Karlsruher Institut für Technologie

SAT Solving with distributed local search

Guangping Li – *uzdif@student.kit.edu*

Institute of Theoretical informatics, Algorithmics II

Outline

Propositional Satisfiability Problem (*SAT*)

- Notations
- Local search in SAT problem

Solving SAT by *swpSolver*

- Basic scheme
- Our improvements

Our Parallel SAT-solver

- The pure portfolio approach
- Failures
- Initialization with a guide of formula partitioning

Conclusion

Outline

Propositional Satisfiability Problem (*SAT*)

- Notations
- Local search in SAT problem

Solving SAT by *swpSolver*

- Basic scheme
- Our improvements

Our Parallel SAT-solver

- The pure portfolio approach
- Failures
- Initialization with a guide of formula partitioning

Conclusion

KIT Department of Informatics
Institute of Theoretical informatics

Propositional Satisfiability Problem

Notations

- propositional variable: variable with two possible logical values *true* or *false*
- literal: an atomic formula either be a positive literal v or a negative literal $\bar{v}$.
- clause: disjunction of literals.
- CNF-formula: conjunction of clauses
- assignment: $V \rightarrow \{true, false\}$
- SAT problem: to determine whether a given formula is satisfiable or not

Here is an example of SAT problem:

$$F = (v_1 \vee \bar{v_3}) \wedge (v_2 \vee v_1 \vee \bar{v_1})$$
$$Vars(F) = \{v_1, v_2, v_3\}$$
$$numV(F) = |Vars(F)| = 3$$
$$Lits(F) = \{v_1, \bar{v_1}, v_2, v_3, \bar{v_3}\}$$
$$Cls(F) = \{C_1, C_2\}$$
$$numC(F) = |Cls| = 2$$
$$C_1 = \{v_1, \bar{v_3}\}$$
$$C_2 = \{v_2, v_3, \bar{v_1}\}$$

$A(v_1)$ = *true*, $A(v_2)$ = *false*, $A(v_3)$ = *true*,
A is an assignment satisfying F.

$\hat{A}(v_1)$ = *true*, $\hat{A}(v_2)$ = *false*, $\hat{A}(v_3)$ = *false*,
$\hat{A}$ is an assignment with conflict in C_2.

Local search in SAT problem

Local Search

- an instance I of a hard combinational Problem P
- a set of solutions $S(I)$
- an object function (score or cost) Γ
- to find the solution with minimum cost by applying local changes.

Algorithmus 1 : Focused Local Search

input : A CNF Formula F
parameter : *Timeout*
output : a satisfying assignment A

$A \leftarrow$ random generated assignment A
while $(\exists$ *unsatisfied clause* $\wedge$ *Timeout does not occur*$)$ **do**
    $c \leftarrow$ random selected unsatisfied clause
    $x \leftarrow pickVar(A, c)$
    $A \leftarrow flip(A, x)$

Local search in SAT problem

Stochastic Local Search (SLS)

- use the probability distribution of the scores of candidate solutions
- the more advantageous a move is, the higher is the probability of choosing that move

---

Algorithmus 2 : PickVar in *probSAT*

---

input : current assignment A, unsatisfied clause c
output : a variable x in c to be flipped

for *v in c* **do**
  Evaluate v with function $\Gamma(A, v)$;
$x \leftarrow$ randomly selected variable v in c with probability

$$p(v) = \frac{\Gamma(A,v)}{\sum_{u \in c} \Gamma(A,u)}$$

Local search in SAT problem

KIT
Karlsruher Institut für Technologie

Random walk in local search

- originally introduced in 1994
- By introducing “uphill noises”, the walkSAT combines greedy local search and random walk.

Algorithmus 3 : PickVar in walkSAT

input : current assignment A, unsatisfied clause c
parameter : probability p
output : a variable x in c to be flipped

for v *in* c **do**
    Evaluate v with function $\Gamma(A, v)$;
with probability p: $x \leftarrow v$ with maximum $\Gamma(A, v)$;
with probability $1 - p$: $x \leftarrow$ randomly selected v in c.

Outline

Propositional Satisfiability Problem (*SAT*)

- Notations
- Local search in SAT problem

Solving SAT by *swpSolver*

- Basic scheme
- Our improvements

Our Parallel SAT-solver

- The pure portfolio approach
- Failures
- Initialization with a guide of formula partitioning

Conclusion

KIT Department of Informatics
Institute of Theoretical informatics

Solving SAT by *swpSolver*

Basic scheme

Algorithmus 4 : Our Local Search

input : A CNF Formula F
parameter : *Timeout*
output : a satisfying assignment A

$A \leftarrow initAssign(F)$;
while $(\exists$ *unsatisfied clause* $\wedge$ *Timeout does not occur*$)$ **do**
    $c \leftarrow pickCla(A)$;
    $x \leftarrow pickVar(A, c)$;
    $A \leftarrow flip(A, x)$;

Evaluation

- 180 instances (*UNIF*) in random benchmark categories in SAT competition 2017
- All the clause have the same length k in a *UNIF* problem file.
- to construct one clause, k literals are randomly chosen from the $2n$ possible literals
- At least 60 (33%) problems form our 180 benchmark collections are unsatisfiable.
- Each experiment is repeated three times.
- PAR-2 runtime for a whole kSAT set

initAssign(F)

- *RandomInit*: build a complete assignment randomly
- *BiasInit*: assign *true* to a variable if the number of occurrences of its positive literal is larger than that of its negative literal.
- *Bias-RandomInit*: assign *true* to variable v_i with probability $\frac{posOccurences[i]}{posOccurences[i]+negOccurences[i]}$.

KIT Department of Informatics
Institute of Theoretical informatics

initAssign(F)

k	*RandomInit*	*BiasInit*	*Bias-RandomInit*
3	9221.9 **55**	9157.76 54	**9078.27** **55**
5	7143.9 82	**4351.09** **87**	4582.54 **87**
7	6238.51 60	**5421.9** 60	6310.7 60

- 3SAT: RandomInit
- 5SAT and 7SAT: BiasInit

pickVar(A,c)

- combine the random walk and stochastic selection
- pick greedy flips with zero breakcounts with a certain probability p.
- use the *SLS* with probability $(1 - p)$

Algorithmus 5 : Our pickVar

input : current assignment A, unsatisfied clause c
parameter : probability p
output : a variable x in c to be flipped

greedyVs $\leftarrow \emptyset$;
for *all v in c* **do**
    if (*break(A,v)= 0*) **then**
        greedyVs = *greedyVs* + $\{v\}$
with probability p: $x \leftarrow$ randomly selected variable $v \in$ *greedyVs* ;
with probability $(1 - p)$: $x \leftarrow$ randomly selected variable v in c with probability $\frac{\Gamma(A,v)}{\sum_{u \in c} \Gamma(A,u)}$

Variant 1: Walk

- statistic list S: how many times each variable is chosen for flipping
- The candidate with the small statistic value will be chosen.
- $p = \alpha \times \frac{s(randomV)}{s(greedyV)+s(randomV)}$
- Getting the random literals using stochastic process consumes the most runtime.

Variant 2: GreedyBreak

- *permitted greedy literal*: literal with zero breakcount and its statistic value is under a certain threshold t
- choose a permitted greedy literal randomly for flipping.
- if no permitted greedy literal exists, we pick a literal using *SLS* heuristic.
- 1st approach *Average*: $t = \alpha \times \frac{numF}{numV}$
- 2nd approach *Random-Flip*: $t = \alpha \times r$ with $r \in [0, numF]$.

PickVar(A,c) with simulated annealing

Simulated Annealing

- proposed by Kirkpatrick, Gelatt, and Vecchi.
- guide local search with a controlling parameter ***temperature***.
- The temperature varies according to the score of the current situation.
- Higher temperature allows uphill moves with higher probability.

- Walk: $p = \alpha \times \frac{s(randomV)}{s(greedyV)+s(randomV)}$
- Average: $t = \alpha \times \frac{numF}{numV}$
- Random-Flip: $t = \alpha \times r$ with $r \in [0, numF]$.
- $\alpha = \tau \times (c_b)^{-q(A)}$
- two variants of $q(A)$:
  - $q_{global}(A) = unsatN(A)$
  - $q_{local}(A) = |\{v | v \in c \wedge break(v) = 0\}|$

KIT Department of Informatics
Institute of Theoretical informatics

Evaluation

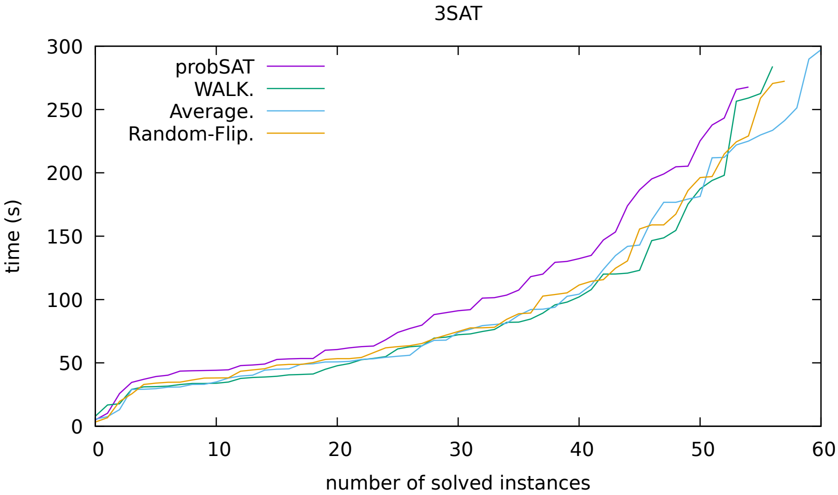
pickVar

- probSAT:
  - implemented by the authors of the original paper
  - non-incremental approach to the 3SAT problems and incremental method for 5SAT and 7SAT
- yalSAT:
  - the third version of yalSAT submitted to the 2017 SAT competition
  - uses a variant of *probSAT* randomly in the restart of a searchround

k	*probSAT*	*yalSAT*	*Walk*	*Average*	*Random-Flip*
3	9221.955	17062.3541	7430.1257	**6161.1161**	7308.0158
5	7143.982	5676.6385	3330.61**89**	**2939.7489**	4003.0688
7	6238.5160	10063.454	5409.6761	**3829.9565**	4903.6160

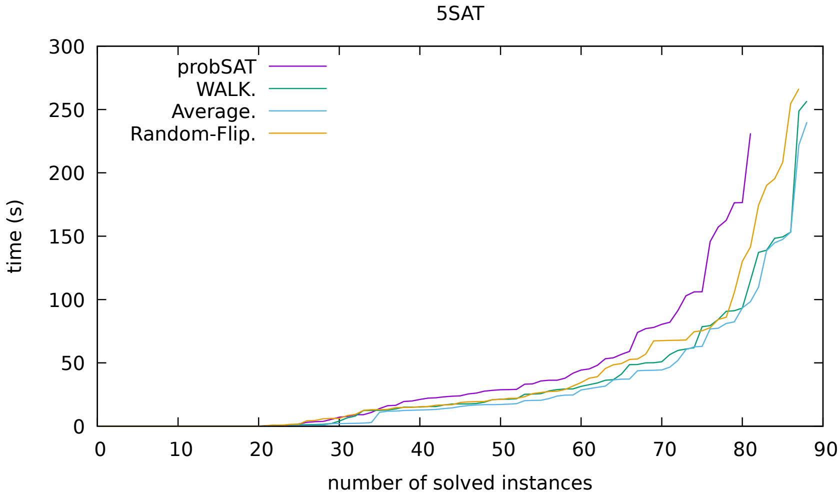
Evaluation

pickVar



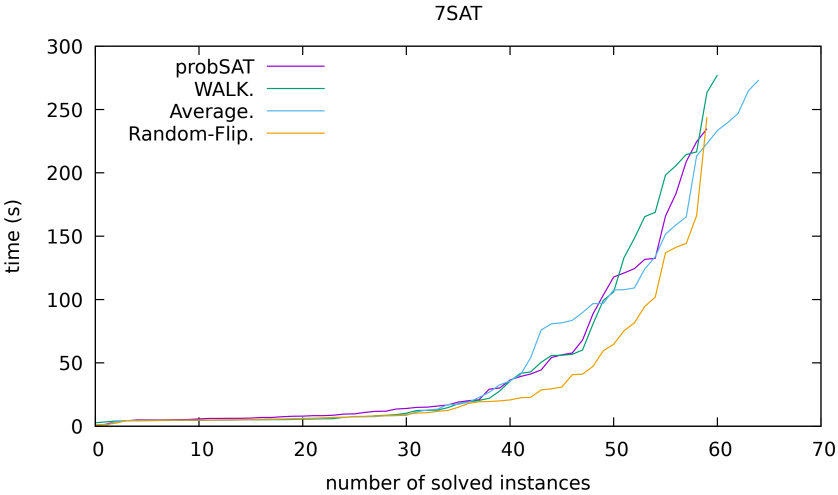
Evaluation

pickVar



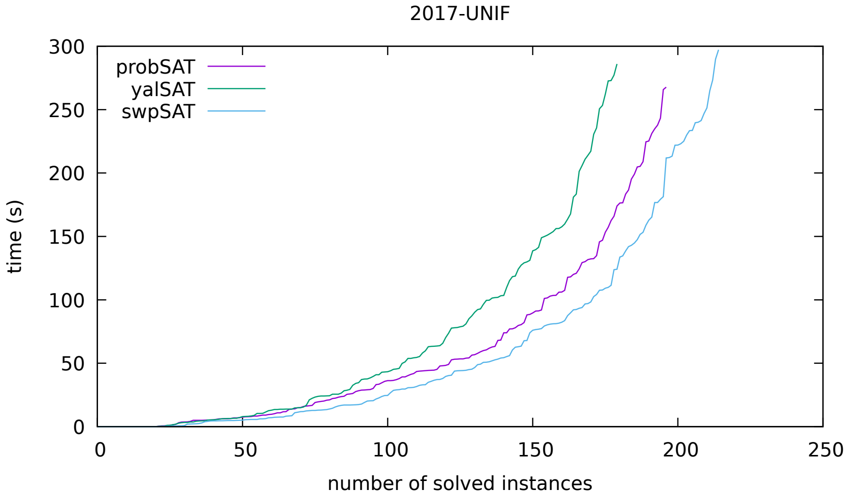
Evaluation

pickVar



Evaluation

swpSolver



Outline

Propositional Satisfiability Problem (*SAT*)

- Notations
- Local search in SAT problem

Solving SAT by *swpSolver*

- Basic scheme
- Our improvements

Our Parallel SAT-solver

- The pure portfolio approach
- Failures
- Initialization with a guide of formula partitioning

Conclusion

The pure portfolio approach

- random generator affects the performance.
- the agents run the *swpSAT* with different random generation policies.

rand()	*minstd_rand*	*mt*19937
*mt*19937_64	*ranlux*24_*base*	*ranlux*48_*base*
*ranlux*24	*ranlux*48	*knuth_b*
default_random_engine	*minstd_rand*0	-

k	*swpSAT*	*pure portfolio*	Speedup	Efficiency
3	12971.3 59	7426.9 69	1.75	0.16
5	8339.98 89	5185.75 94	1.61	0.14
7	13406.26 66	2853.81 83	4.70	0.43

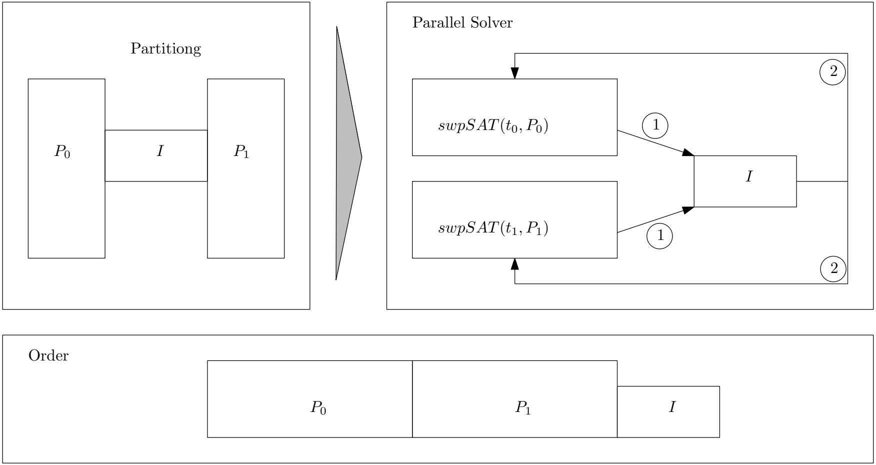
Evaluation

benchmark *COMBINE*

- formula partitioning by a relatively balanced partitioning with small intersection
- combine two *UNIF* benchmark instances
- build the intersection based on a randomly chosen satisfying assignment
  - *BIG*: *COMBINE* problems with big intersection ($\frac{numCI}{numC} > 1\%$)
  - *SMALL*: *COMBINE* problems with big intersection ($\frac{numCI}{numC} < 1\%$)

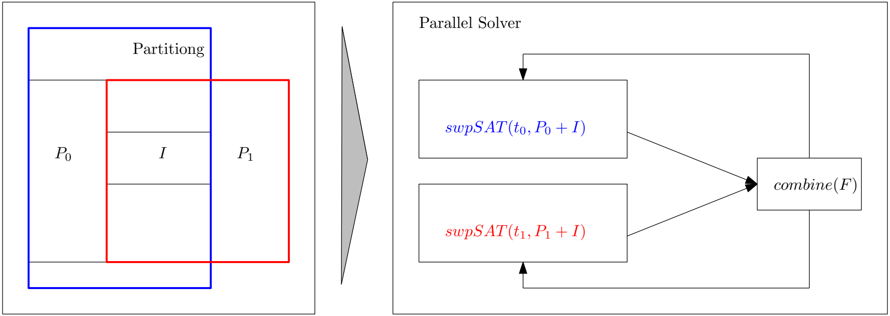
Failures

2nd Approach: Solver with formula partitioning



Failures

3nd Approach: Solver with combination of sub-assignments



Our Parallel GCP-solver

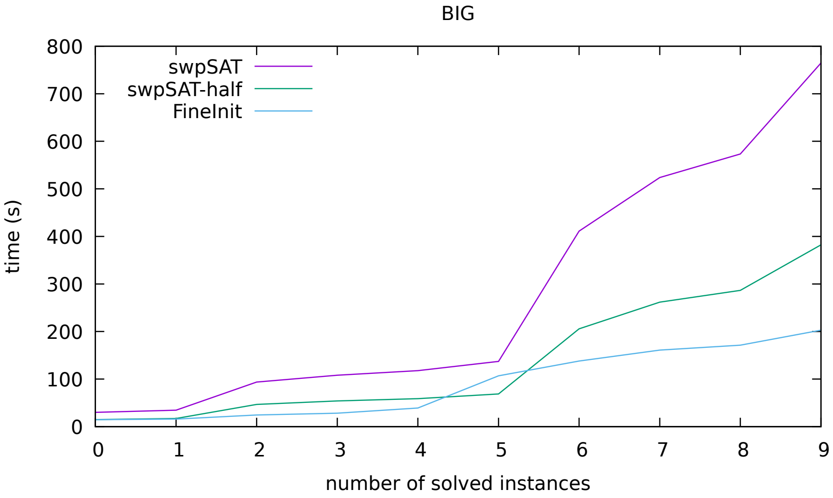
4th Approach: Initialization with formula partitioning (*FineInit*)

Karlsruher Institut für Technologie

- The formula partitioning information is only used to get an initial solution.
- The statistic information shared among the agents encourages the further search to flip non-critical variables in clauses.
- The candidate with the smallest statistic value will be chosen in the next step.
- The agents use one common statistic matrix.

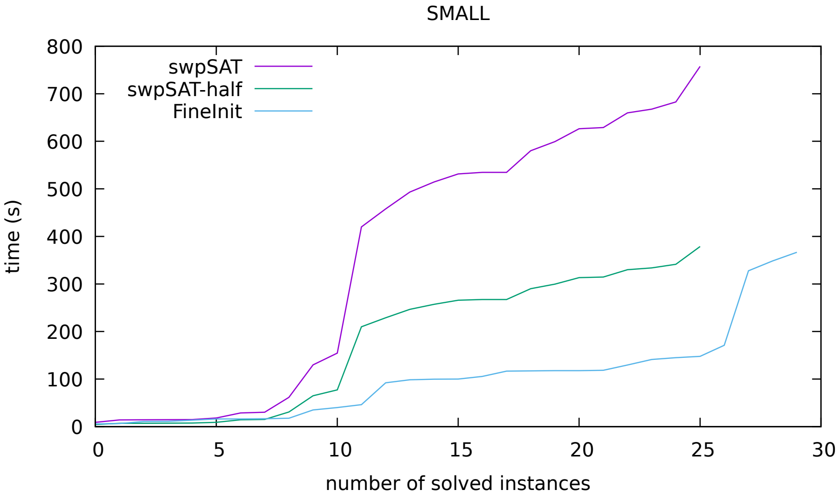
Evaluation

benchmark *COMBINE-BIG*



Evaluation

benchmark *COMBINE-SMALL*



Our parallel solver

- uses *FineInit* as initialization
- tries different search paths with pure portfolio approach

input : A CNF Formula F, number of Processors n_p

$A \leftarrow initAssign(F)$;
foreach $(Processor_t \ for \ t \in \{1, .., n_p\})$ **do**
    $A_t \leftarrow A$;
    $i \leftarrow t\%2$;
    swpSAT(P_i);
    swpSAT(P_{1-i}) ;
    while $(!sat \wedge !Timeout)$ **do**
        $A_t \leftarrow A$;
        swpSAT(F);
        $sat \leftarrow true$;

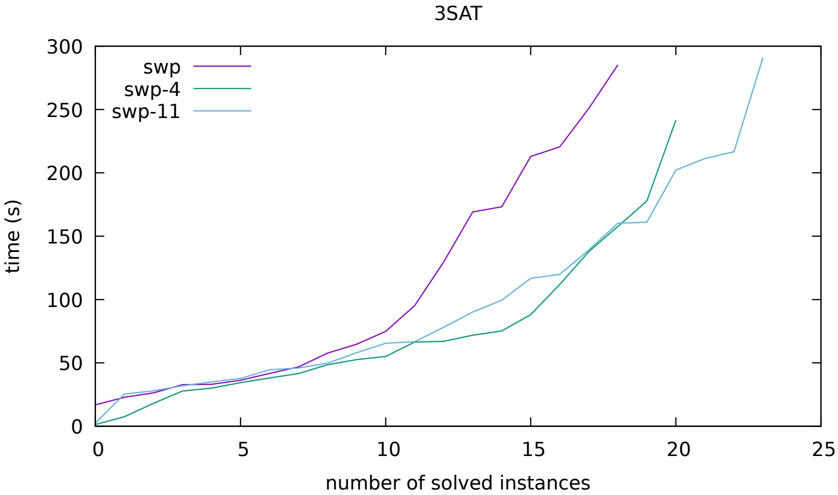
Evaluation

benchmark UNIF

- separate the vertices according their indices in two partition sets
- All vertices v_i with $i < \frac{numV}{2}$ belong to P_0.
- The rest vertices belong to P_1.

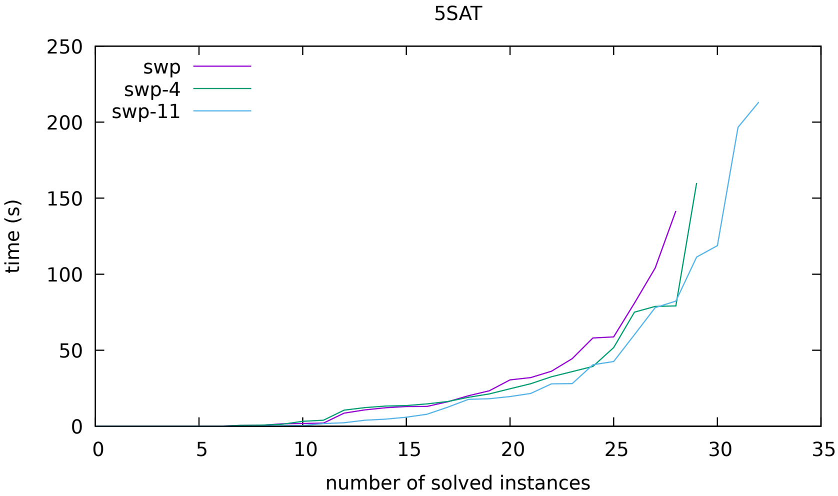
k	*swp-4*	S	E	*swp-11*	S	E
3	3350.36 21	1.49	0.37	2376.26 24	2.10	0.19
5	2535.86 30	1.23	0.31	1115.99 33	2.79	0.25
7	3874.51 22	1.28	0.32	1076.26 27	4.62	0.42

3SAT



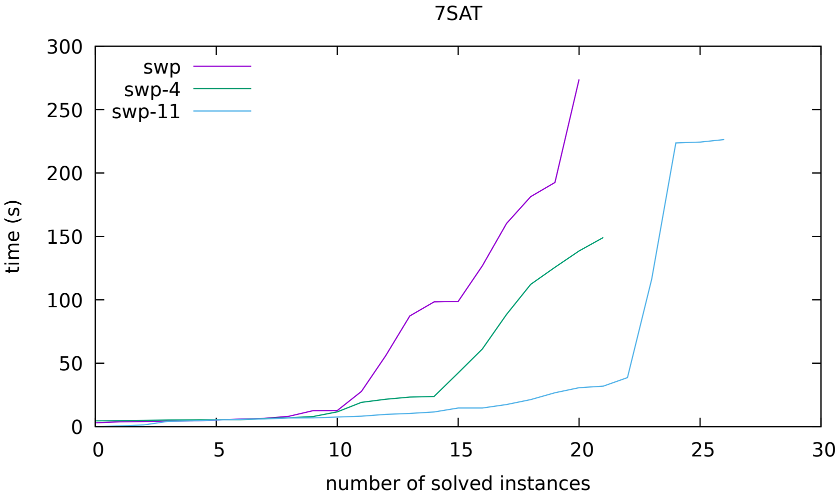
3SAT
swp
swp-4
swp-11
time (s)
number of solved instances
0
50
100
150
200
250
300
0
5
10
15
20
25

5SAT



5SAT
swp
swp-4
swp-11
time (s)
250
200
150
100
50
0
0
5
10
15
20
25
30
35
number of solved instances

7SAT


7SAT
swp
swp-4
swp-11
time (s)
0
50
100
150
200
250
300
0
5
10
15
20
25
30
number of solved instances

Conclusion

Karlsruher Institut für Technologie

Our improvement:

- a stochastic local search algorithm *swpSAT* with the incorporation of *walkSAT* and *probSAT*
- different local variants, which get better performance than the *probSAT* algorithm
- parallel *swpSAT* solver with formula partitioning
- the formula partitioning information can guide the local search

Further work

- use different search strategies
- use different cooperation strategies
- use different random generation in local search

THANK YOU
FOR
YOUR
ATTENTION
ANY QUESTIONS?