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
- Two 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
- Two 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 ← random generated assignment A
while (∃ unsatisfied clause ∧ Timeout does not occur) do
    c ← random selected unsatisfied clause
    x ← pickVar(A, c)
    A ← flip(A, x)
```


KIT Department of Informatics
Institute of Theoretical informatics

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 Γ(A, v);
x ← randomly selected variable v in c with probability
```

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

KIT Department of Informatics
Institute of Theoretical informatics

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
- Two failures
- Initialization with a guide of formula partitioning

Conclusion

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 benchmark instances used in our experiments are the 180 instances (*UNIF*) in random benchmark categories in SAT competition 2017.
- all the clause have the same length 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*: buid 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.
- using the *probSAT* 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

- a statistic list S to record 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 greefy literal exists, we pick a literal using *probSAT* heuristic.
- 1.approach *Average*: $t = \alpha \times \frac{numF}{numV}$
- 2.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 $q(A)$:
  - $q_{global}(A) = unsatN(A)$
  - $q_{local}(A) = |\{v | v \in c \wedge break(v) = 0\}|$

Guangping Li:
SAT Solving with distributed local search
KIT Department of Informatics
Institute of Theoretical informatics

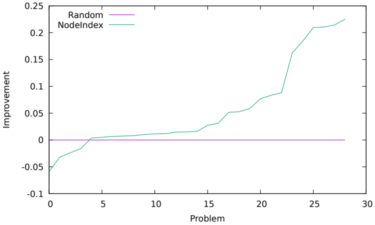
Evaluation

pickVar

- The 68 graphs used in our experiments are from the DIMACS benchmark collection.
- The single-threaded experiments were run on computers that had four AMD(R) Opteron(R) processors 6168 (1.9 Ghz with 12 cores) and 256GB RAM. The computers ran the 64-bit version of Ubuntu 12.04.
- The multi-threaded experiments were run on fat nodes InstitutsClusterII. IC2 is a distributed memory parallel computer with 480 16-way so-called thin compute nodes and 5 32-way so-called fat compute nodes. The thin nodes are equipped with 16 cores, 64 GB main memory, whereas the fat nodes are equipped with 32 cores, 512 GB main memory.

Evaluation

- An advantage plot shows the advantage of an algorithms to another algorithm. The y-axis gives the ordered percentage differences.



KIT Department of Informatics
Institute of Theoretical informatics

Our improvements

Basic scheme

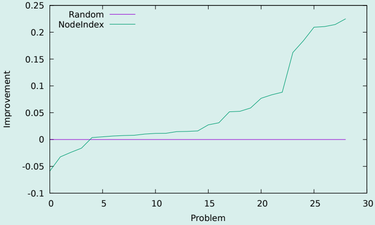
- Step 1: Initialize Coloring (How to initialize coloring?)
- Step 2: Solve k-VCP (How to improve Tabucol?)
- Step 3: Reduce a color (How to reconstruct coloring?)

Solving VCP by Tabucol

Step 1:How to initialize coloring?

Our Node-index initialization vs Random initialization

- **Node-index initialization** is to use c: $v_i \to i$ as the initial solution.
- **Random initialization** is to build a coloring randomly.

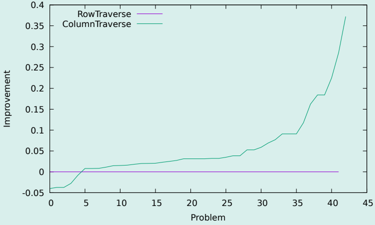


Solving VCP by Tabucol

Step 2: How to improve Tabucol?

Our column-traverse vs row-traverse of solution matrix

- To find next move, the maximum element in the solution matrix must be found.
- If more than one candidate exists, the first found one is chosen as the next step.
- This matrix can be traversed row by row or column by column.

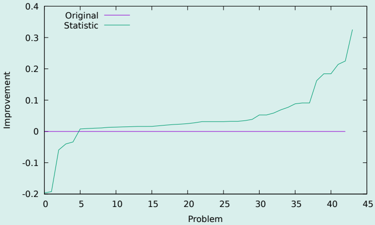


Solving VCP by Tabucol

Step 2: How to improve Tabucol?

Statistic matrix

- The Tabucol algorithm uses a tabu list to avoid short-term cycling.
- To recognize long-term cycling, a *statistic matrix* S is added.
- The S_{ij} represents how many times a one-step move $[i, j]$ was chosen as the next step.
- The candidate with the smallest statistic value will be chosen in the next step.



Solving VCP by Tabucol

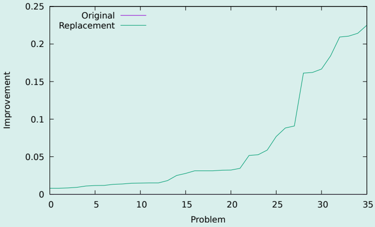
Step 3: How to reconstruct new coloring?

An observation

It seems that the solution loses its potential in the process of reducing colors iteratively. So it should be helpful to use a new and perhaps more potential coloring.

Replace by a randomly generated solution

We replace the current illegal solution occasionally by a new randomly generated coloring of the same size.



Outline

The vertex coloring problem

- k-VCP
- VCP

The Tabucol algorithm

Solving VCP by Tabucol

- Basic scheme
- Our improvements

Our Parallel GCP-solver

- Parameter combinations
- Our approaches

Conclusion

Our Parallel GCP-solver

Parameter combinations

- Most graphs get better results with the our suggestions.
- Some graphs get better results with the original GCP-solver.
- The agents run with different combinations of the suggestions (L, α, the search directions and whether a statistic matrix is introduced).

Our Parallel GCP-solver

Parameter combinations

Index	L	α	Initialization	Replace	Traverse	Statistic
1	9	0.38	Node-Index	true	Column	true
2	1	0.77	Node-Index	true	Column	true
3	11	0.90	Node-Index	true	Column	true
4	17	0.59	Random	true	Column	true
5	18	0.42	Node-Index	false	Column	false
6	4	0.92	Node-Index	true	Column	true
7	16	0.76	Node-Index	false	Row	false
8	17	0.47	Node-Index	false	Column	false
9	2	0.60	Node-Index	true	Column	false
10	2	0.54	Node-Index	false	Column	true
11	5	0.46	Random	true	Column	true
12	11	0.63	Random	true	Column	true
13	7	0.83	Node-Index	true	Column	true
14	8	0.98	Node-Index	false	Row	true
15	18	0.58	Node-Index	true	Column	false
16	13	0.90	Node-Index	false	Column	true

Our Parallel GCP-solver

Parameter combinations

Index	L	α	Initialization	Replace	Traverse	Statistic
17	20	0.56	Node-Index	true	Column	false
18	10	0.95	Node-Index	true	Column	true
19	15	0.55	Node-Index	true	Row	true
20	17	0.39	Node-Index	true	Column	true
21	18	0.52	Node-Index	false	Column	true
22	11	0.32	Node-Index	true	Column	true
23	15	0.62	Node-Index	false	Column	true
24	6	0.94	Random	true	Column	true
25	9	0.94	Node-Index	false	Column	false
26	12	0.96	Node-Index	true	Column	true
27	16	0.58	Node-Index	false	Column	true
28	9	0.45	Node-Index	false	Column	true
29	19	0.95	Node-Index	true	Column	true
30	18	0.31	Node-Index	true	Column	false
31	6	0.50	Node-Index	false	Column	false
32	15	0.93	Node-Index	false	Column	false

Our Parallel GCP-solver

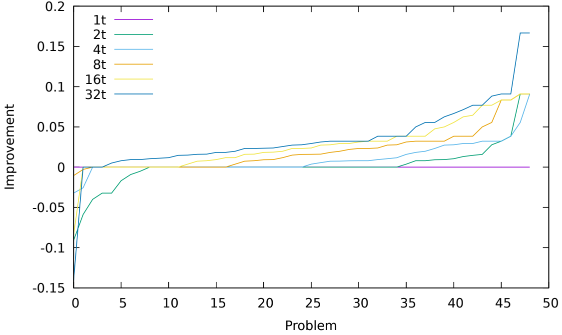
1st Approach: The pure portfolio approach

- The agents run the GCP solver with different parameter combinations.
- After collecting the solutions found by each agent, the search takes the coloring of the minimum size as the result.

Our Parallel GCP-solver

2nd Approach: Forced color reducing

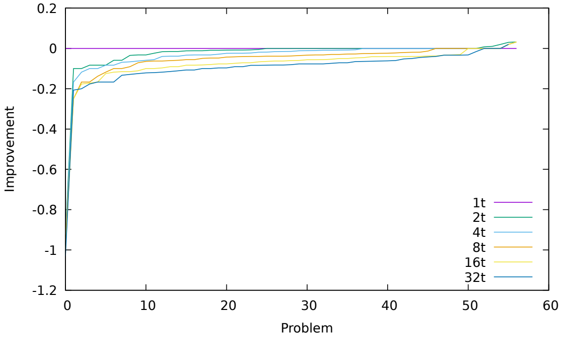
- This approach is based on the pure portfolio approach.
- The agents share the minimum size.
- One agent has already found a *k*-coloring and broadcasts it.
- With this notification, all agents search for a legal $k-1$ coloring.



Our Parallel GCP-solver

3rd Approach: Tabu sharing

- A tabu list records the search path to avoid short-term cycling.
- The agents share the “traps” of local search loops.



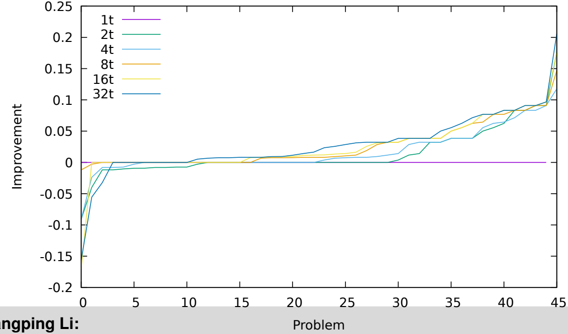
Guangping Li:
SAT Solving with distributed local search

KIT Department of Informatics
Institute of Theoretical informatics

Our Parallel GCP-solver

4th Approach: Statistic sharing

- To recognize long-term cycling, a *statistic matrix* S is added.
- The S_{ij} represents how many times a one-step move $[v_i, j]$ was chosen as the next step.
- The candidate with the smallest statistic value will be chosen in the next step.
- The agents use one common statistic matrix.



Outline

The vertex coloring problem

- k-VCP
- VCP

The Tabucol algorithm

Solving VCP by Tabucol

- Basic scheme
- Our improvements

Our Parallel GCP-solver

- Parameter combinations
- Our approaches

Conclusion

KIT Department of Informatics
Institute of Theoretical informatics

Conclusion

Our improvement:

- An algorithm solves the VCP with parallel Tabucol searches.
- The statistic matrix recognizes long-term cycling and brings improvement to our algorithm.
- Certain information exchange (minimum size, statistic matrix) can improve the performance of the parallel search.

Comparison of our GCP-solver with DSATUR, PASS, TRICK:

- 50 of 68 (73%) benchmark graphs get best results with our solver.
- 20 of 68 (29%) benchmark graphs get unique best results with our solver.

Further work

- Using different search strategies
- Using different cooperation strategies
- Using different algorithms in agents

THANK YOU
FOR
YOUR
ATTENTION
ANY QUESTIONS?