Warped Landscapes and
Random Acts of SAT Solving

Dave A.D. Tompkins & Holger H. Hoos

Department of Computer Science
University of British Columbia
Canada

Outline

1. Dynamic Local Search (DLS) for SAT and MAX-SAT
2. Do DLS Algorithms Learn?
3. Is Randomness Needed?
4. Conclusions & Future Work

Dynamic Local Search (DLS) for (MAX-)SAT

Propositional Satisfiability Problem (SAT):

Given: Propositional formula Φ in conjunctive normal form.

Objective: Find an assignment of truth values to variables in Φ
such that Φ is satisfied, or declare Φ as unsatisfiable.

Example:

$(a \vee b) \wedge (\neg a \vee \neg b)$

$\rightsquigarrow$ satisfiable, solution: $a = \text{true}, b = \text{false}$

Maximum Propositional Satisfiability Problem (MAX-SAT):

Given: Propositional formula Φ in conjunctive normal form.

Objective: Find an assignment of truth values to variables in Φ
that *maximises the number of satisfied clauses* in Φ.

Weighted MAX-SAT:

Given: Propositional formula Φ in conjunctive normal form,
weights $w(c)$ associated with each clause $c \in \Phi$

Objective: Find an assignment of truth values to variables in Φ
that maximises the *total weight* of satisfied clauses in Φ.

$\leadsto$ hard *vs.* soft constraints

Stochastic Local Search (SLS)

Approach:

- Guess (*i.e.*, randomly generate) *initial candidate solution* (SAT: randomly determine truth value for each variable).
- Iteratively perform *search steps* by modifying small parts of the candidate solution guided by *evaluation function* (SAT: pick a variable and change its truth value in order to reduce number of unsatisfied clauses).
- Stop this process when *termination condition* is satisfied, *e.g.*, solution found or time-limit reached.
- Stochastic decisions are used to overcome / avoid search stagnation caused by, *e.g.*, local minima.

Note:

- SLS algorithms are amongst the best-performing methods for solving hard, satisfiable SAT instances.
- SLS algorithms are (by a large margin) the best-performing methods for solving hard MAX-SAT instances.

Dynamic Local Search (DLS)

Key idea: Modify evaluation function during search process to escape from local minima in objective function g.

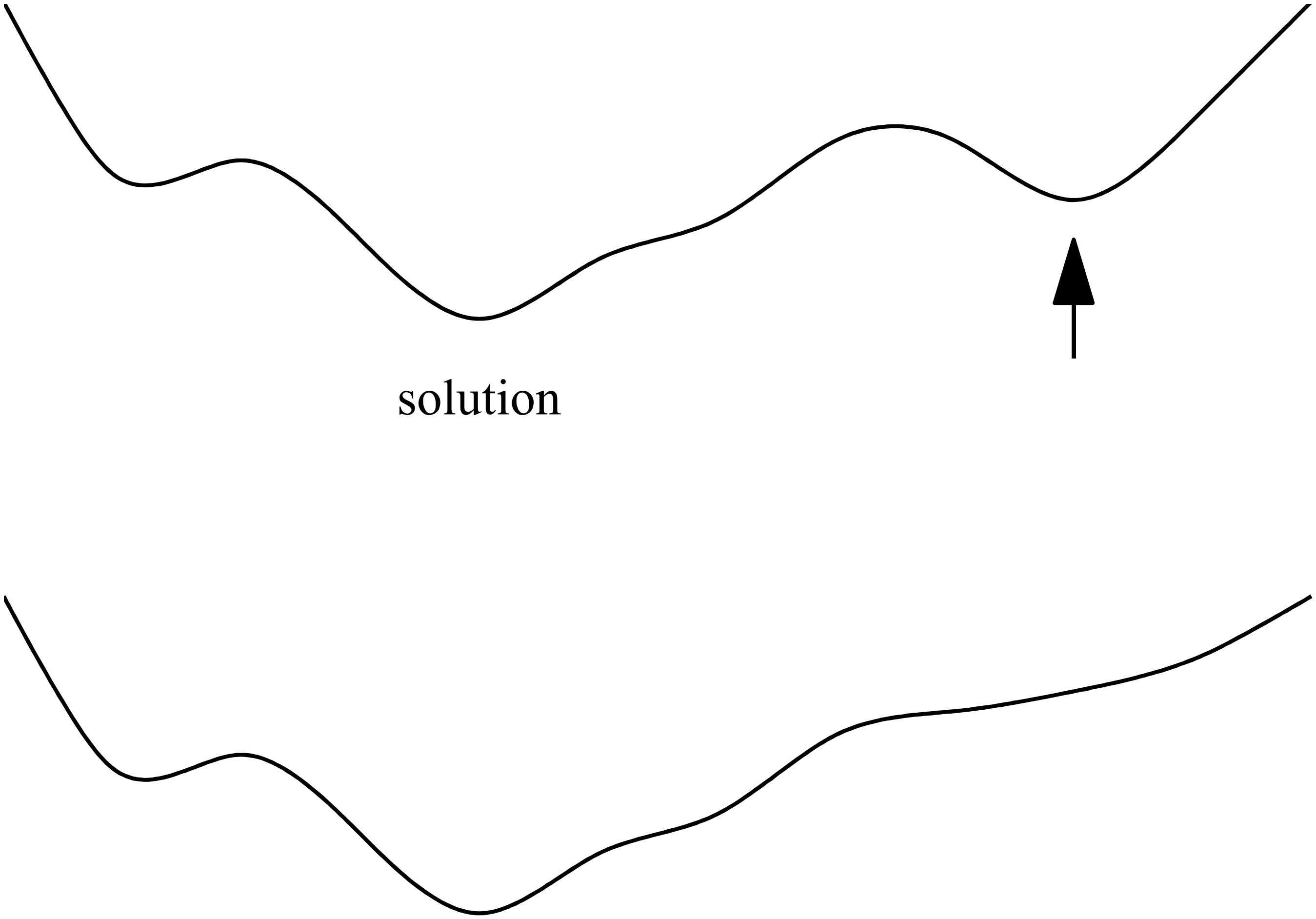
DLS for SAT:

- associate *penalty values* $clp(c)$ with every clause c
- initialise clause penalties (typically $clp(c) := 1$)
- perform local search on

$$g'(clp, a) := \sum_{c \text{ is unsat under } a} clp(c)$$

- modify clause penalties (important choices: when? how?)

Dynamic Local Search



Note:

- DLS for SAT effectively finds locally optimal solutions for a series of weighted MAX-SAT instances, where the clause weights correspond to the clp values.
- Many DLS algorithms are motivated by methods from continuous optimisation, but important theoretical properties do not carry over.
- Modifications of clause weights typically have high time complexity compared to local search steps.

Some DLS Algorithms for SAT

– Breakout Method [Morris, 1993]

∗ GSAT with clause weights [Selman & Kautz, 1993]

– GSAT with rapid weight adjustment [Frank, 1997]

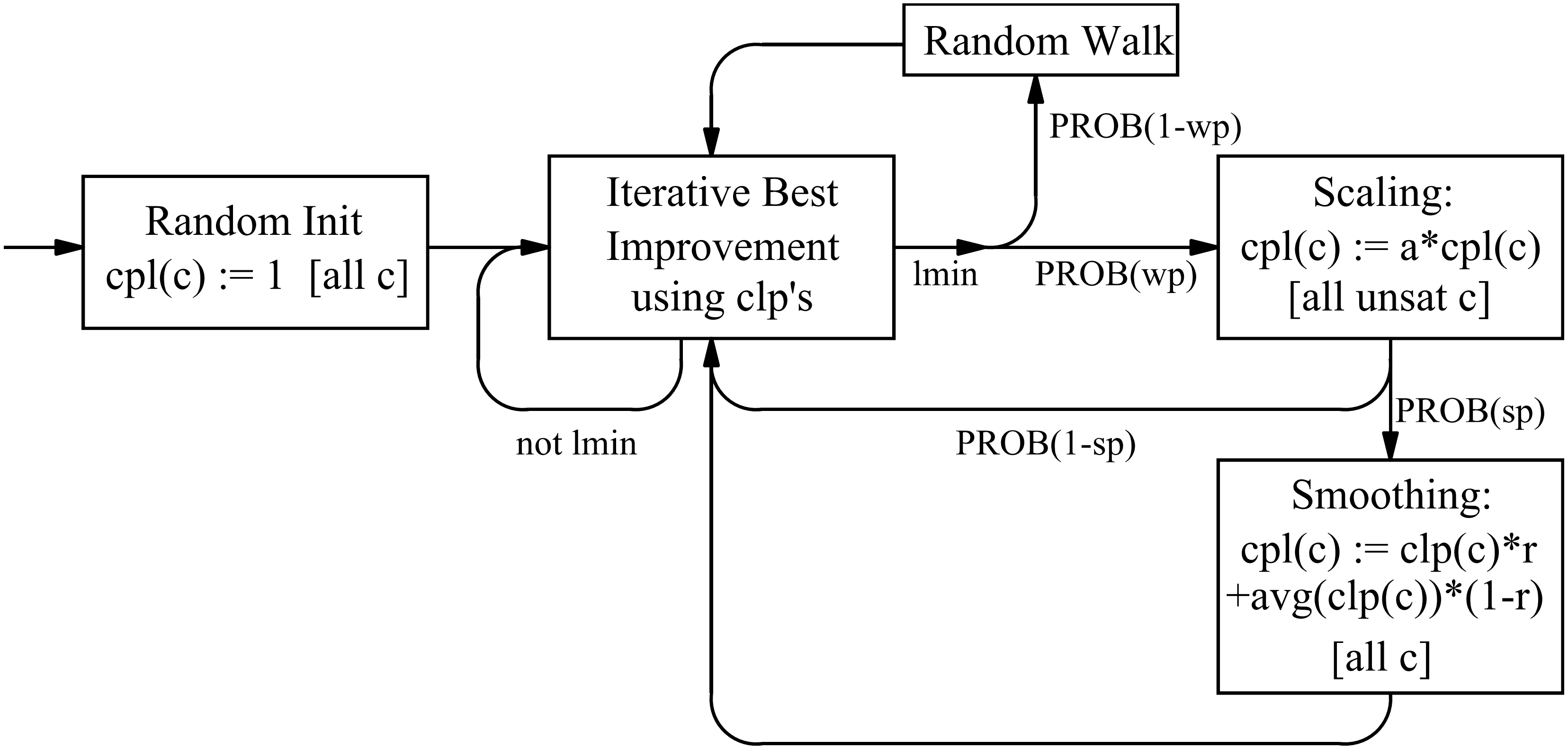
∗ Discrete Lagrangian Method (DLM) [Wah *et al.*, 1998-2000]

– Smoothed Descent and Flood (SDF) algorithm
[Schuurmans & Southy, 2000]

∗ Exponentiated Subgradient (ESG) algorithm
[Schuurmans *et al.*, 2001]

∗∗ Scaling and Probabilistic Smoothing (SAPS) algorithm
[Hutter, Tompkins, & Hoos, 2002]

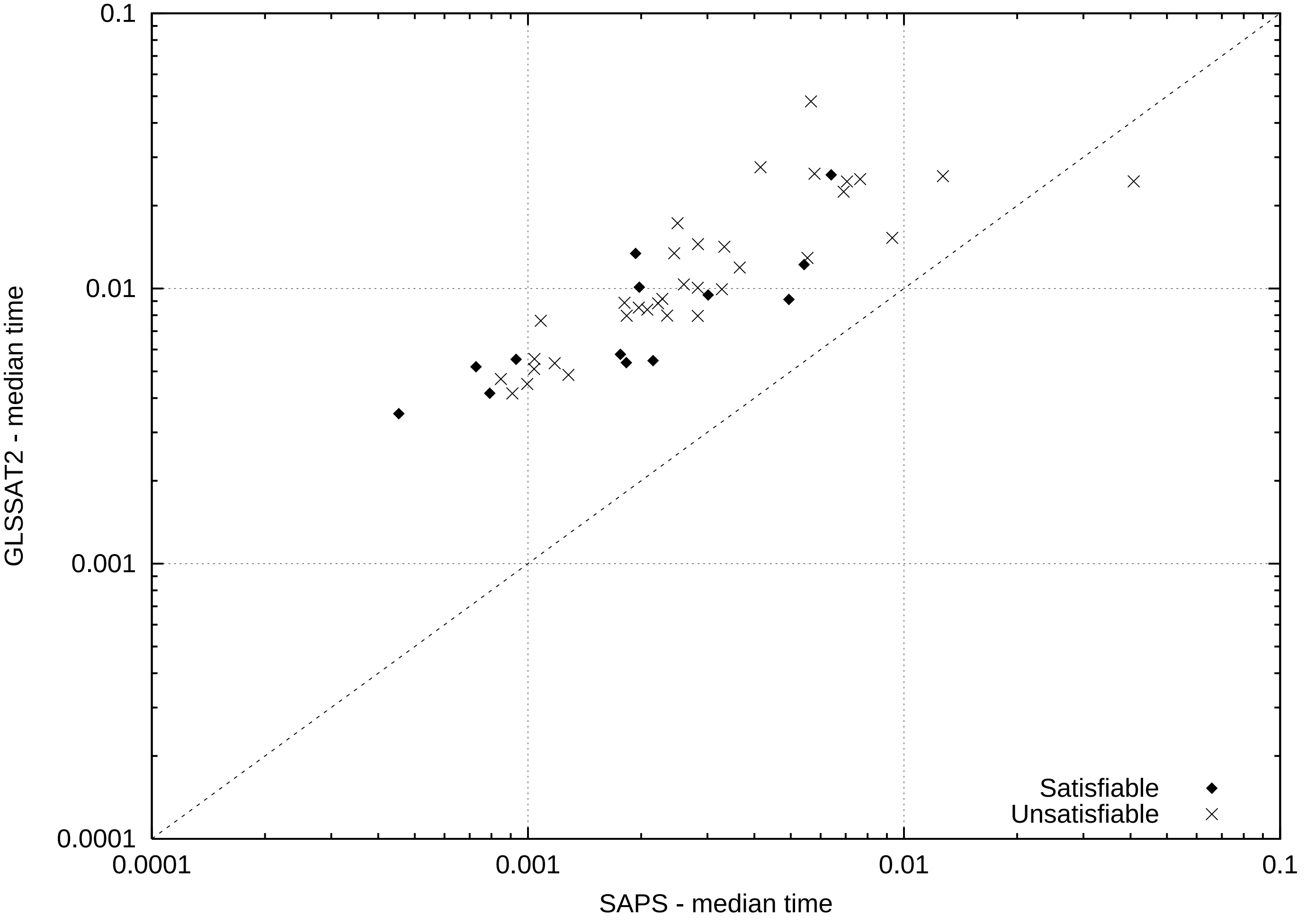
Scaling And Probabilistic Smoothing (SAPS)



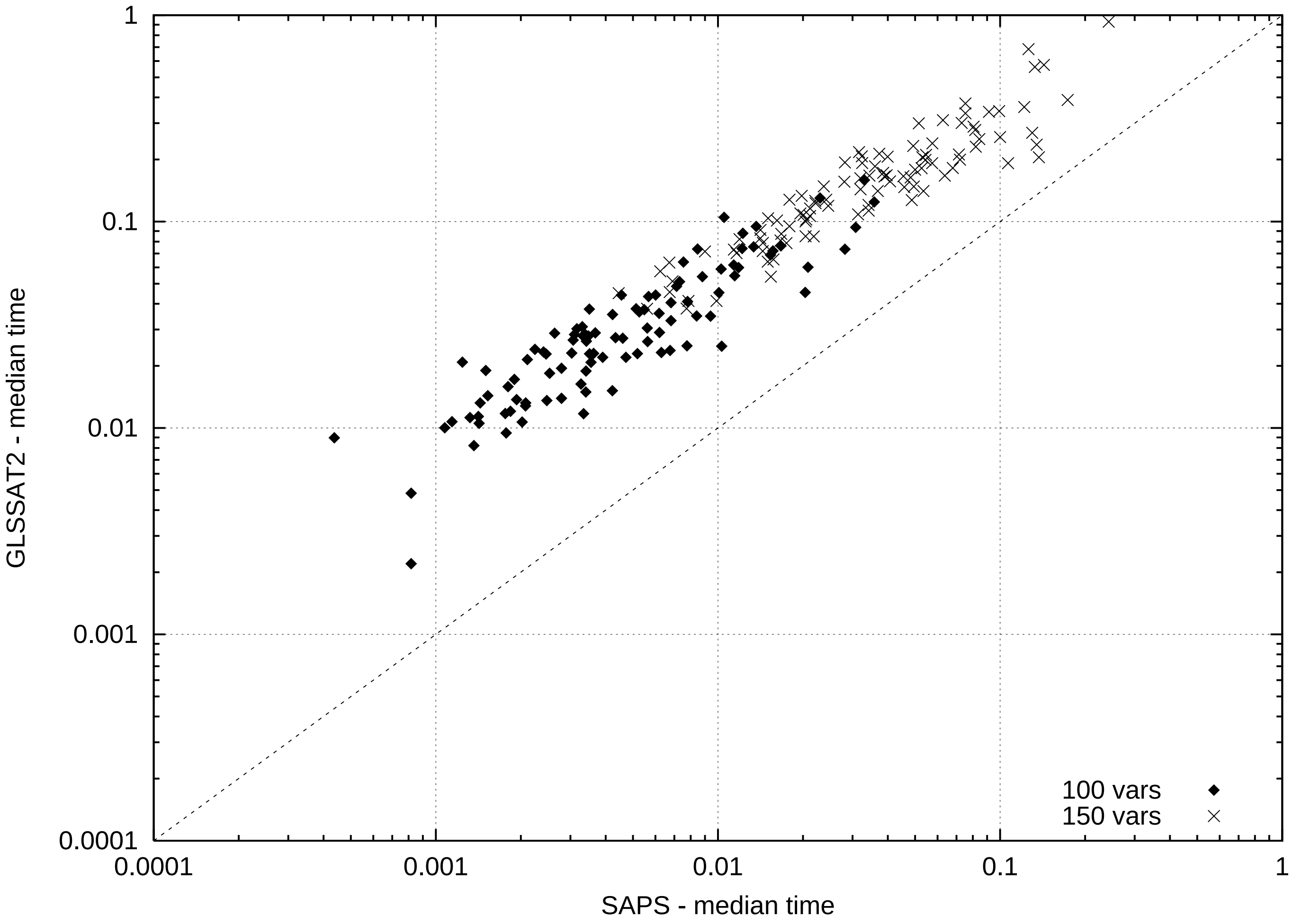
SAPS on SAT (median run-time in CPU sec)

Problem Instance	Novelty$^+$	ESG	SAPS	*s.f.*
uf100-hard	0.046	0.006	0.006	1.00
uf250-med	0.015	0.0195	0.011	1.36
uf250-hard	2.745	0.461	0.291	1.58
uf400-med	0.160	0.324	0.103	1.55
uf400-hard	22.3	9.763	1.973	4.95
flat100-med	0.008	0.013	0.008	1.00
flat100-hard	0.089	0.037	0.032	1.16
flat200-med	0.208	0.237	0.087	2.39
flat200-hard	18.862	5.887	3.052	1.93
bw_large.a	0.014	0.016	0.009	1.56
bw_large.b	0.339	0.280	0.179	1.56
logistics.c	0.226	0.229	0.037	6.10
ais10	4.22	0.139	0.051	2.73

SAPS on MAX-SAT: test-set wjnh



SAPS on MAX-SAT: test-sets rnd100-1000u, rnd150-1500u



Do DLS Algorithms Learn?

Original motivation of DLS:

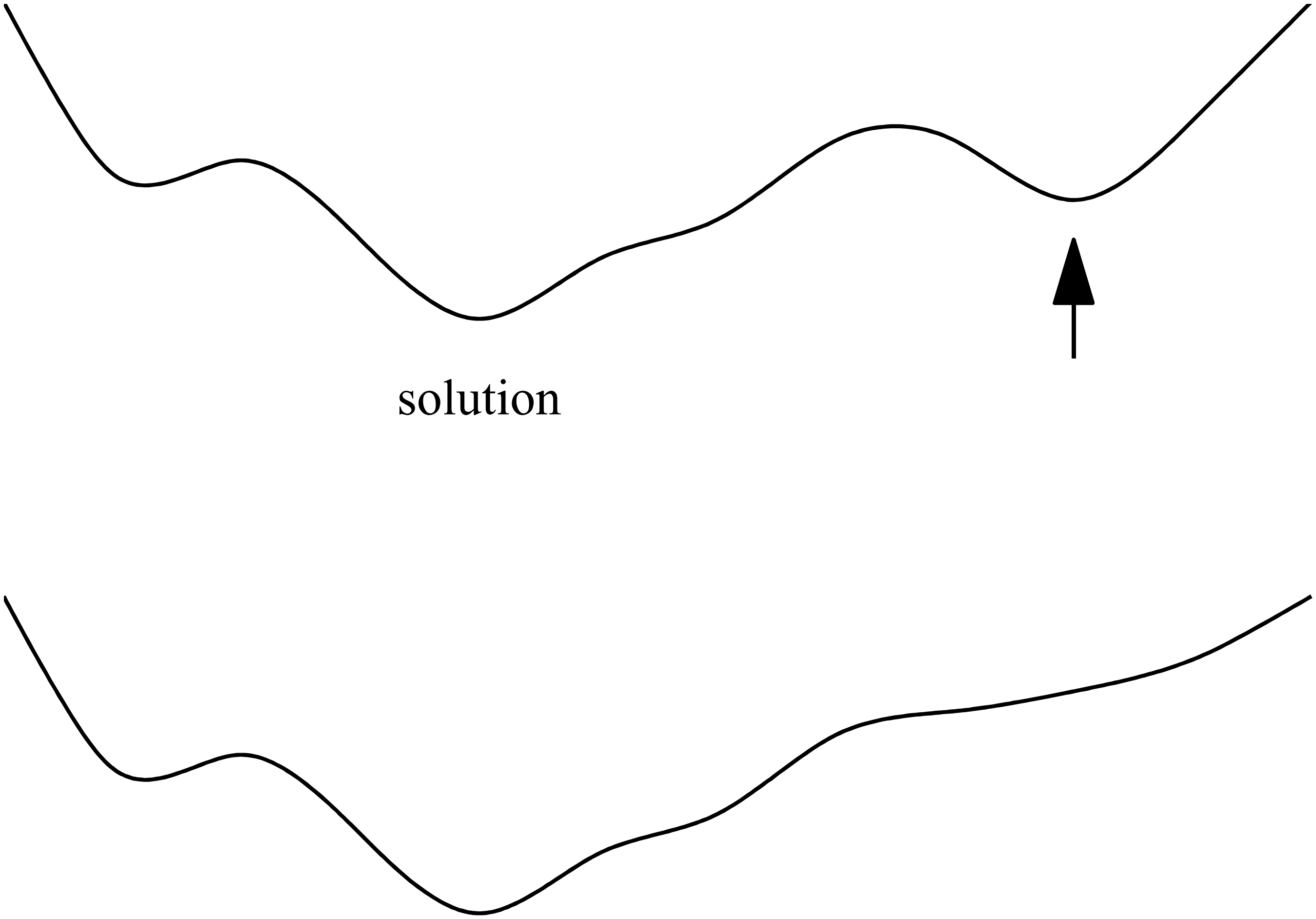
- Fill in local minima
- Learn important / hard clauses

$\leadsto$ **Hypothesis:**

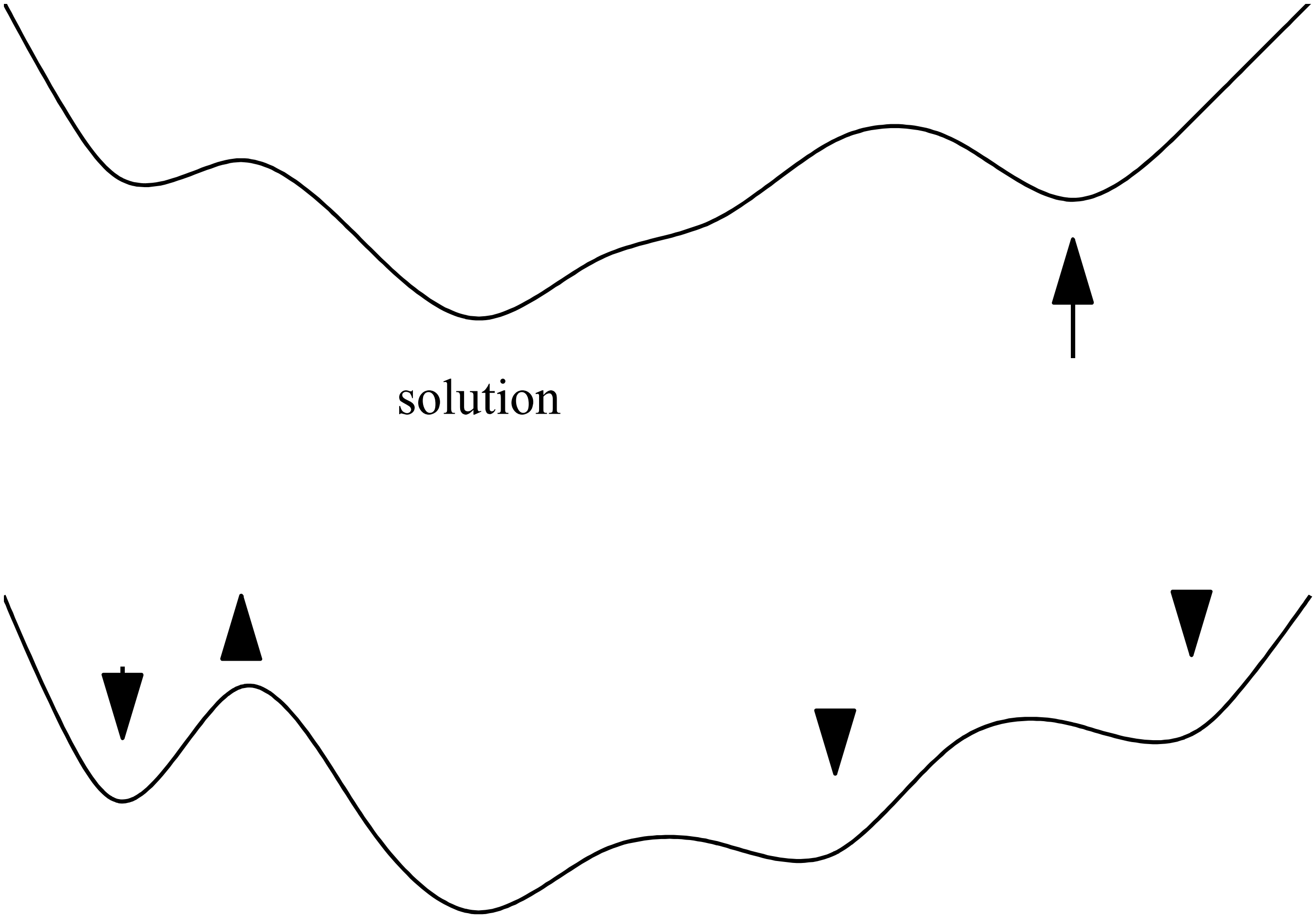
Clause penalties determined by DLS algorithm
render problem instance easier to solve

Note: This hypothesis was never tested!

Dynamic Local Search



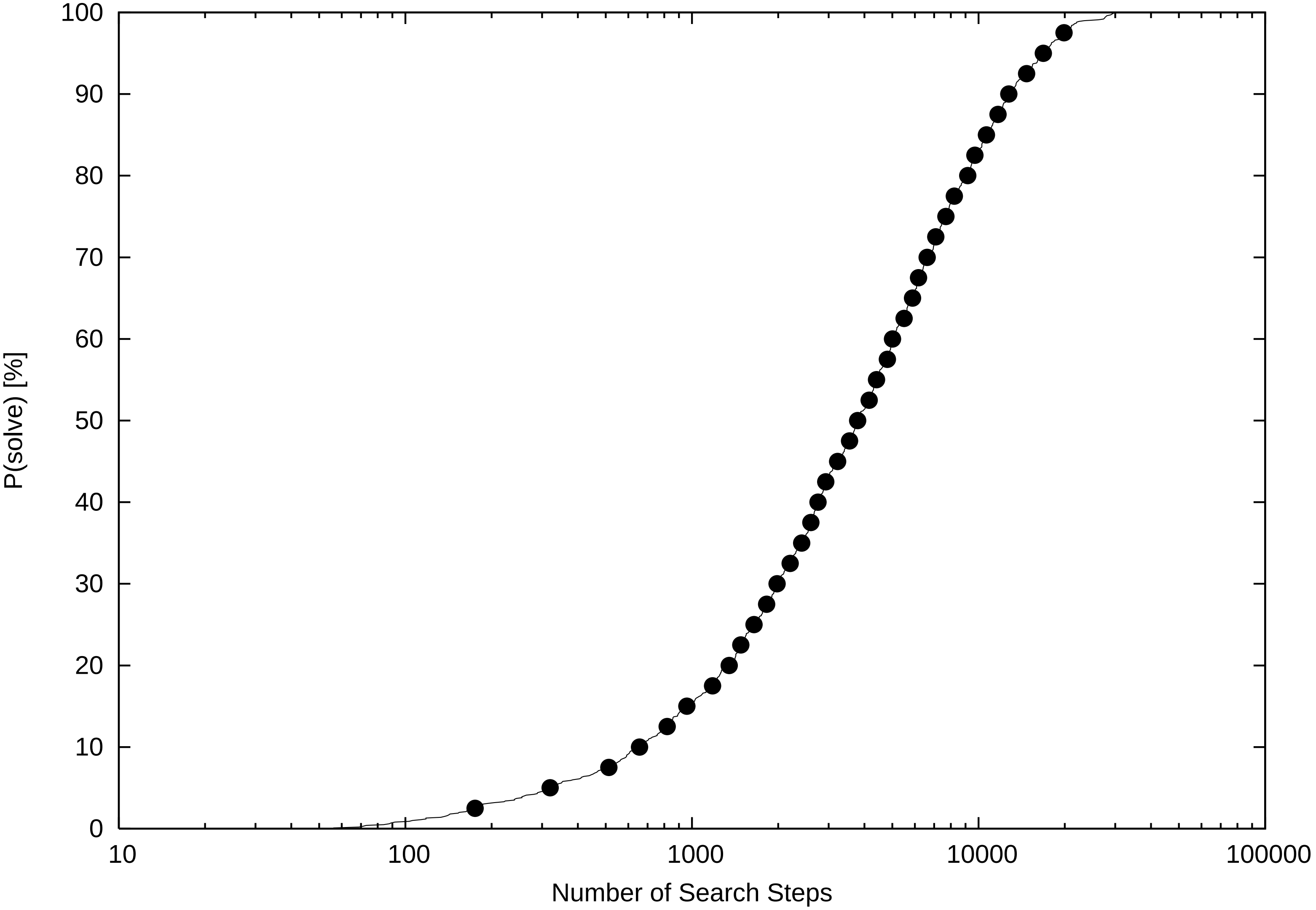
Dynamic Local Search



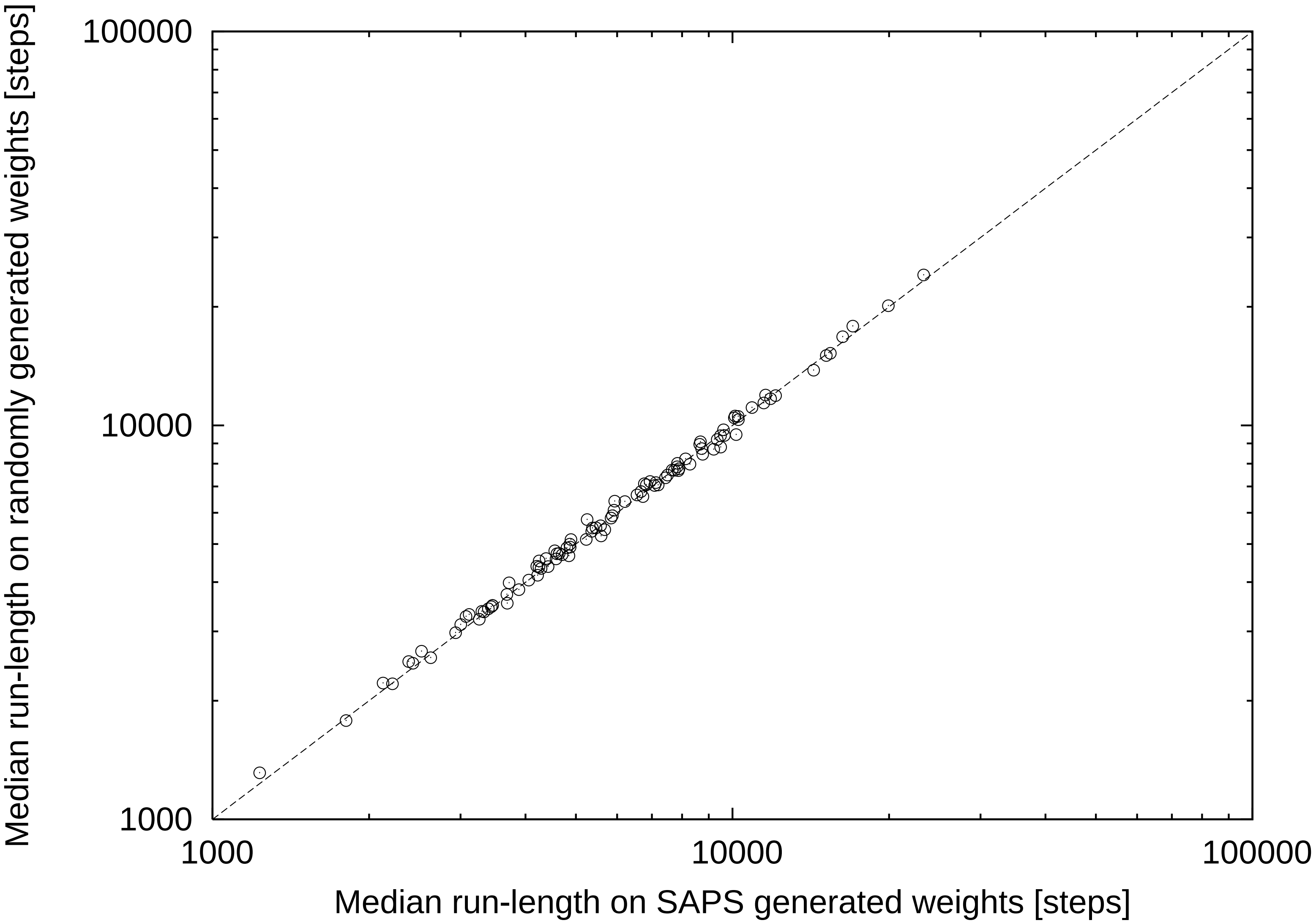
Experiment:

1. Solve benchmark instances using SAPS;
   measure search cost (median # variable flips).
2. Take snapshots of clause penalty values
   at end of characteristic successful runs.
3. Initialise clause penalties according to snapshots;
   measure search cost for SAPS.
4. Initialise clause penalties randomly;
   measure search cost for SAPS.
5. Analyse differences in search cost
   for “learned” and random penalties.

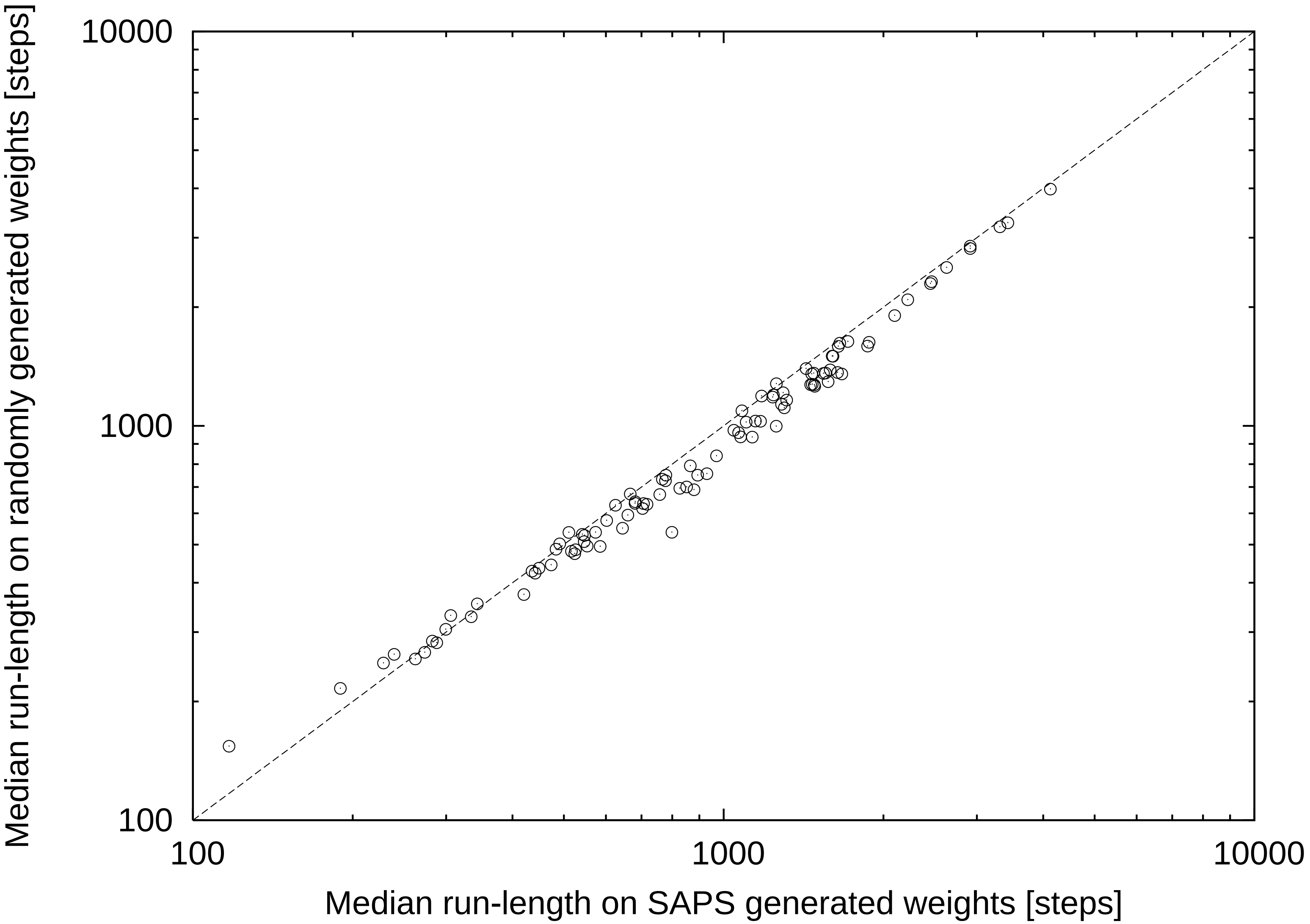
Selecting Characteristic Runs



Flat100: SAPS-generated *vs.* random weights



UF100: SAPS-generated *vs.* random weights



Instance	Unweighted	SAPS Generated Weighted Instances			Randomly Generated Weighted Instances		
		$q_{0.25}$	Median	$q_{0.75}$	$q_{0.25}$	Median	$q_{0.75}$
uf100-easy	81	0.98	1.01	1.06	1.31	1.36	1.46
uf100-hard	3,763	1.08	1.11	1.14	1.03	1.06	1.10
uf250-hard	197,044	0.98	1.06	1.14	0.97	1.03	1.06
uf400-hard	2,948,181	0.92	1.04	1.17	0.95	1.10	1.19
flat100-hard	24,248	0.99	1.02	1.04	0.98	1.01	1.04
bw_large.a	2,499	0.90	0.93	0.98	1.01	1.04	1.07
bw_large.b	34,548	0.97	1.02	1.08	0.99	1.07	1.11
logistics.c	9,446	0.97	1.03	1.06	1.05	1.07	1.14
ssa7552-038	3,960	0.86	0.91	0.95	1.02	1.08	1.12
ais10	20,319	1.06	1.09	1.11	1.04	1.11	1.19

Result:

No support for hypothesis that clause penalties determined by SAPS render problem instances easier.

So . . . why does SAPS work?

- Main effect of scaling: escape from local minimum and avoid being immediately sucked back in.
- *But:* adverse side effects (*e.g.*, very likely new / more local minima) due to large “footprints” of clauses.
- *Hence:* Need mechanism for undoing unwanted effects of scaling $\rightsquigarrow$ smoothing!

Note:

The main role of penalty modifications appears to be *search diversification*, which in many other SLS algorithms is achieved through strong randomisation of the search.

Is Randomness Needed?

Random decisions in SAPS:

1. random initialisation of variable assignment
2. random tie-breaking in subsidiary local search
3. random walk steps (in local minimum)
4. probabilistic smoothing

SAPS/NR:

- deterministic tie-breaking
- no random walk steps ($wp = 0$)
- deterministic periodic smoothing

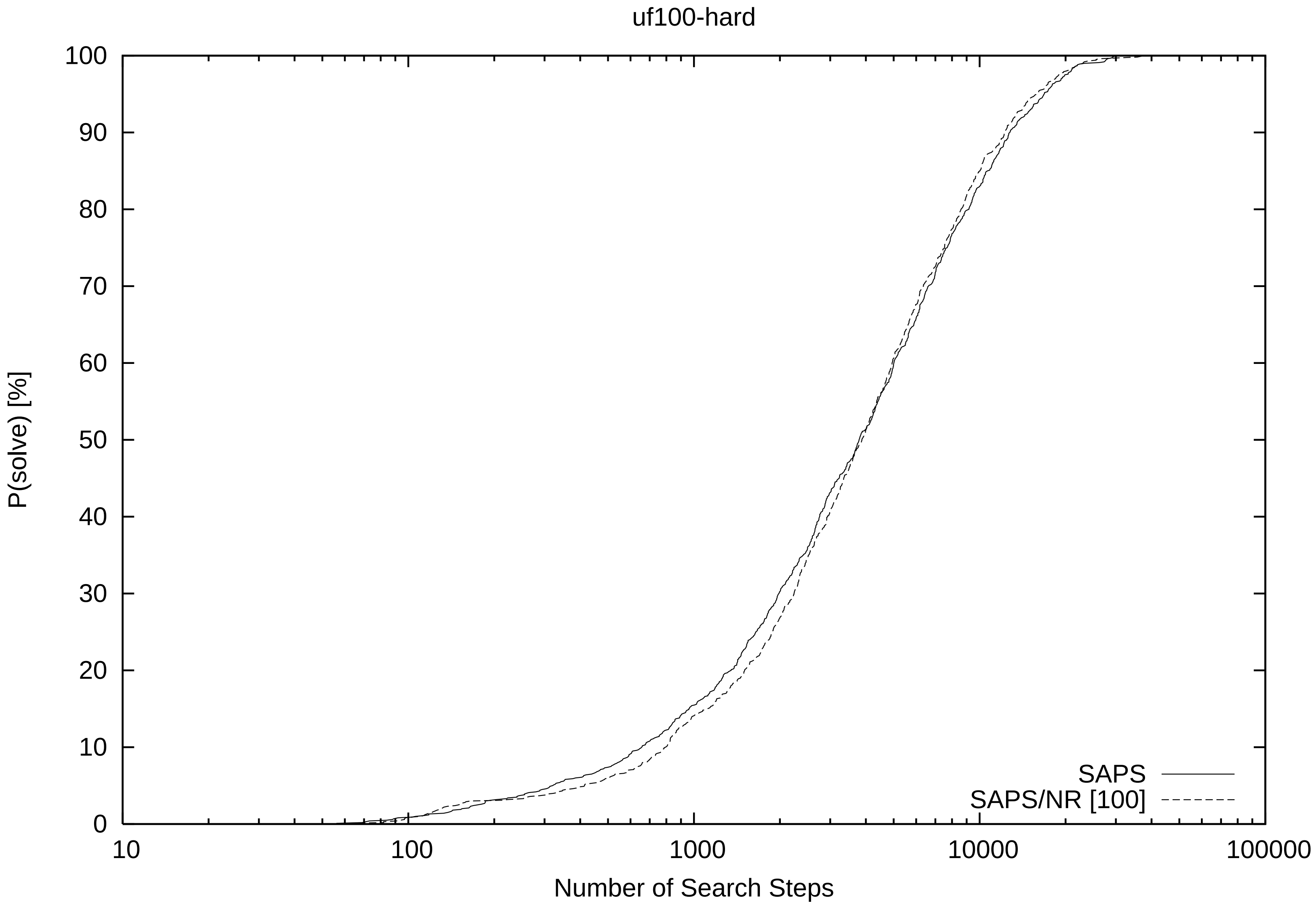
$\leadsto$ after initialisation, SAPS/NR is completely deterministic

Experiment:

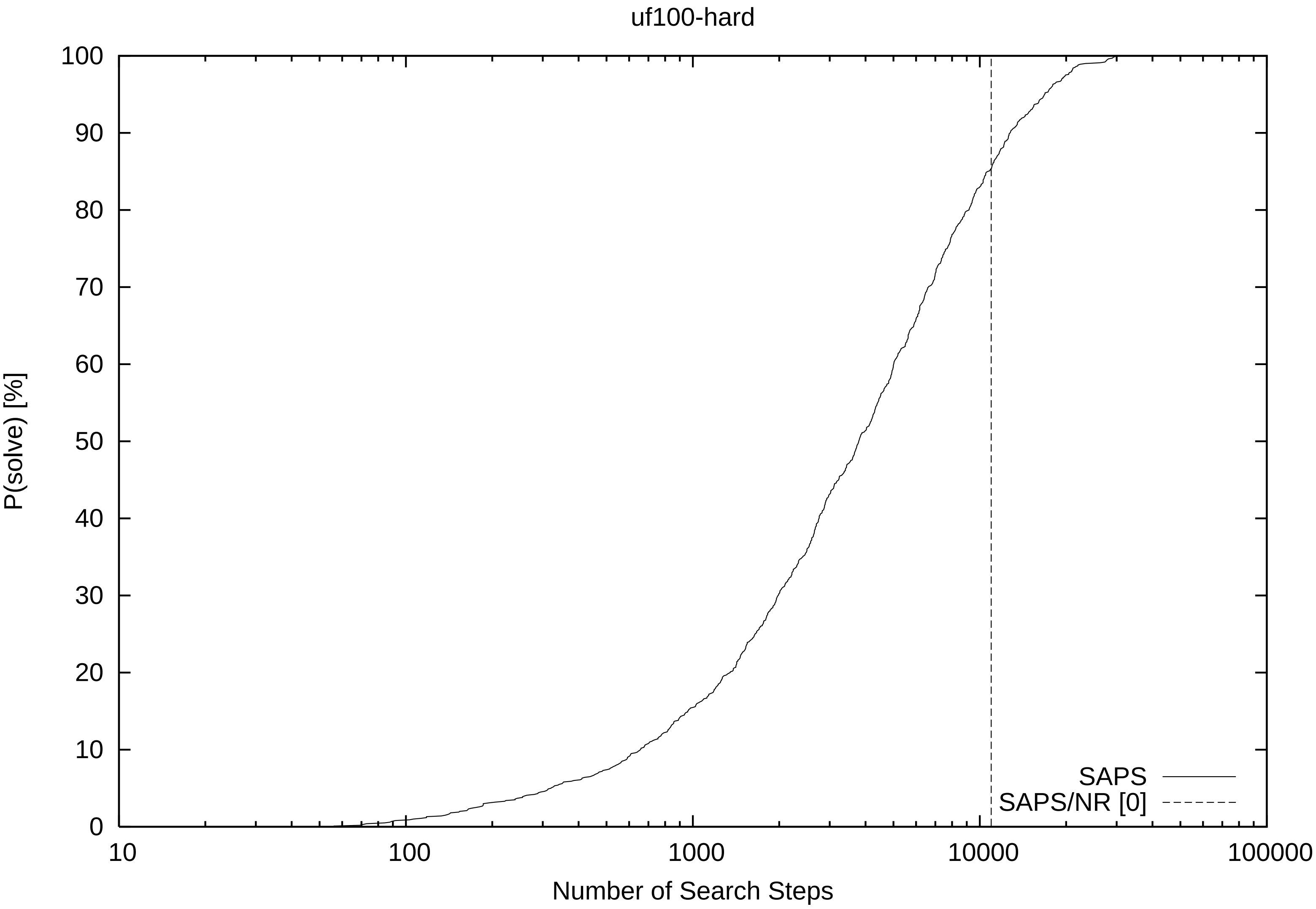
1. Compare performance and behaviour of SAPS and SAPS/NR.
2. Study variants of SAPS/NR in which only a fraction of variables is initialised with random truth values (others set deterministically).

Instance	SAPS		SAPS/NR	
	Mean	*c.v.*	Mean	*c.v.*
uf100-easy	102	0.75	103	0.70
uf100-hard	5,572	0.95	5,458	0.97
uf250-hard	296,523	0.98	282,668	1.02
uf400-hard	4,349,480	0.75	3,662,192	0.83
flat100-hard	35,124	1.02	33,519	0.98
bw_large.a	3,374	0.85	3,245	0.81
bw_large.b	50,025	0.95	50,266	0.94
logistics.c	12,873	0.76	12,458	0.83
ssa7552-038	4,460	0.44	4,399	0.41
ais10	32,810	1.01	31,527	0.99

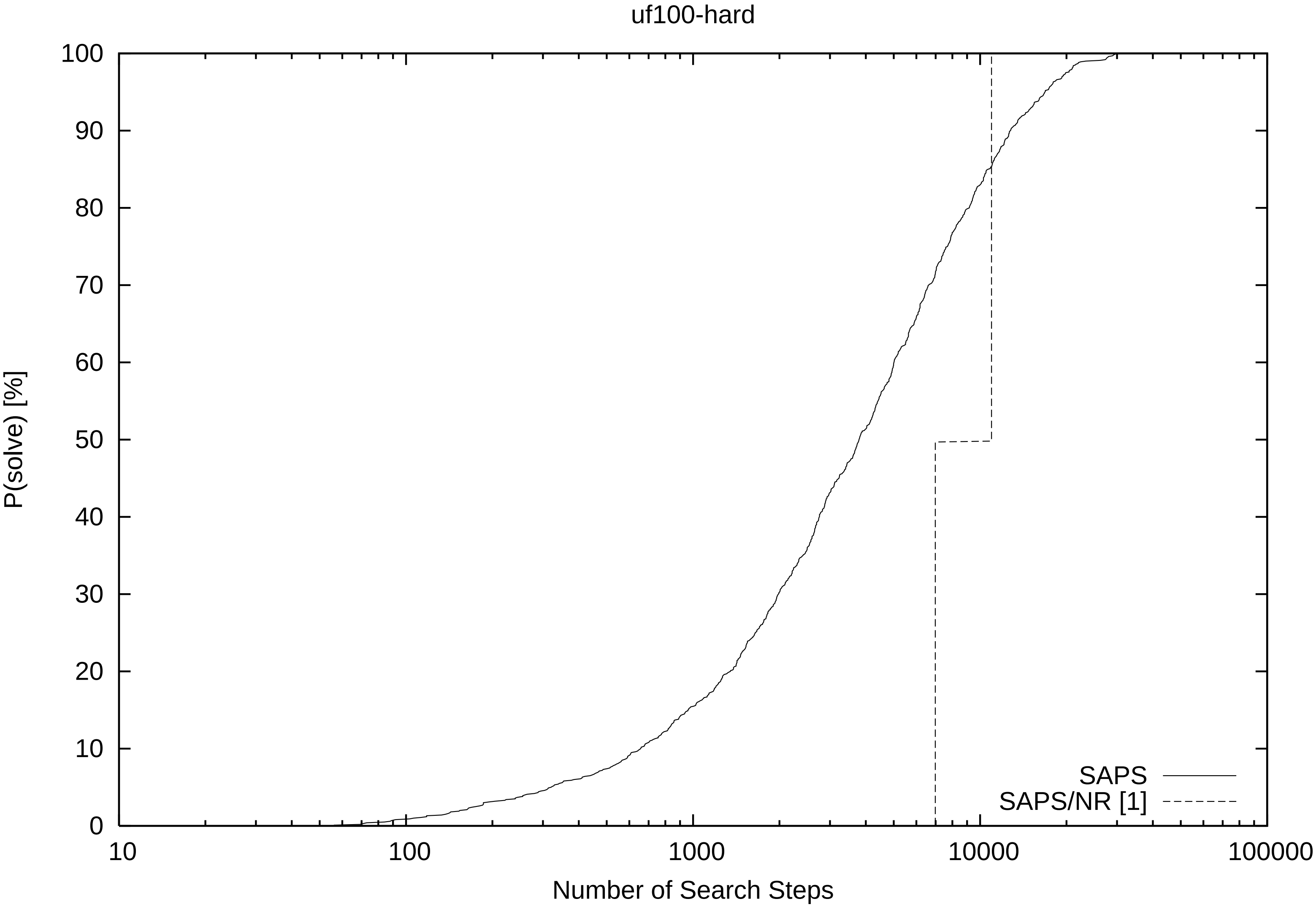
SAPS vs. SAPS/NR (100 random decisions)



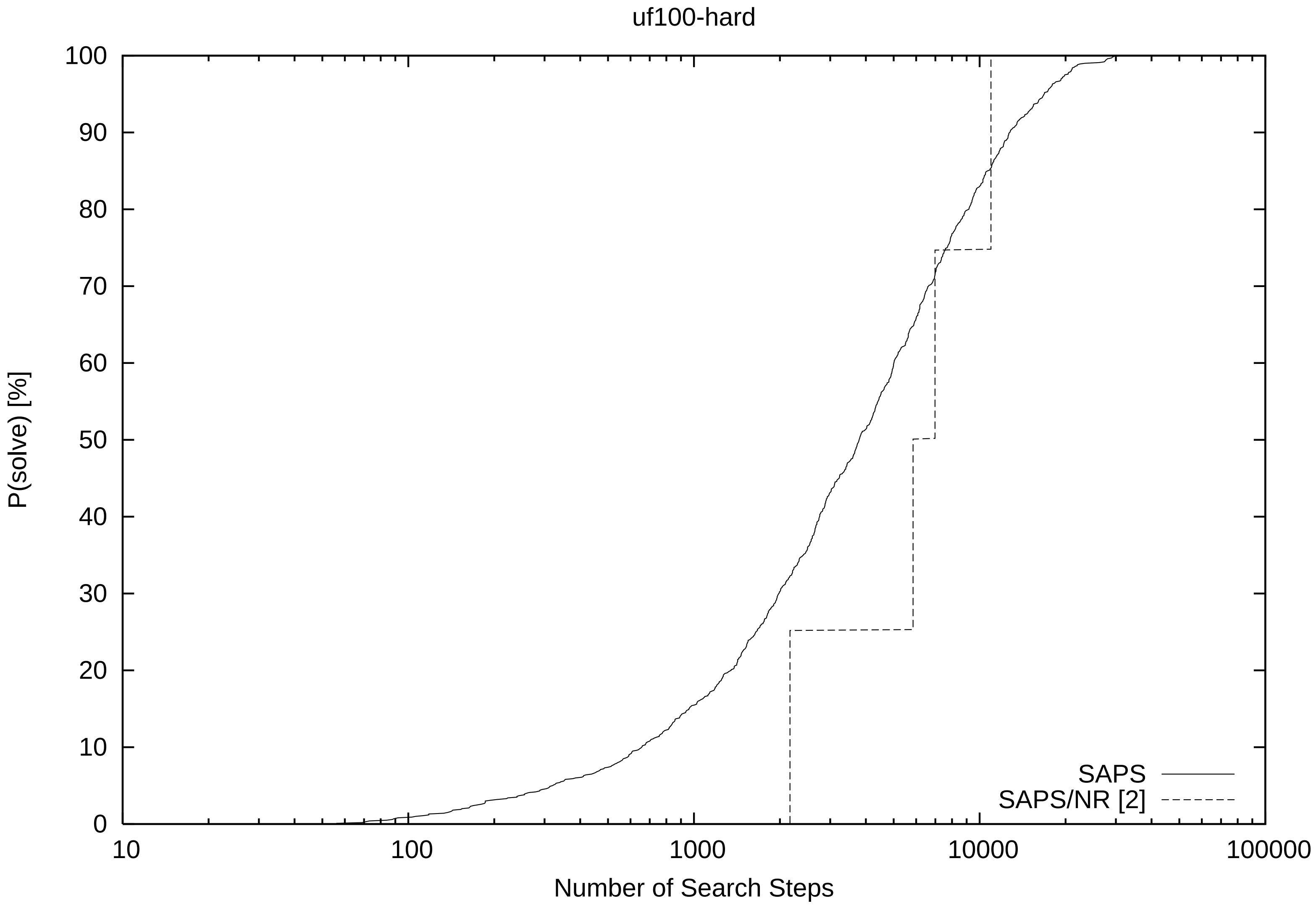
SAPS vs. SAPS/NR (0 random decisions)



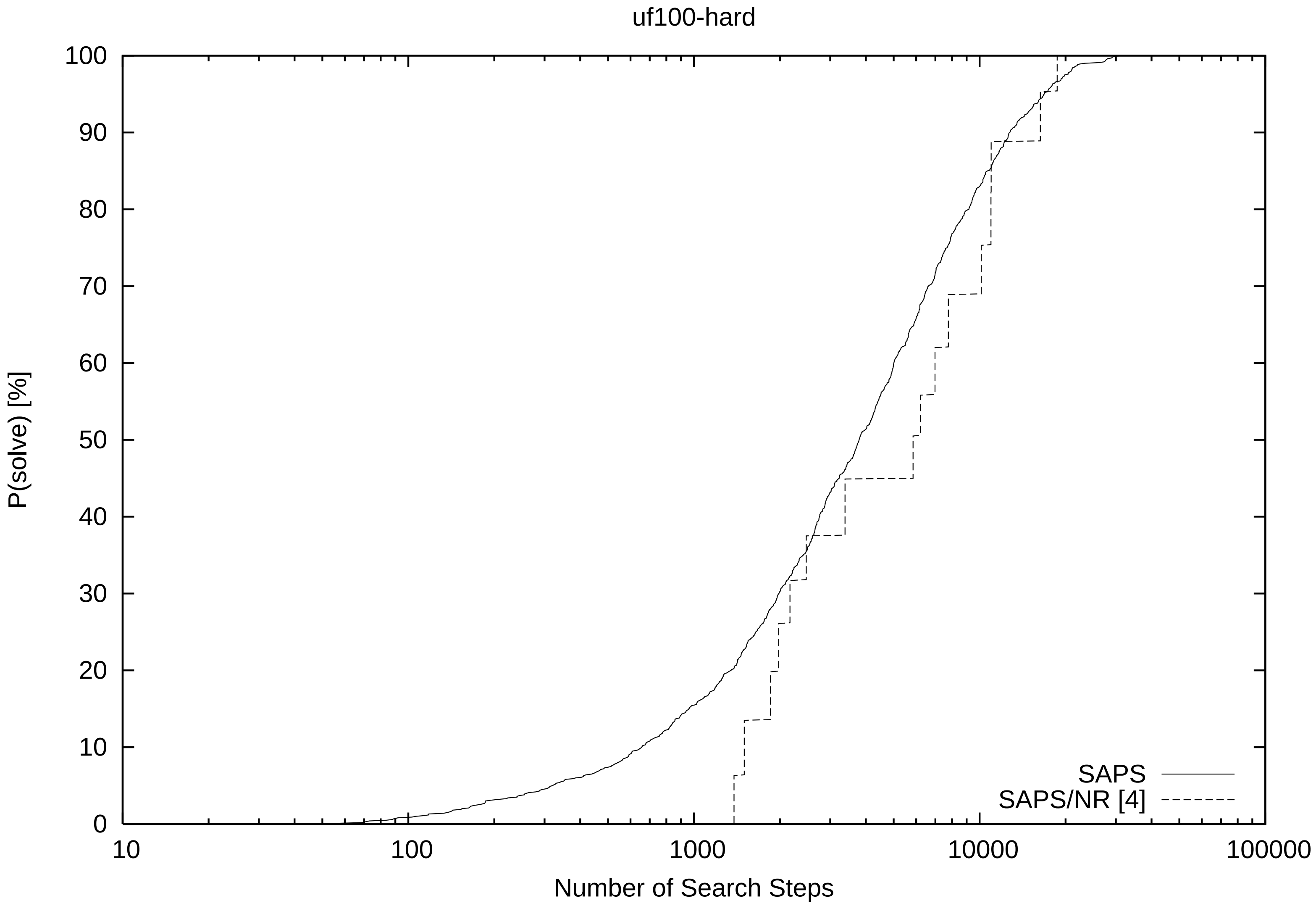
SAPS vs. SAPS/NR (1 random decision)



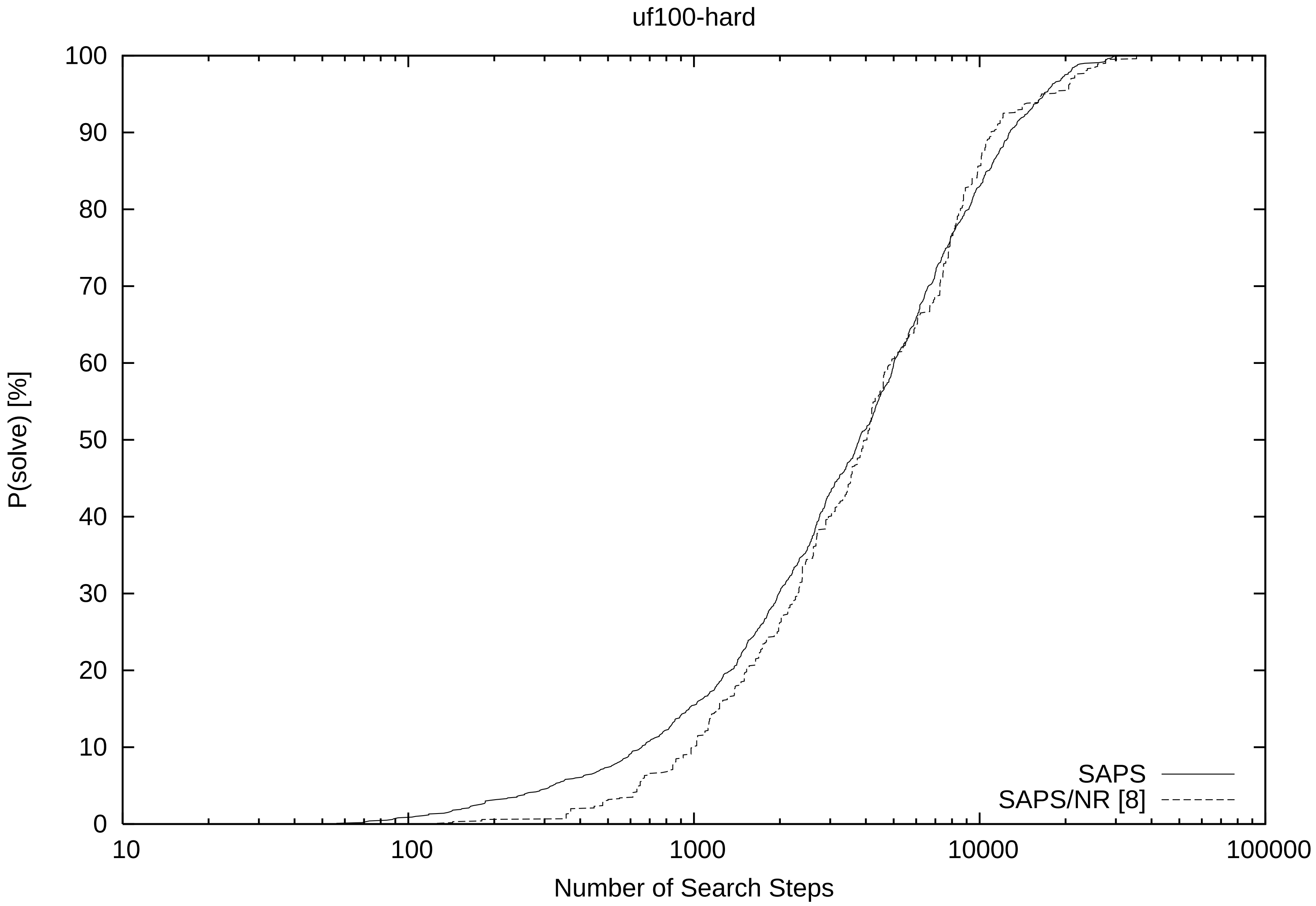
SAPS vs. SAPS/NR (2 random decisions)



SAPS vs. SAPS/NR (4 random decisions)



SAPS vs. SAPS/NR (8 random decisions)



Result:

- Behaviour and performance of SAPS/NR
  + random initialisation is indistinguishable
  from fully randomised SAPS
- Performance of (deterministic) SAPS/NR
  shows sensitive dependence on initial conditions
  $\rightsquigarrow$ central component in definition of chaotic behaviour!
- Diversifying effect of penalty updates
  is sufficient to propagate small amount
  of randomness throughout entire search process.

Conclusions

- Penalty mechanism in DLS $\not\Rightarrow$ global simplification (no “long-term memory”)
- Local (“short-term memory”) effects dominate search behaviour
- Penalty mechanism in SAPS primarily provides search diversification
- Only few initial random decisions are sufficient for obtaining same behaviour as fully randomised SAPS algorithm
- Behaviour of deterministic SAPS/NR algorithm sensitively depends on initial conditions (chaotic behaviour?)

Future Work

- characterisation of “warped” search spaces
- separation of short-term and long-term memory in DLS
- optimally weighted SAT instances
- advanced initialisation methods for SAPS/NR
- further investigation of “chaotic” behaviour in SAPS/NR