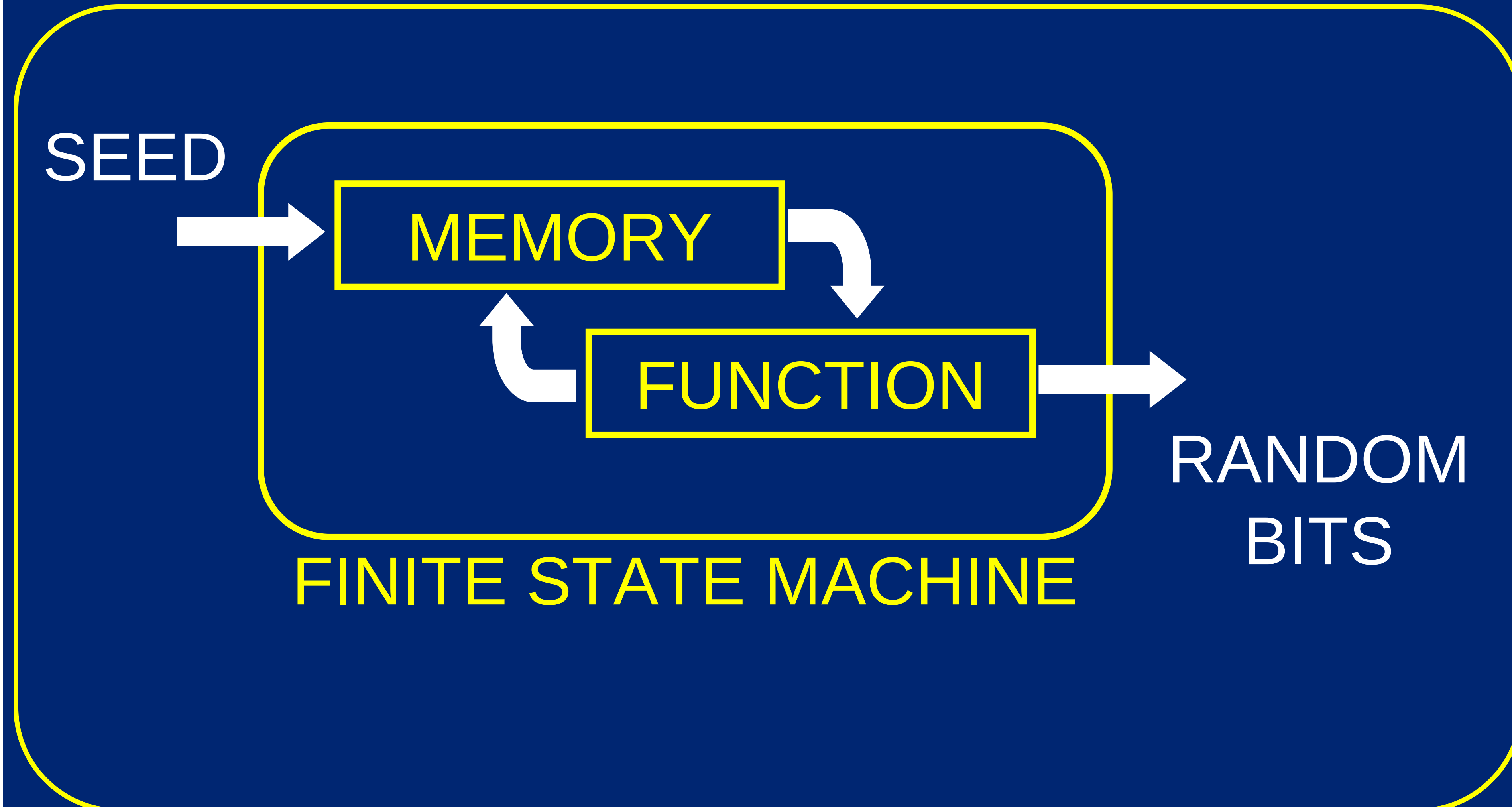
- High performance, *state-of-the-art* SLS algorithm
- SLS Leader in last two SAT competitions
  `www.satcompetition.org`
- Uses random decisions in four different ways:
  - Selecting clauses
  - Decide to take a random walk step
  - Selecting variables in random walks
  - Selecting between “best” & “second best” choices
- Deterministically adapts noise during search
  - Based on current search progress


AI06-Sudoku
P(solve) [%]
100
90
80
70
60
50
40
30
20
10
0
1e-05
0.0001
0.001
0.01
Time [seconds]
AdaptNov+

Random Number Generators (RNGs)

- SLS algorithms use random decisions in a variety of ways
- Obviously a "true" RNG is ideal (prohibitive)
- We use Pseudo-RNGs (PRNGs)
- The qualities of a "good" PRNG:
  - Unbiased
  - Uncorrelated
  - Long Period
- Software packages available for measuring the "quality" of a bitstream
- Quality is related to underlying PRNG function

Model of a Pseudo-RNG



Measuring Quality

- Tested different streams with statistical tests and on SLS algorithms
  - True RNG (atmospheric noise)
  - Pseudo-RNGs:
    - Unix "C" Random
    - Linear congruential [ANSI C]
    - Lagged Fibinoacci [Knuth]
    - Mersenne twister [Matsumoto, Takuji]
  - Intentionally bad streams:
    - Added bias
    - Cycled (periodic) behaviour

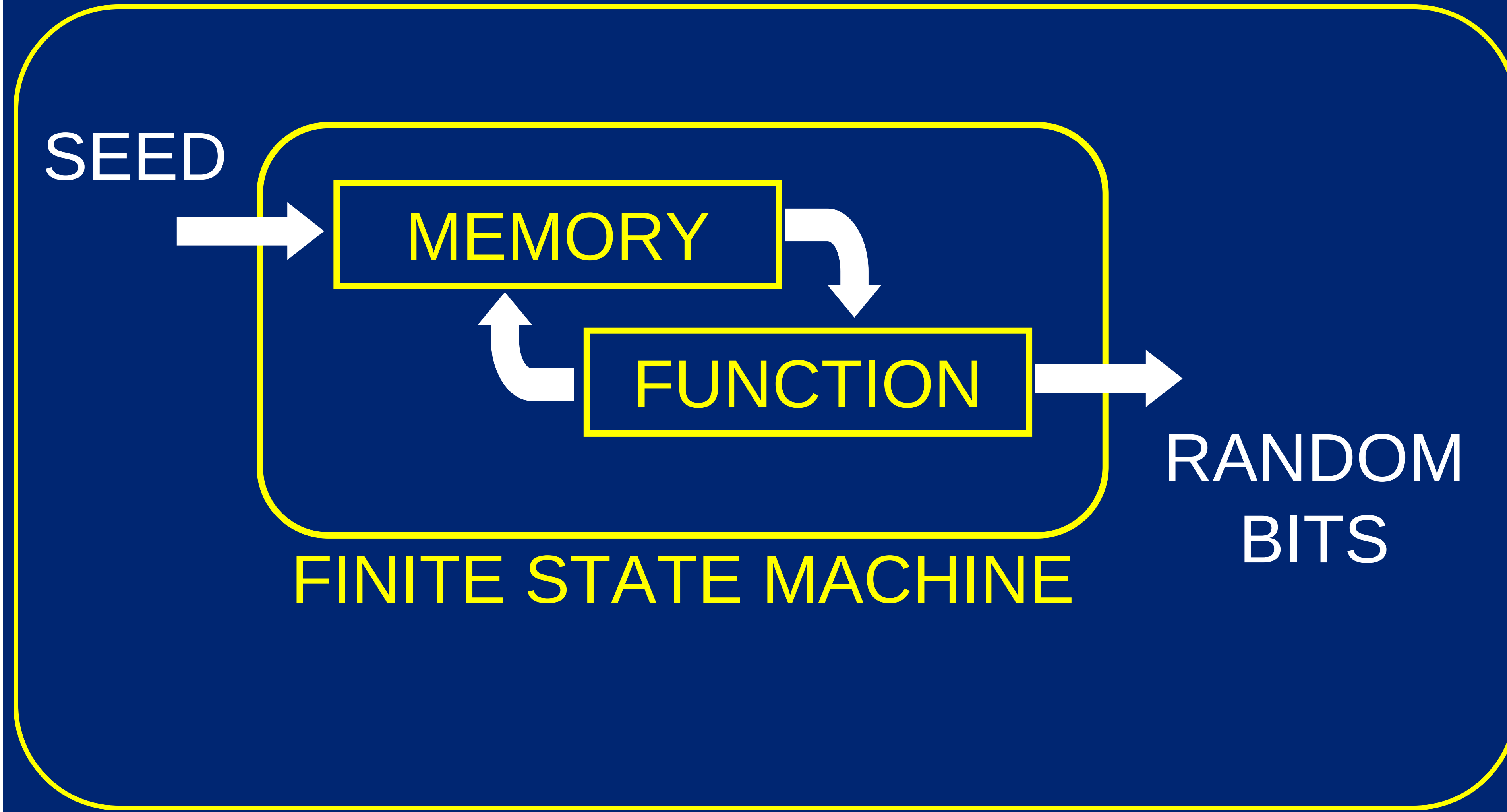
Observations

- Standard PRNGs are all “good” enough
- We could affect the SLS algorithm performance with biased streams (but they were *really* biased)
- With cycled streams, we could get the algorithms to become “stuck”

PAC Property of SLS Algorithms

- Many SLS algorithms are Probabilistically Approximate Complete (PAC)
  - Will solve a soluble instance with arbitrarily high probability when allowed to run long enough
- CRWALK & Adaptive Novelty+ are both PAC
- Even though the algorithms were PAC, we could make them “incomplete” with a poor RNG

Model of a Pseudo-RNG

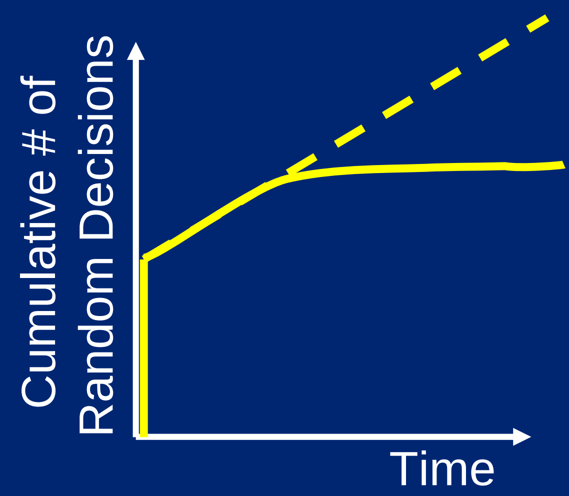


Conclusions

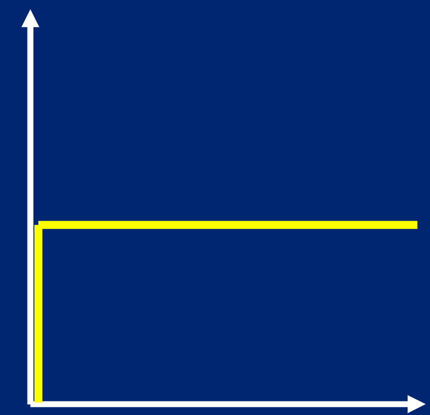
- Since all PRNGs eventually cycle, no conventional algorithm implementation is truly PAC
- Desired PRNG features
  - Reasonably “good” score on a statistical quality test
  - Long cycle period
  - Efficiency
  - Platform independence
- Mersenne Twister; period is $(2^{19937} - 1)$

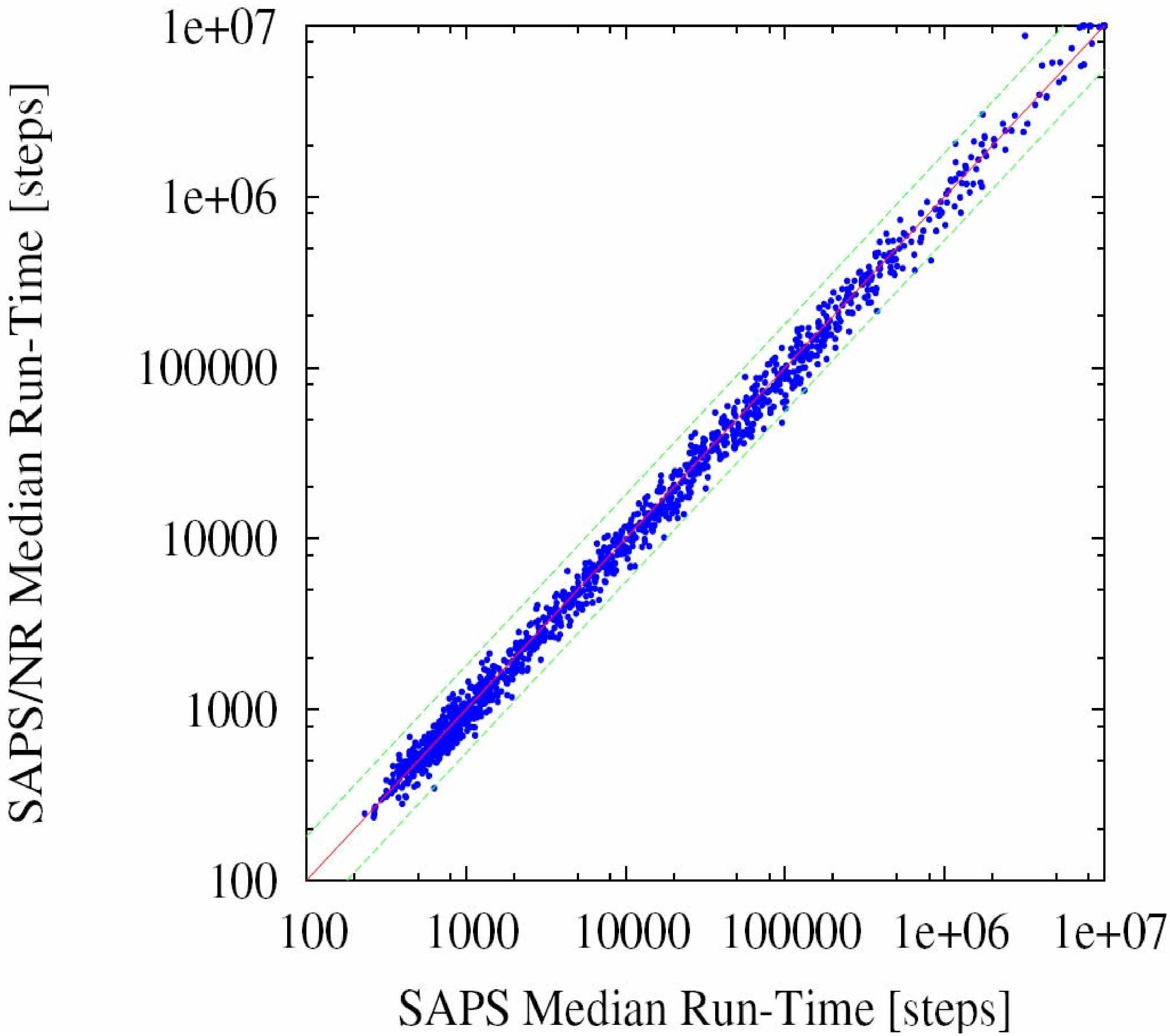
Quantity of Random Decisions

- Previous Observation:
  Scaling and Probabilistic Smoothing (SAPS) algorithm essentially becomes deterministic after initial search phase



- We derandomized the algorithm: SAPS/NR
  [Tompkins, Hoos 2004]





SAPS/NR Median Run-Time [steps]
1e+07
1e+06
100000
10000
1000
100
100
1000
10000
100000
1e+06
1e+07
SAPS Median Run-Time [steps]

Derandomization

- Can we achieve similar results with algorithms that rely more heavily on random decisions?
- We developed derandomized versions of CRWALK and Adaptive Novelty+
- Used *straightforward* derandomization methods

Derandomized CRWALK

- BEFORE:
  - – Select unsatisfied clause at random
  - – Select variable to flip at random

- DERANDOMIZED:
  - – Select clause with the lowest value of: (# times selected / # times unsat)
    - Breaking ties with the “first” clause
  - – Select variable to flip in sequential order

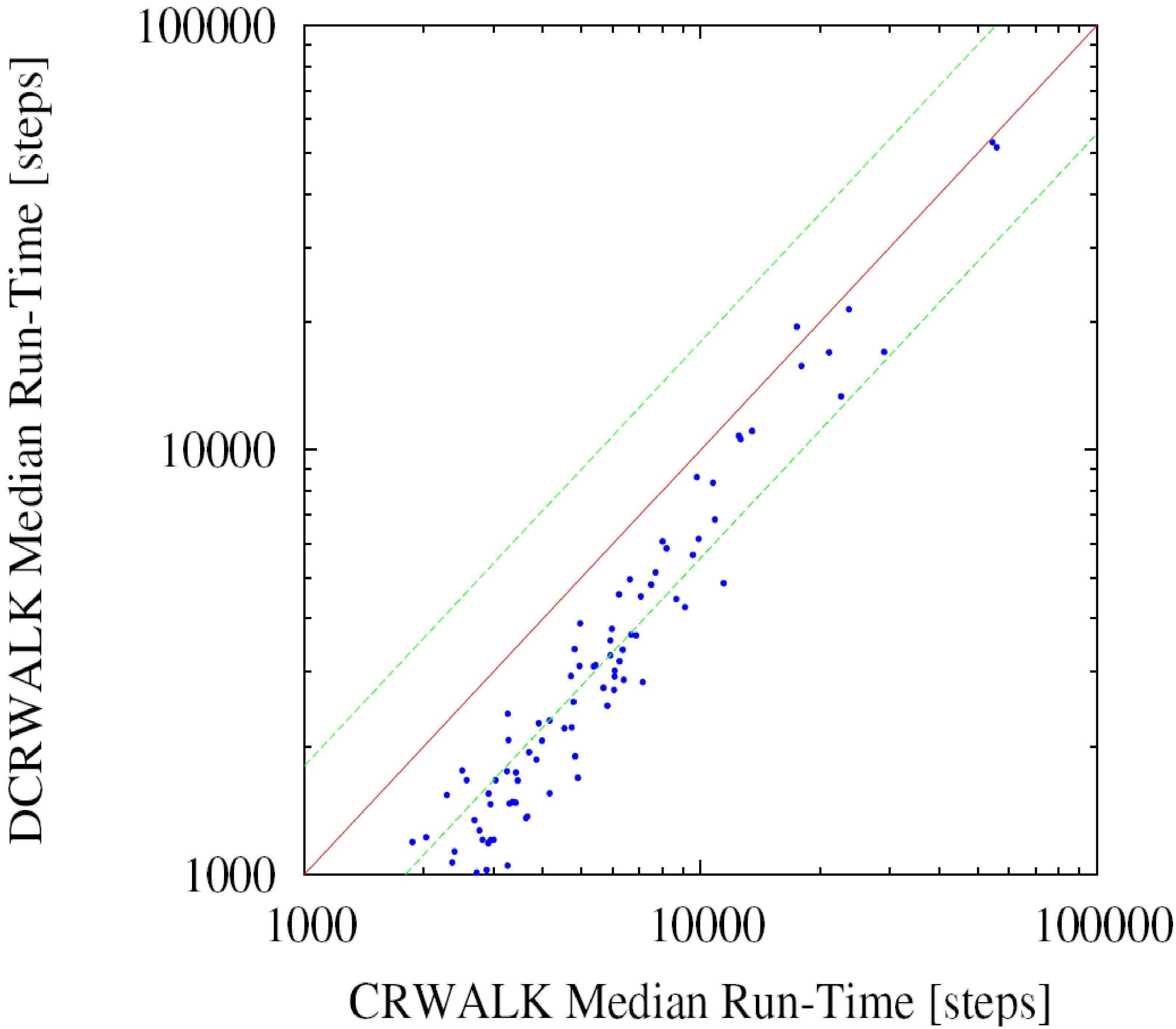
DCRWALK (Deterministic)

- a=T, b=T, c=T, d=F, e=F…
- (a v b v ¬c) (a v ¬b v d) (¬a v d v e)…

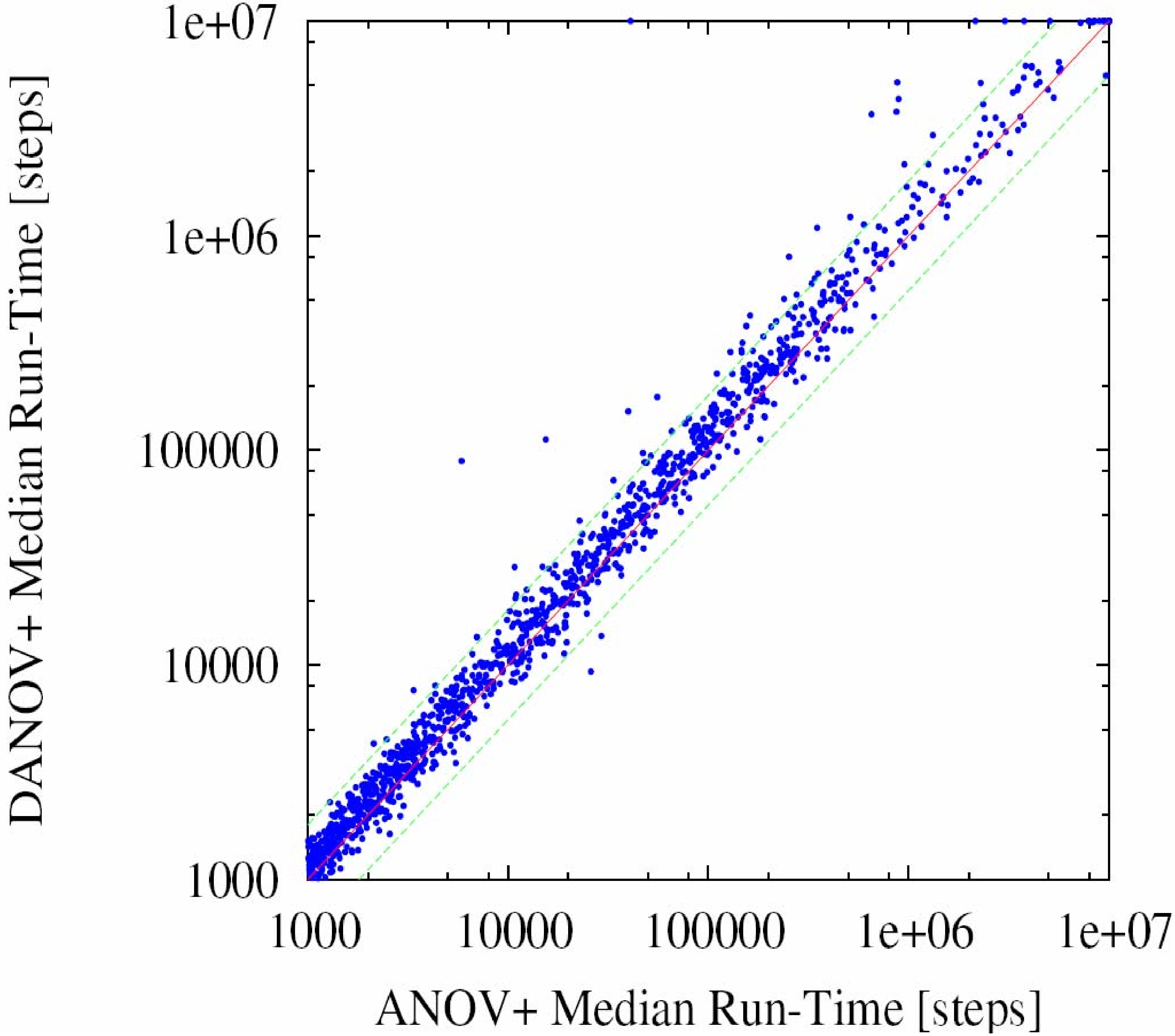
- a=F, b=T, c=T, d=F, e=F…
- (a v b v ¬c) (a v ¬b v d) (¬a v d v e)…

- a=T, b=T, c=T, d=F, e=F…
- (a v b v ¬c) (a v ¬b v d) (¬a v d v e)…

- a=F, b=T, c=T, d=T, e=F…
- (a v b v ¬c) (a v ¬b v d) (¬a v d v e)…

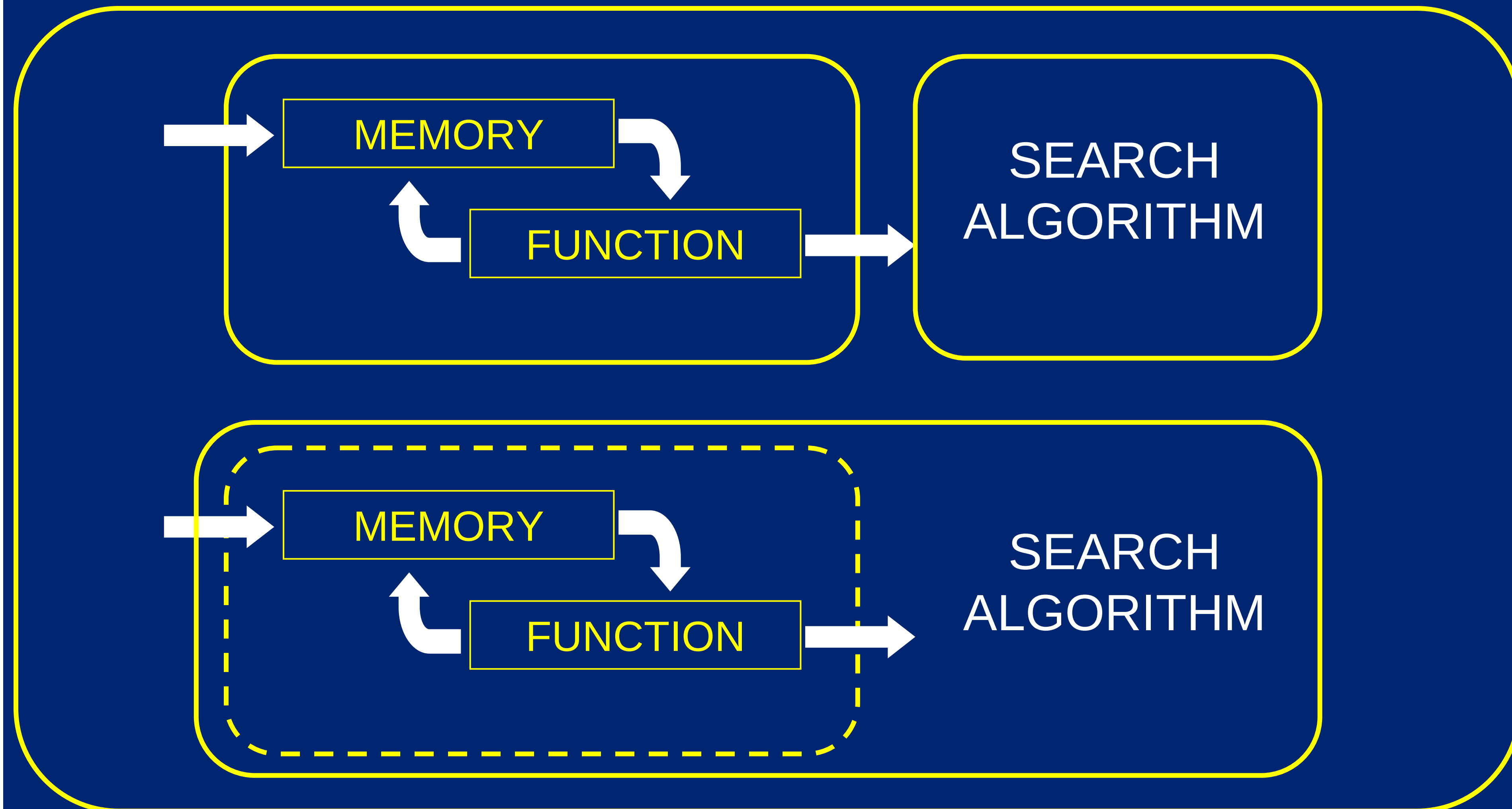


100000
10000
1000
1000
10000
100000
DCRWALK Median Run-Time [steps]
CRWALK Median Run-Time [steps]



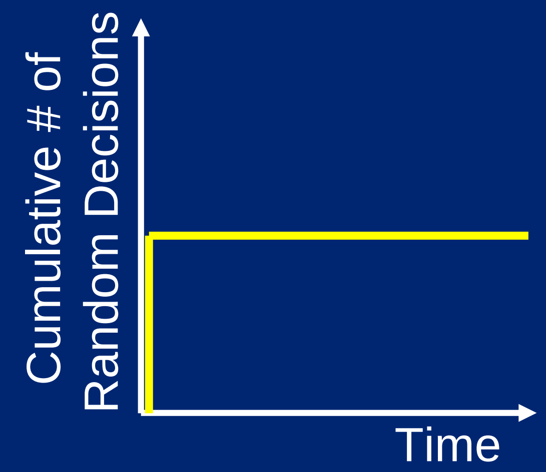
DANOV+ Median Run-Time [steps]
1e+07
1e+06
100000
10000
1000
1000
10000
100000
1e+06
1e+07
ANOV+ Median Run-Time [steps]

Advanced Derandomization

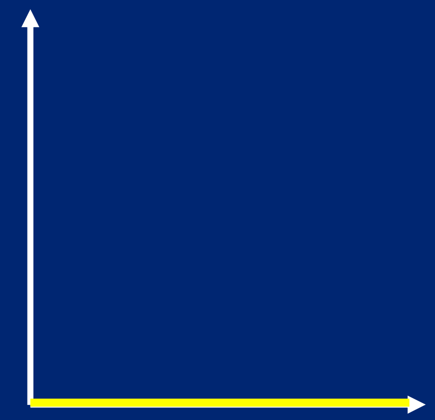


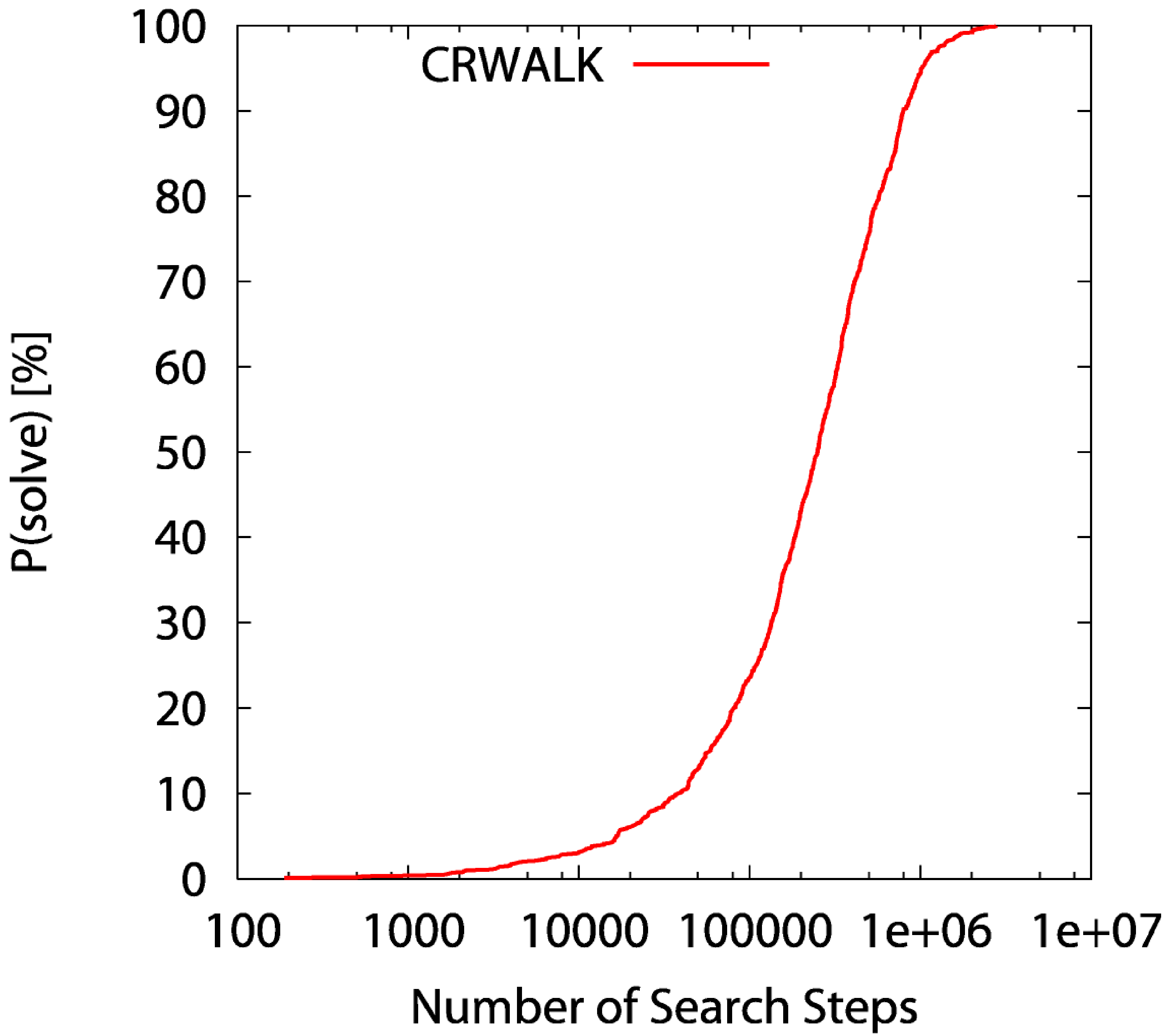
Quantity of Random Decisions

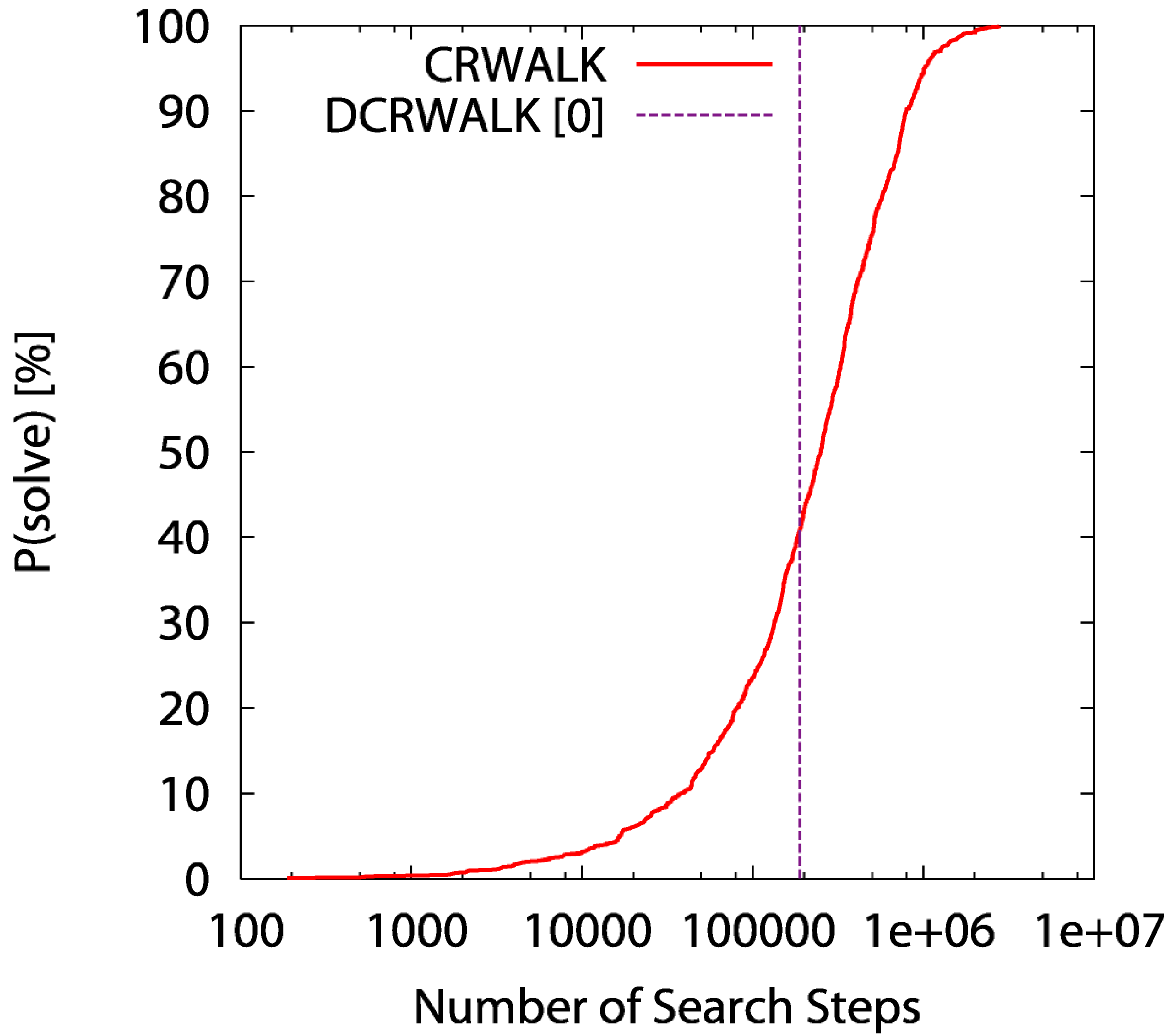
- Deterministic Algorithms

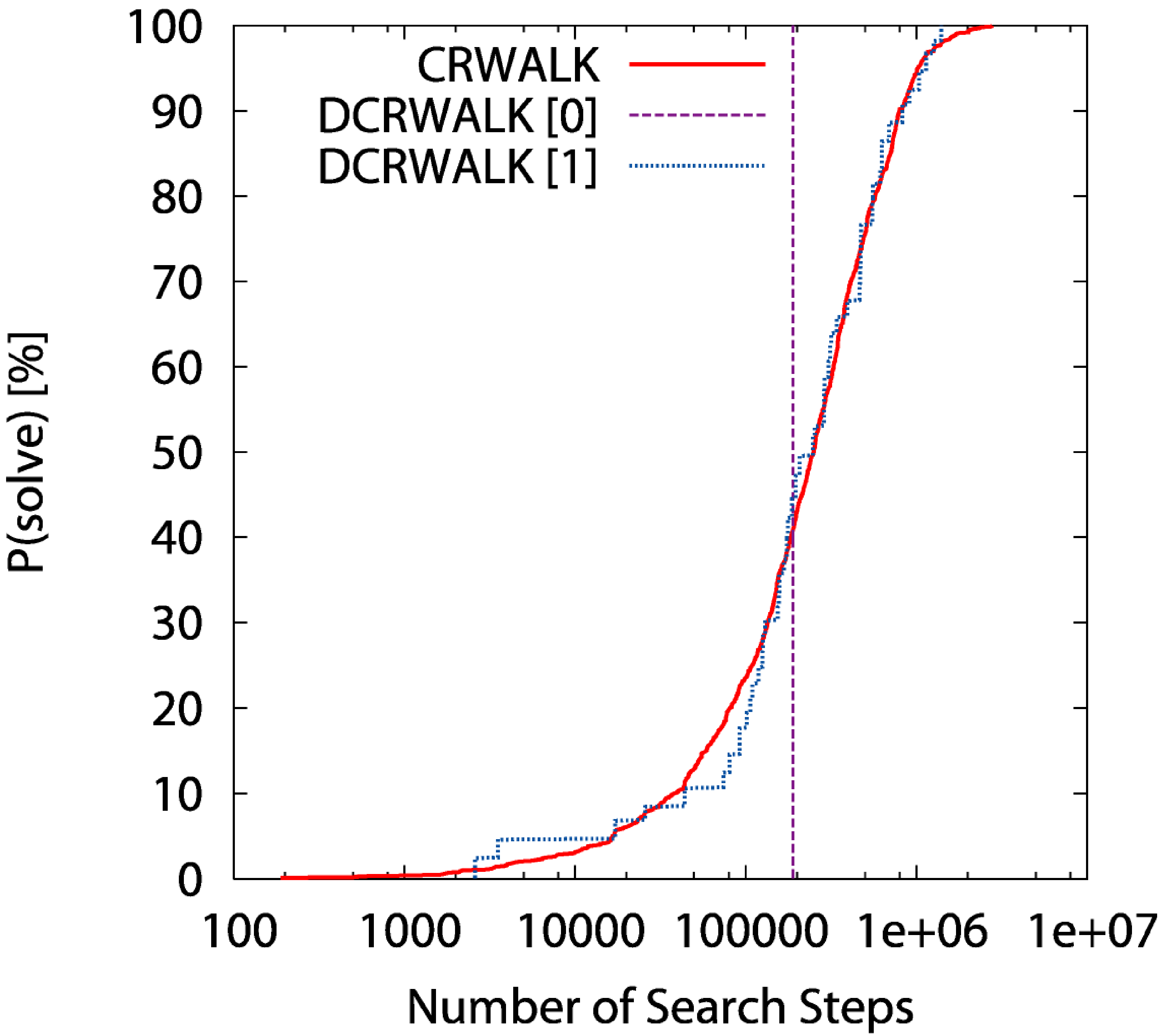


- Deterministic Initialization

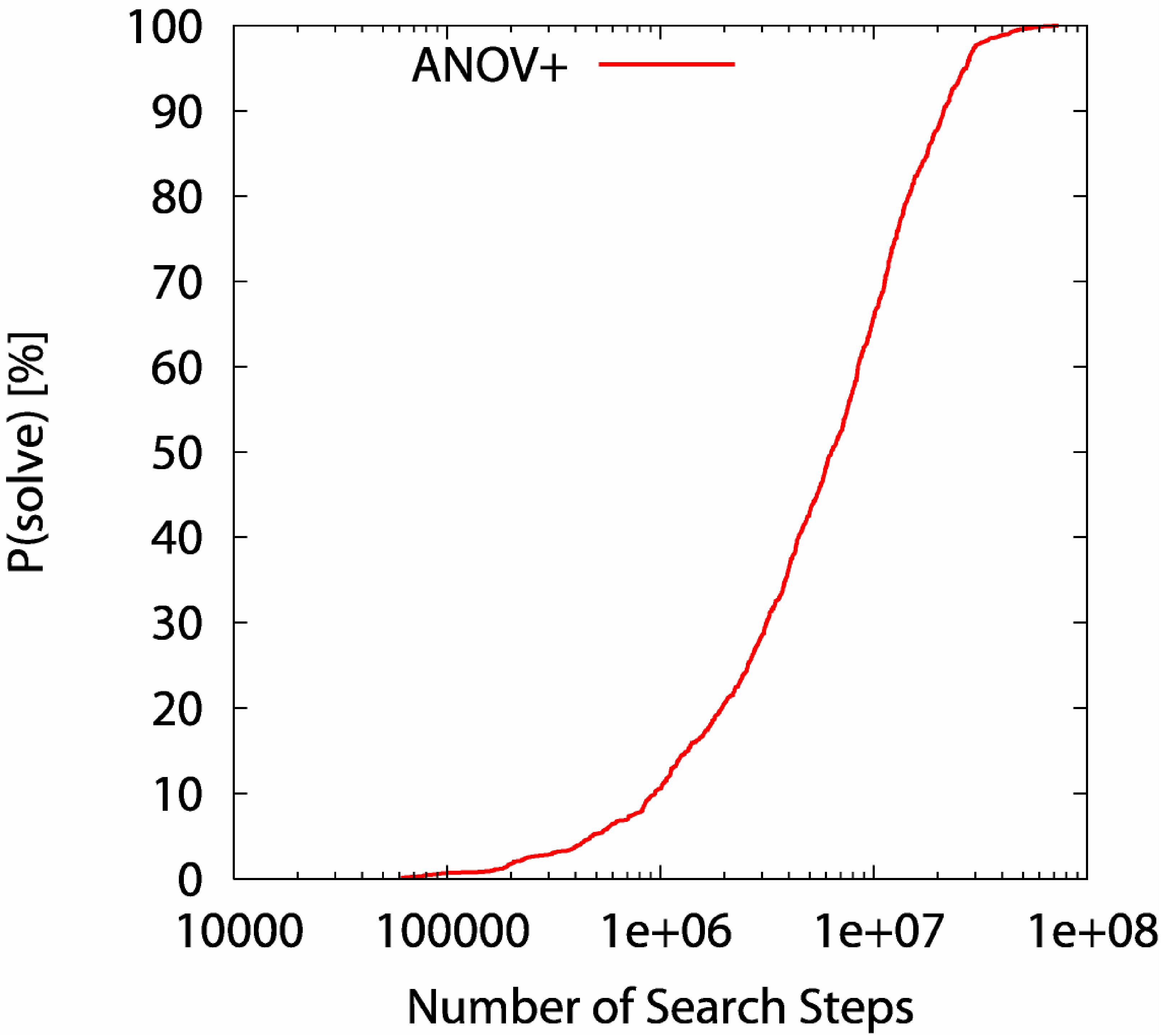


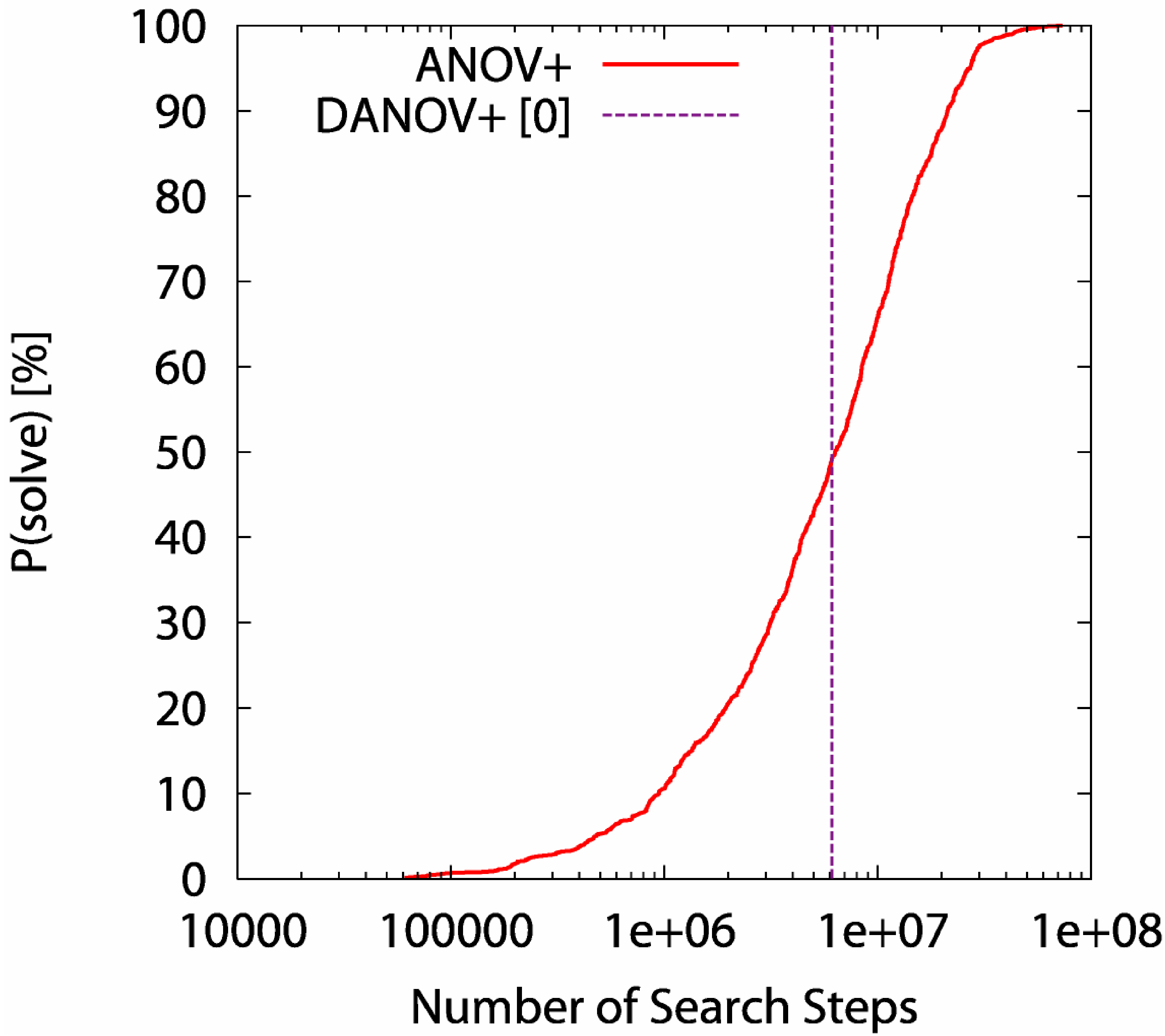

CRWALK
P(solve) [%]
100
90
80
70
60
50
40
30
20
10
0
100
1000
10000
100000
1e+06
1e+07
Number of Search Steps


CRWALK
DCRWALK [0]
P(solve) [%]
100
90
80
70
60
50
40
30
20
10
0
100
1000
10000
100000
1e+06
1e+07
Number of Search Steps

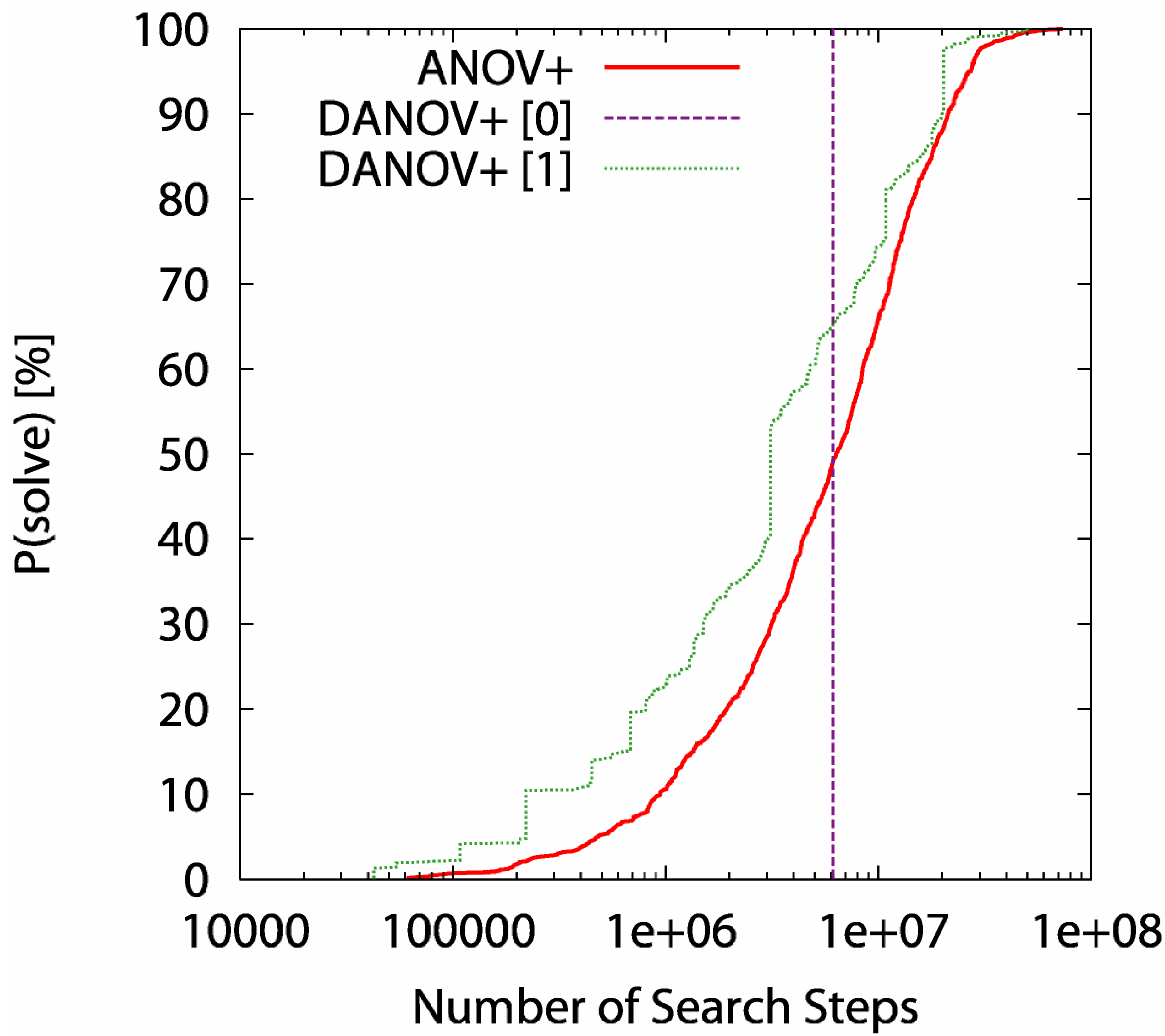


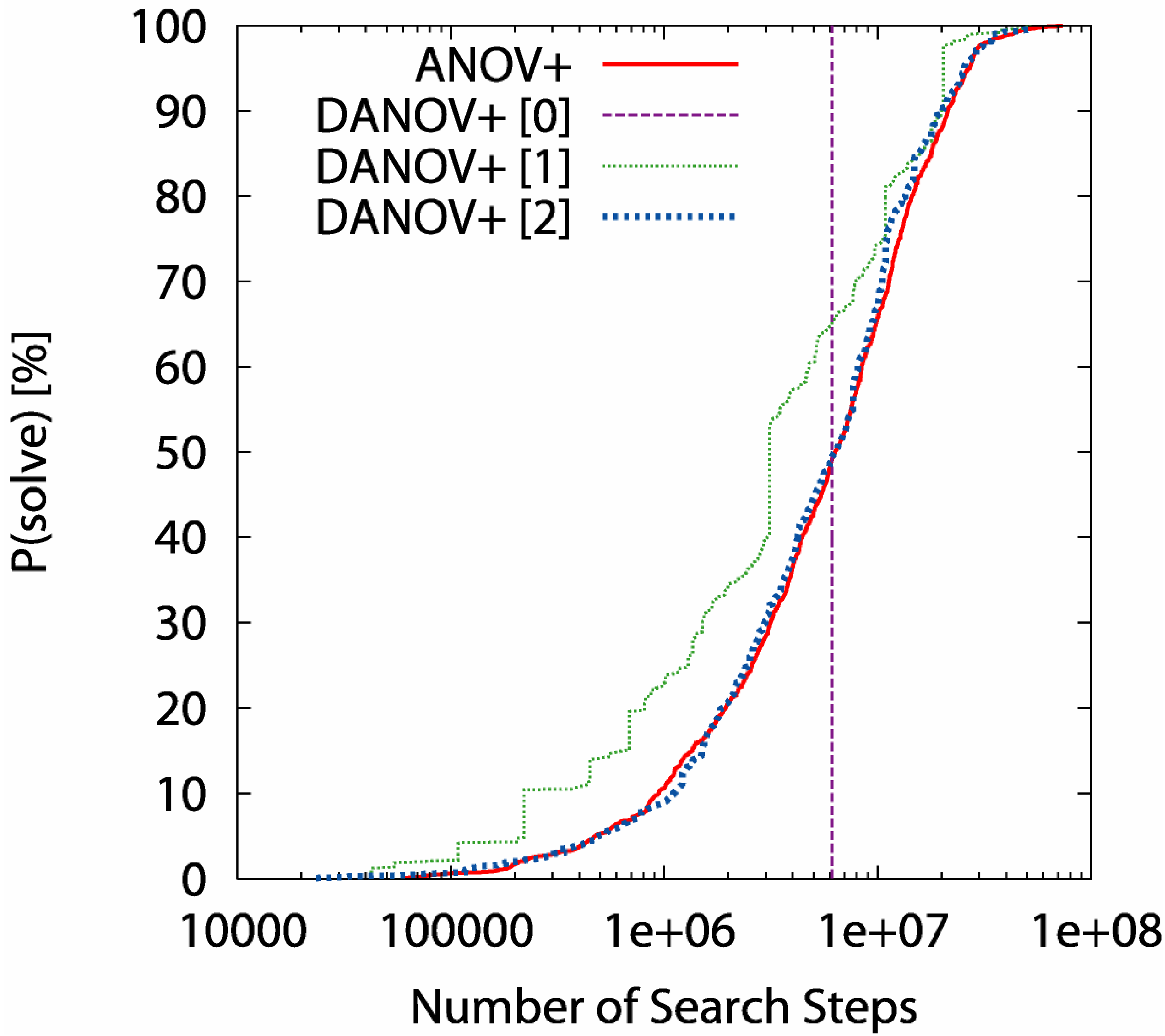
CRWALK
DCRWALK [0]
DCRWALK [1]
P(solve) [%]
0
10
20
30
40
50
60
70
80
90
100
100
1000
10000
100000
1e+06
1e+07
Number of Search Steps


ANOV+
P(solve) [%]
100
90
80
70
60
50
40
30
20
10
0
10000
100000
1e+06
1e+07
1e+08
Number of Search Steps



ANOV+
DANOV+ [0]
P(solve) [%]
100
90
80
70
60
50
40
30
20
10
0
10000
100000
1e+06
1e+07
1e+08
Number of Search Steps


ANOV+
DANOV+ [0]
DANOV+ [1]
P(solve) [%]
0
10
20
30
40
50
60
70
80
90
100
10000
100000
1e+06
1e+07
1e+08
Number of Search Steps



ANOV+
DANOV+ [0]
DANOV+ [1]
DANOV+ [2]
P(solve) [%]
0
10
20
30
40
50
60
70
80
90
100
10000
100000
1e+06
1e+07
1e+08
Number of Search Steps

Conclusions & Future Work

- SLS algorithms are very robust w.r.t. the quality of the random number generator
- With straightforward implementations, a surprisingly few number of random decisions can exhibit full variability
- Future Work
  - Other domains & algorithms
  - Time analysis of PRNGs & randomized vs. deterministic
  - Statistical outliers: investigate for further insight

Thank you

BONJOUR
MON NOM EST:

Dave
Tompkins

Questions?