SoCS 2021 14th Annual Symposium on Combinatorial Search

Configuration Checking Based Local Search

Shaowei Cai

Institute of Software, Chinese Academy of Sciences

Outline

01. Local Search and Cycling Issue

02. Configuration Checking for Assignment Problems

03. Configuration Checking for Subset Problems

04. Conclusions and Challenges

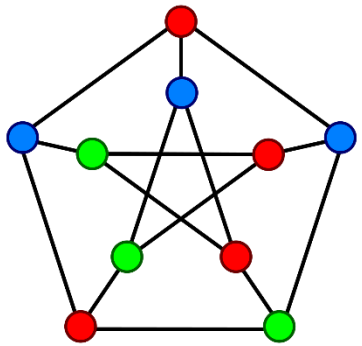
1

PART ONE

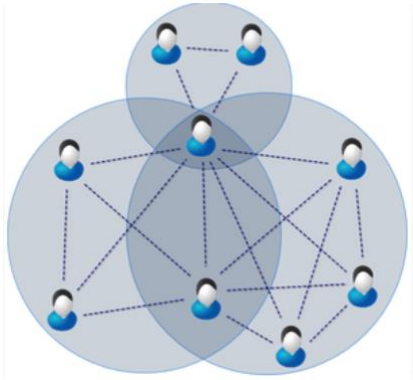
Local Search and Cycling Issue

Combinatorial Problems

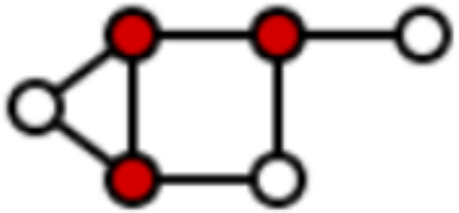
$\varphi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$ **SAT, MaxSAT**



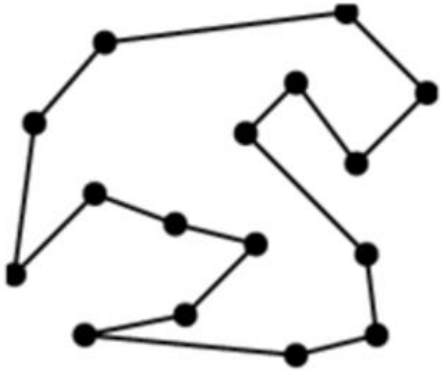
Graph Coloring



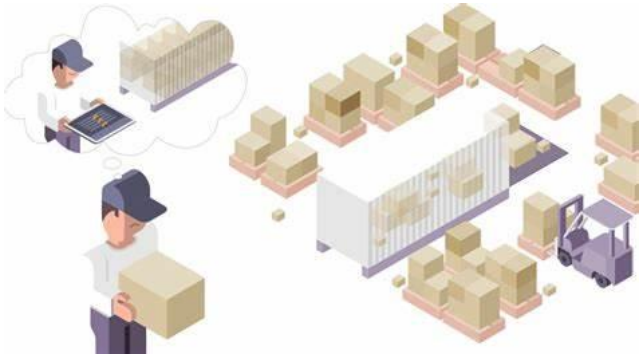
Max Clique



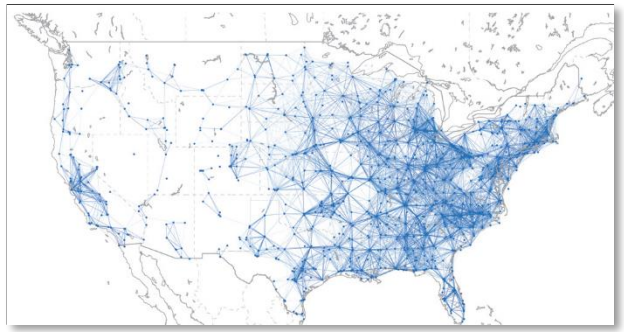
Vertex Cover



Travel Salesman Problem



Bin Packing

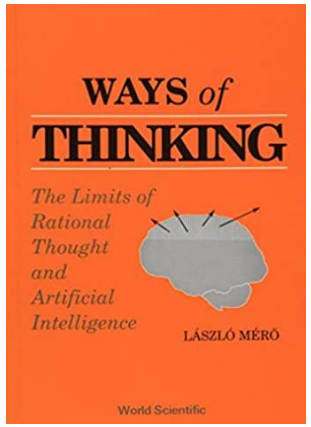


Spectrum Allocation

Local Search: A Heuristic Search Method

Many important combinatorial problems are NP-hard.

Resort to **heuristic** methods to tackle hard combinatorial problems practically.



"The name heuristics applies to every rule, conclusion, evaluation, and principle that works in certain situations most of the time, but not always."

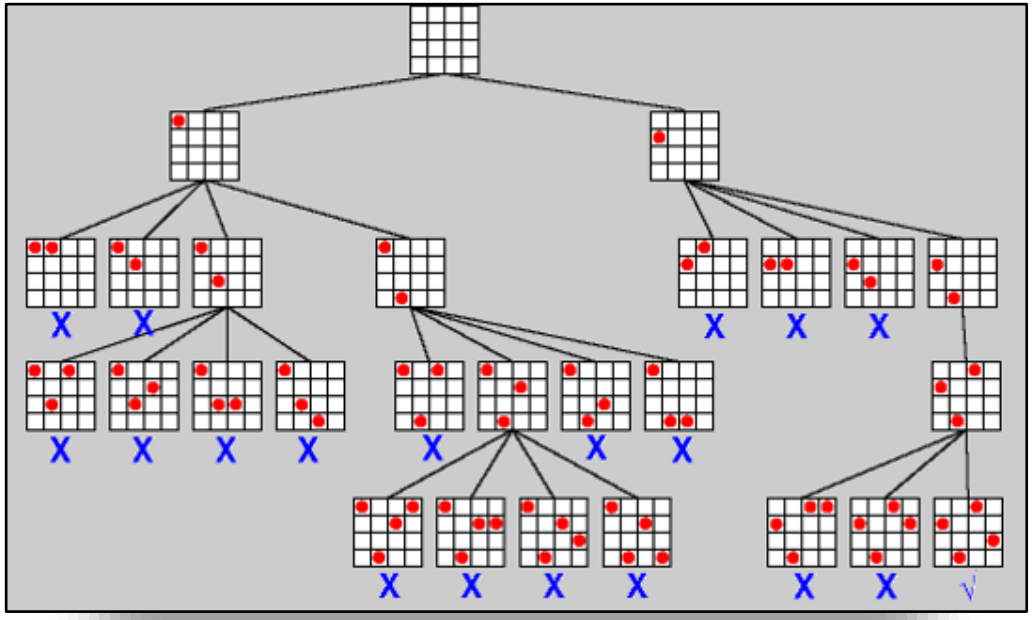
"The algorithms …called heuristics: for most of them we know that they do not work on worst-case instances, but there is good evidence that they work very well on many instances of practical interest."

---Sanjeev Arora (theoretical computer scientist)

Local Search: A Heuristic Search Method

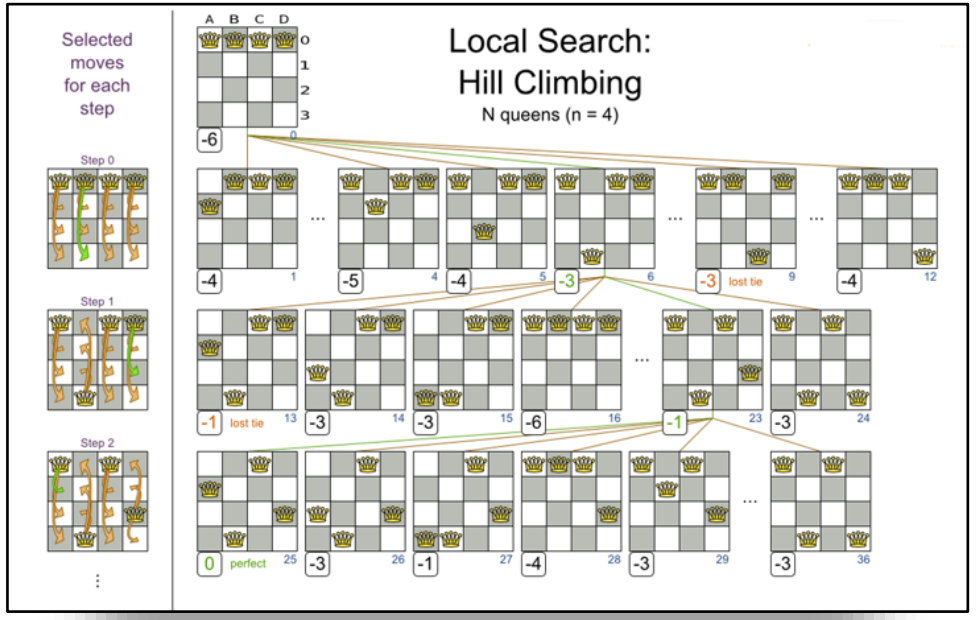
Local Search is a stochastic heuristic search method,
and is one of the most effective methods for solving NP hard combinatorial problems.

Systematic Search



VS

Stochastic Search

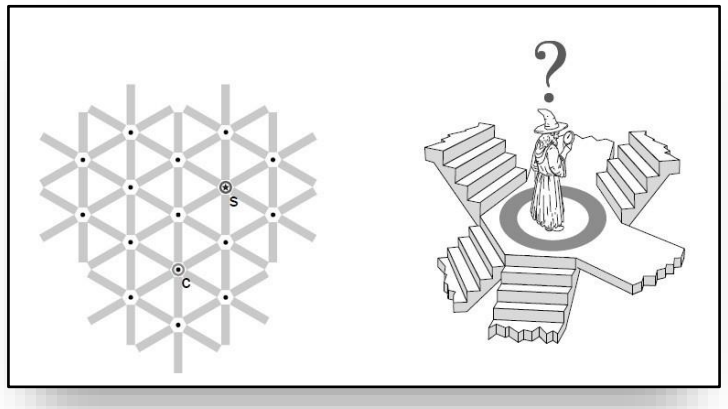


Basic Concepts of Local Search

- The **search space S** is organized as a network, by defining **the neighborhood relation**

  $\mathbf{N \subseteq S \times S}$ (usually characterized by **operator**: $s \xrightarrow{op} s'$)

- There is an **objective function** which needs to be minimized (maximized) that can be computed for each point



From “Stochstic Local Search: Foundations and Applications” (2005), Hoos and Stützle

The SAT Problem

- **Boolean satisfiability (SAT) --- the first problem proved to be NP complete**

SAT usually adopts the Conjunctive Normal Form (CNF):

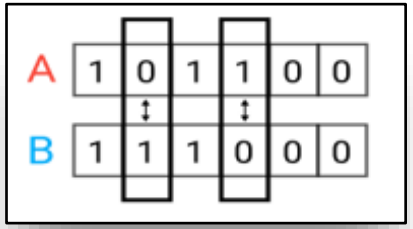
e.g., $\varphi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$

test whether there exists an assignment to the variables in φ that satisfies all the clauses.

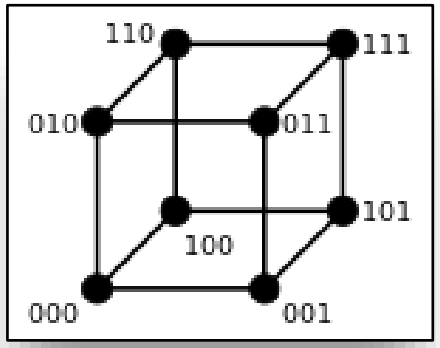
- Boolean variables have two possible values: True, False.
- A literal is a Boolean variable x or its negation $\neg x$
- A clause is a disjunction ($\vee$) of literals

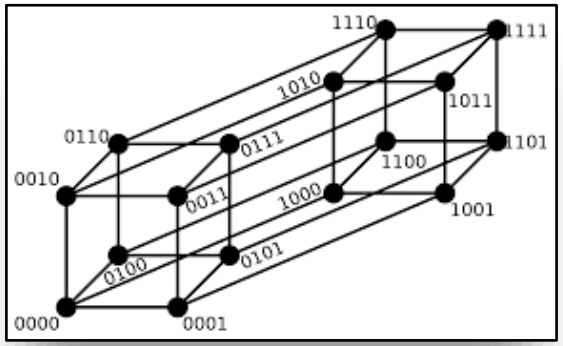
Example: SAT

- **objective function:** (minimize) the number of unsatisfied clauses.
- **search space**: for a SAT instance with n variables, there are 2^n complete assignments.
- Hamming distance between two assignments α and β
  $d(\alpha, \beta) = |\{x \mid \alpha(x) \neq \beta(x)\}|$



- a **neighborhood relation**: $\boldsymbol{N} = \{(\alpha, \beta) | \, d(\alpha, \beta) = 1\}$,
  equally described by the $flip$ **operator** (flipping the value of a variable).

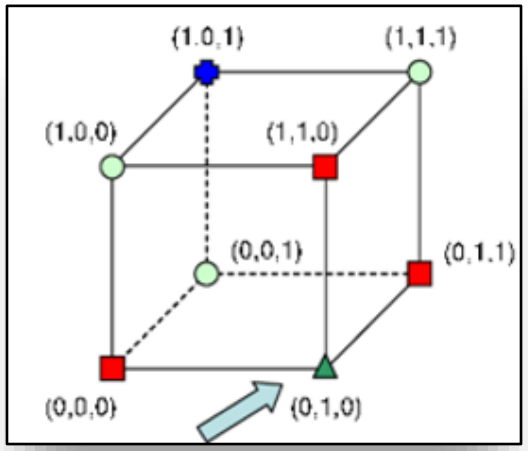




Basic Concepts of Local Search

How does local search work?

- starts from a candidate solution
- iteratively moves from current position to a neighboring position (modifies the candidate solution by an operation)



Choose which neighboring position (which operation to perform)?

- scoring functions
- search strategies

Basic Concepts of Local Search

- Pros:
  - simple and easy to implement
  - use limited memory
  - able to find reasonable solutions in large search space
  - easy to parallelize

- Cons:
  - often incomplete
  - often difficult to analyze theoretically

The Cycling Issue of Local Search

Local search suffers from the cycling problem, i.e., revisiting candidate solutions (stuck in plateaus, limited areas)

- wastes time
- prevents it from getting out of local minima

Cycling is an inherent problem of local search

- local search does not allow to memorize all previously visited parts of the search space.
- Addressing the issue is also related to the “intensification vs. diversification” balance of local search

Previous methods on dealing with cycling

Naive methods

- Random walk
- Non-improving search [simulated annealing]
- Restart

The tabu mechanism [Glover, ORSA J. Comput. 1989]

- forbids reversing the recent changes
- has a parameter called tabu tenure, which controls the strength of forbidding

Example 1 (SAT): if the value of a variable was changed from T to F, then it is forbidden to change from F to T in the next ***t*** steps.

Example 2 (Vertex Cover): if a solution vertex is removed, then it is forbidden to be selected in the next ***t*** steps.

Configuration Checking

Address cycling problem by Configuration Checking (CC)

[Cai et al. Artificial Intelligence 2011]

- Consider the circumstance information (formally defined as configuration) of the variables
- Reduce cycling by avoid local structure cycling

CC is particularly effective for two types of combinatorial problems:

- Assignment Problems: find an assignment to all variables to reach some goal(s).
  - SAT, MaxSAT, Graph Coloring
- Subset Problems: find a subset from the universal set to reach some goal(s).
  - Vertex Cover, Max Clique, Dominating Set

2

PART TWO

Configuration Checking for Assignment Problems

Local Search Framework for SAT

Algorithm : Local Search Framework for SAT

```
1 begin
2     α ← a complete assignment;
3     while not reach terminal condition do
4         if α satisfies F then return α;
5         pick a variable x;
6         α ← α with x flipped;
7     return "Solution not found";
```

Scoring functions

Search strategies

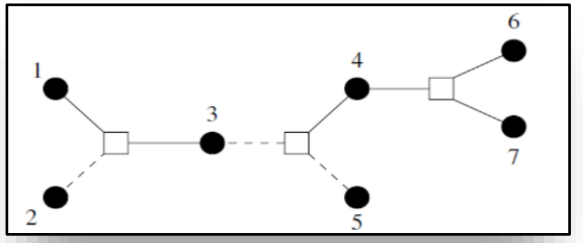
Configuration Checking for SAT

A key concept in the CC strategy is the configuration of variables.

- A typical definition: the configuration of a variable x is a vector consisting of truth value of all variables in $N(x)$.
- $\boldsymbol{N(x)} = \{y | y \; and \; x \; occur \; in \; at \; least \; one \; clause\}$

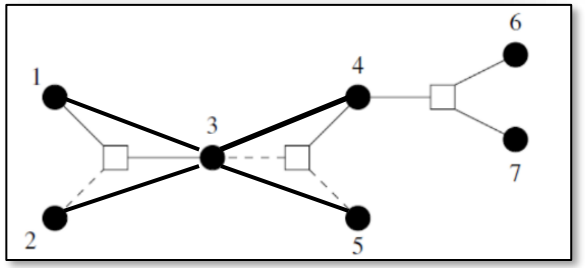
Example:

- $(\boldsymbol{x_1} \vee \neg \boldsymbol{x_2} \vee \boldsymbol{x_3}) \wedge (\neg \boldsymbol{x_3} \vee \boldsymbol{x_4} \vee \neg \boldsymbol{x_5}) \wedge (\neg \boldsymbol{x_4} \vee \boldsymbol{x_6} \vee \neg \boldsymbol{x_7})$
  $\alpha = 1010101$



Factor graph

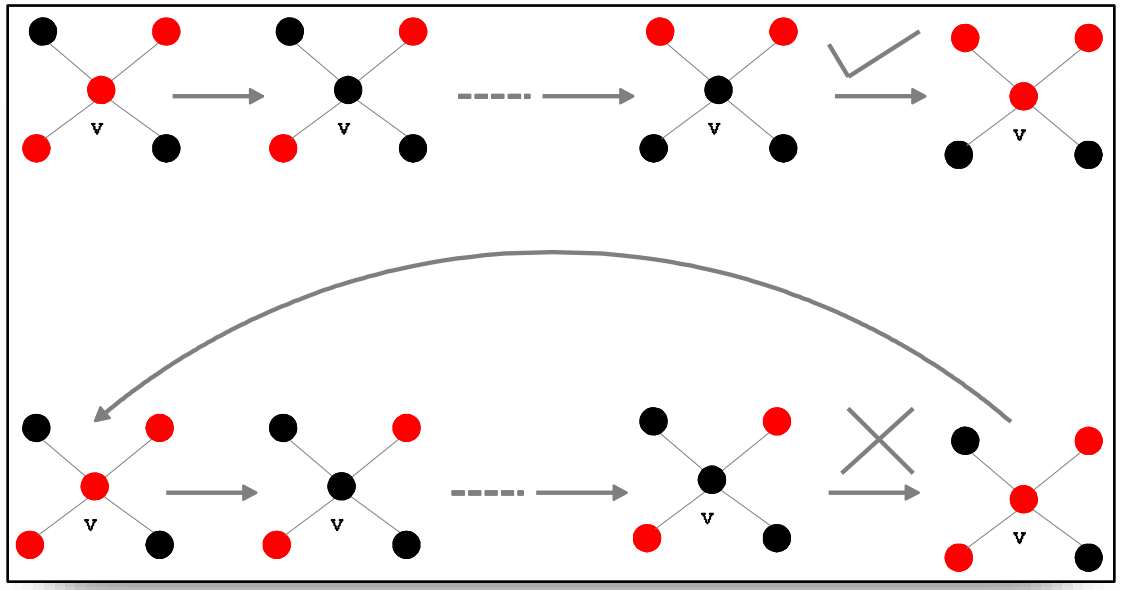
- the configuration of x_3 under α
  $< \alpha(x_1), \alpha(x_2), \alpha(x_4), \alpha(x_5) > = < 1,0,0,1 >$



$\boldsymbol{N(x_3)}$

Configuration Checking for SAT

- **CC for SAT**: if the configuration of x has not changed since x's last flip, then it should not be flipped.



Avoid local structure cycles

Shaowei Cai, Kaile Su: Local search for Boolean Satisfiability with configuration checking and subscore. Artif. Intell. (2013)

Configuration Checking for SAT

Naïve implementation of CC

Implement CC according to the definition

- Store the configuration for a variable x when it is flipped
- Check the configuration when considering flipping a variable

Complexity per step:

$\Delta(F) = \max\{\#N(x): x \in V(F)\}$

- It needs $O(\Delta(F))$ for both storing and checking the configuration of a variable.
- The worst case complexity of CC in each step is $O(\Delta(F)) + O(\Delta(F)n)$

Configuration Checking for SAT

An efficient Implementation of CC

Observation: when a variable is flipped, the configuration of all its neighboring variables has changed.

Data structure: CC array

- CC[x] = 1 means x meets the CC criterion (its configuration has changed since x's last flip).
- CC[x] = 0 on the contrary. → forbidden to be flipped.

Updating rules

- Rule 1: for each variable x, CC[x] is initialized as 1.
- Rule 2: when flipping x, CC[x] is reset to 0, and for each $y \in \mathrm{N}(x)$, CC[y] is set to 1.

Configuration Checking for SAT

An efficient Implementation of CC

Complexity of the implementation per step

- O(1) for checking whether a variable is configuration changed (check whether CC[x]=1).
- updating CC values for N(x) needs $O(\#(N(x))= O(\Delta(F))$

- Thus, the worst case complexity of CC in each step is $O(n) + O(\Delta(F))$
  - Suppose the number of candidate variables for flipping is $O(n)$, which is usually much smaller

Shaowei Cai, Kaile Su: Local search for Boolean Satisfiability with configuration checking and subscore. Artif. Intell. 204: 75-98 (2013)

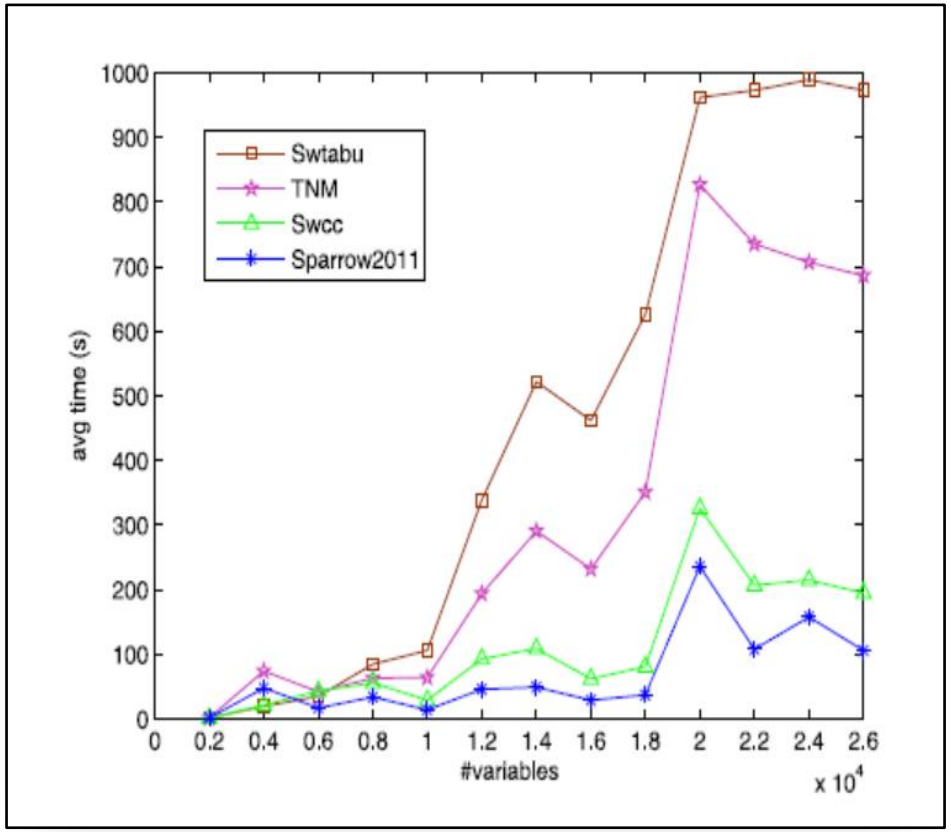
Configuration Checking for SAT

CC-based Variable Selection

- $score(x)$: the decrement on the number (or total weight) of unsatisfied clauses if x were to be flipped.
- $GoodVars = \{x | score(x) > 0 \ \& \ CC(x) = 1\}$

PickVar-CC

$if\,(GoodVars \neq \emptyset)$
    return a variable $x \in GoodVars$ with the highest $score$;
$else$
    pick a random unsatisfied clause c;
    return the oldest variable x from c;



Comparing Swcc with Swtabu and winners of random track of SAT Comp 2009 and 2011 on the 2011 competition 3-SAT benchmark

Configuration Checking for SAT

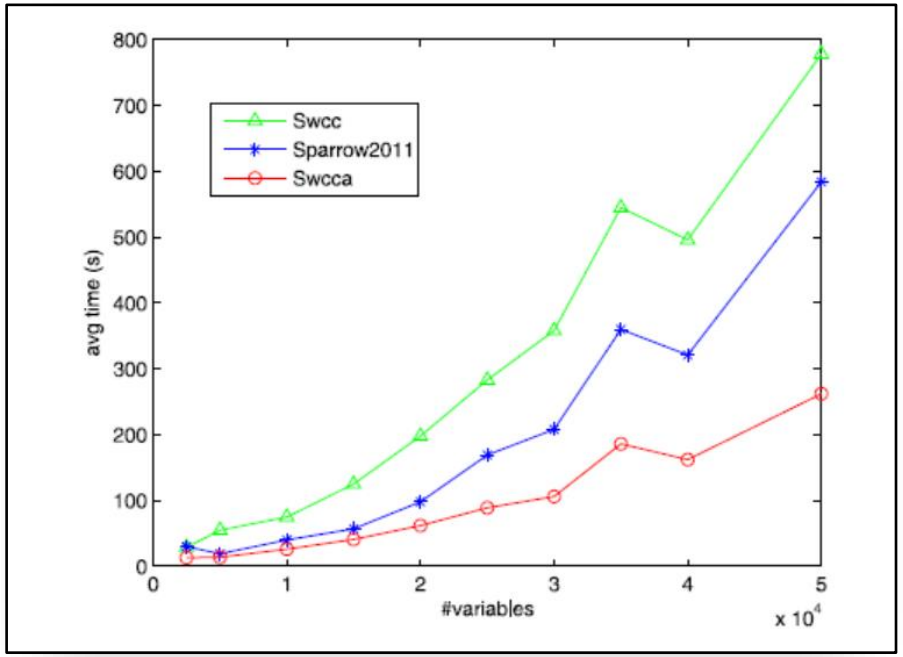
Configuration Checking with Aspiration

- The CC strategy forbids all variables failing to meet the CC criterion, regardless of the benefit its flip can bring.

→improve CC by introducing an **aspiration** mechanism

PickVar-CCA

$if(GoodVars \neq \emptyset)$
  return a variable $x \in GoodVars$ with the highest $score$;
$else\ if(\{x|score(x) > threshold\_score\} \neq \emptyset)$
  return such a variable with the highest $score$;
$else$
  pick a random unsatisfied clause c;
  return the oldest variable x from c;



Comparing Swcc, Swcca and winner of SAT-Comp 2011 random track on 3-SAT benchmark

Configuration Checking for SAT

CCASat won the random track of SAT Challenge 2012 with a large margin.

- 423(70.5%) vs 321(53.5%)

Set	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$
1	4.2 40000	9 10000	20 1600	40 400	85 200
2	4.208 35600	9.121 8800	20.155 1420	40.674 360	85.558 180
3	4.215 31400	9.223 7800	20.275 1280	41.011 340	85.837 170
4	4.223 27200	9.324 6800	20.395 1140	41.348 320	86.116 160
5	4.23 23000	9.425 5800	20.516 1000	41.685 300	86.395 150
6	4.237 18800	9.526 4800	20.636 860	42.022 280	86.674 140
7	4.245 14600	9.627 3800	20.756 720	42.359 260	86.953 130
8	4.252 10400	9.729 2800	20.876 580	42.696 240	87.232 120
9	4.26 6200	9.83 1800	20.997 440	43.033 220	87.511 110
10	4.267 2000	9.931 800	21.117 300	43.37 200	87.79 100

CCAnr: well known for its good performance on solving non-random instances

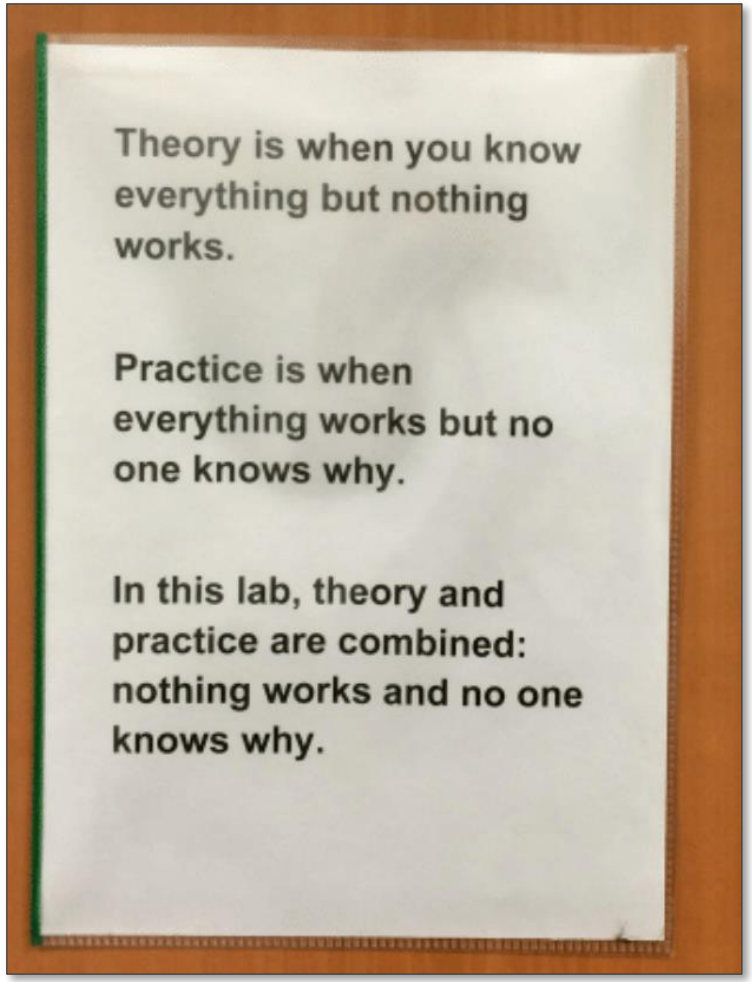
- CCAnrglucose ranked 2nd in the hard-combinatorial track of SAT Competition 2014.
- Integrated in some winners of main track in recent SAT Competitions.

Configuration Checking for SAT

OK, the CC strategy works practically for SAT.

Any theoretical analysis?

- Why does it work?
- Can it lower the complexity of local search?



Configuration Checking for SAT

The effectiveness of CC is related to the neighborhood of variables.

Consider an extreme case: each variable is a neighbor to all other vars.

→ CC becomes trivial and is equal to tabu method with t=1.

Theorem: For random k-SAT model $F_k(n, r)$, the neighborhood based CC becomes trivial that forbids only one variable when $ln(n-1) < \frac{k(k-1)r}{n-1}$ (r is the clause-variable ratio).

- Let $f(n) = ln(n-1) - \frac{k(k-1)r}{n-1}$, $f(n)$ monotonically increase with n (n>1).

  Thus, f(n)<0 iff $n \leq \lfloor n^* \rfloor$, where $f(n^*) = 0$

Formulas	3-SAT ($r = 4.2$)	4-SAT ($r = 9.0$)	5-SAT ($r = 20$)	6-SAT ($r = 40$)	7-SAT ($r = 85$)
n^*	11.652	32.348	90.093	223.095	564.595

Variants of Configuration Checking for SAT

Different definitions of configuration → different CC variants

- **Clause-based CC**: configuration of a variable x is a vector consisting of status of all clauses in $CL(x)$.

Example:

- $(\boldsymbol{x_1} \vee \neg \boldsymbol{x_2} \vee \boldsymbol{x_3}) \wedge (\neg \boldsymbol{x_3} \vee \boldsymbol{x_4} \vee \neg \boldsymbol{x_5}) \wedge (\neg \boldsymbol{x_4} \vee \boldsymbol{x_6} \vee \neg \boldsymbol{x_7})$
  $\alpha = 1010101$

- $\boldsymbol{x_1} \vee \neg \boldsymbol{x_2} \vee \boldsymbol{x_3}$: satisfied
  $\neg \boldsymbol{x_3} \vee \boldsymbol{x_4} \vee \neg \boldsymbol{x_5}$: unsatisfied

- the configuration of x_3 under α
  $< s(C_1), s(C_2) > \quad = < \text{satisfied}, \text{unsatisfied} >$

FrwCB: with clause-based CC

Solve random 3-SAT near phase transition (r=4.2) with 4 million vars within 3hs, while other algorithms failed at 2 million vars

When combined with SP, can reach to 10 millions.

Chuan Luo, Shaowei Cai, Wei Wu, Kaile Su: Focused Random Walk with Configuration Checking and Break Minimum for Satisfiability. **CP** 2013: 481-496

Variants of Configuration Checking for SAT

Different ways of checking

- Quantitative CC: the CC value counts the times that a variable's configuration has changed since its last flip.

Chuan Luo, Kaile Su, Shaowei Cai: Improving Local Search for Random 3-SAT Using Quantitative Configuration Checking. ECAI 2012: 570-575

Combine with other heuristics

- The Novelty heuristic + CC → NCCA
- Combining different CC strategies automatically

André Abramé, Djamal Habet, Donia Toumi: Improving configuration checking for satisfiable random k-SAT instances. Ann. Math. Artif. Intell. 79(1-3): 5-24 (2017)

Chuan Luo, Holger Hoos, Shaowei Cai: PBOCCSAT: Boosting Local Search for SAT via Programming by Optimization

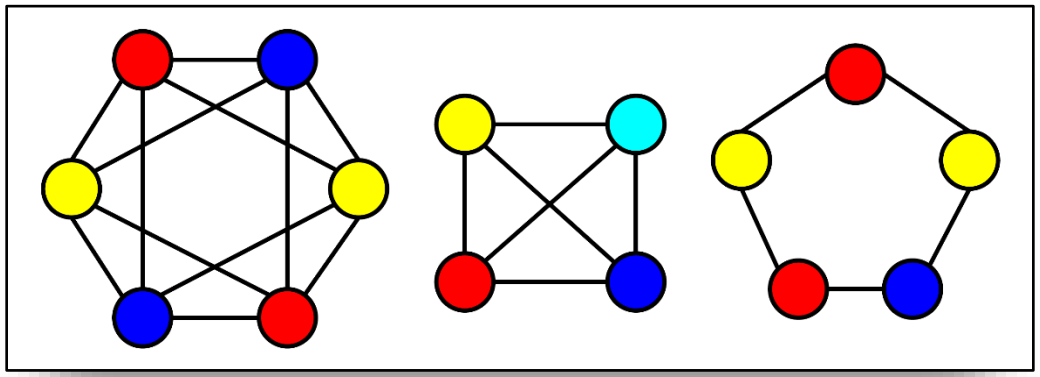
Variants of Configuration Checking for SAT

CC has been widely used in local search for SAT and MaxSAT, including at least 10 medal awarding solvers.

- CCASat
- CCAnr
- NCCA+
- CCLS
- CCEHC
- SC2016
- ….

Graph Coloring Problem

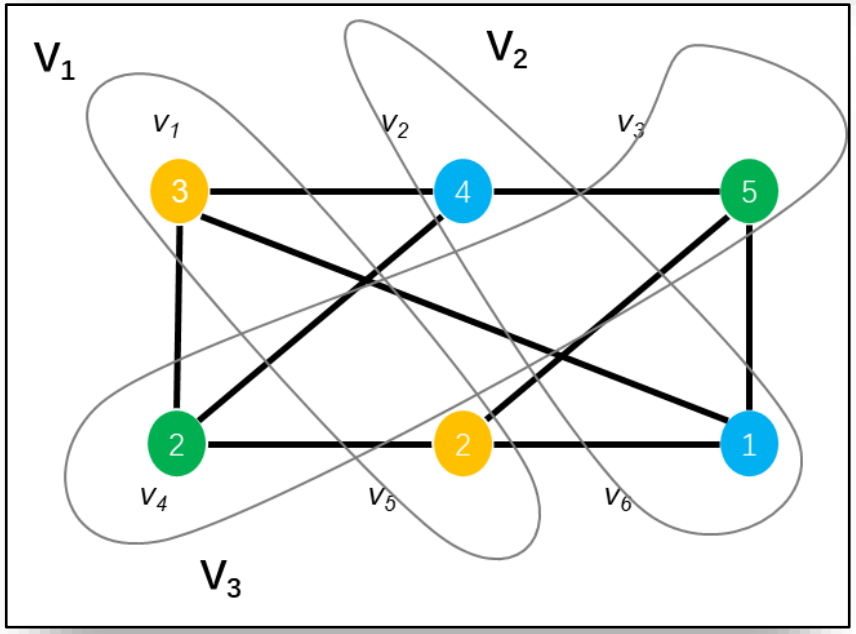
- A coloring is an assignment of colors to vertices such that no two adjacent vertices share the same color.
- A coloring can be viewed as a partition of the vertex set V into independent sets {V1, V2,…, Vk } such that no two adjacent vertices are in the same Vi.



- The Graph Coloring Problem (GCP) is to find a coloring of a graph while minimizing the number of colors.

Weighted Graph Coloring Problem

WGCP is to find a feasible coloring S= $\{V_1, V_2, \ldots, V_k\}$ which minimizes $\text{cost}(S) = \sum_{i=1}^{k} max_{v \in V_i} w(v)$.



$$S = \{V_1, V_2, V_3\} = \{\{v_1, v_5\}, \{v_2, v_6\}, \{v_3, v_4\}\}$$

$$\text{cost}(S) = \sum_{i=1}^{3} max_{v \in V_i} w(v) = 3 + 4 + 5 = 12$$

Local Search Operator for WGCP

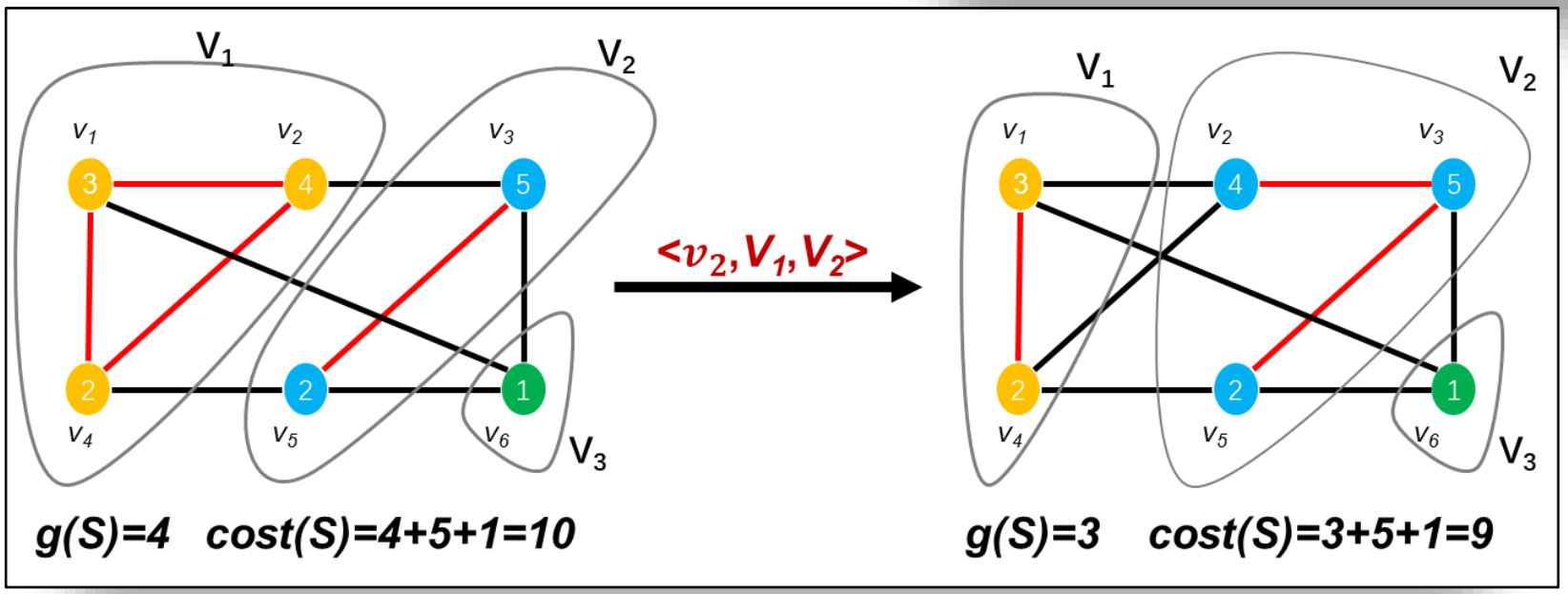
The operator move<v,V_i,V_j>

- move v from its color class V_i to another color class V_j

Measure the benefits of operations

- Number of conflicts: g(S)
- Value of the cost function: cost(S)

A good operation reduces both g(S) and cost(S).



Configuration Checking for WGCP

Original CC:

Data structure: CC array

- CC[v] = 1 means v meets the CC criterion (its configuration has changed since v changed the value).
- CC[v] = 0 on the contrary. → forbidden to change color.

Updating rules

- Rule 1: for each vertex v, CC[v] is initialized as 1.
- Rule 2: when changing the color of v, CC[v] is reset to 0, and for each $y \in \mathrm{N}(v)$, CC[y] is set to 1.

Configuration Checking for WGCP

Use CC as an "encouraging" mechanism to explore the sequential operations following good operations.

An operation is good if it reduces both g(S) and cost(S).

***When a good* operation is performed**

- *Moving v to a more suitable set, it is reasonable to adjust the colors of its neighbors in the following steps.*

→ *encourage vertices in* $N(v)$ *to be moved by setting their cc values to 1.*

For other operations

- we do not modify *cc* values

Configuration Checking for WGCP

CC rule for WGCP:

After performing *an* operation move<v,V_i,V_j>

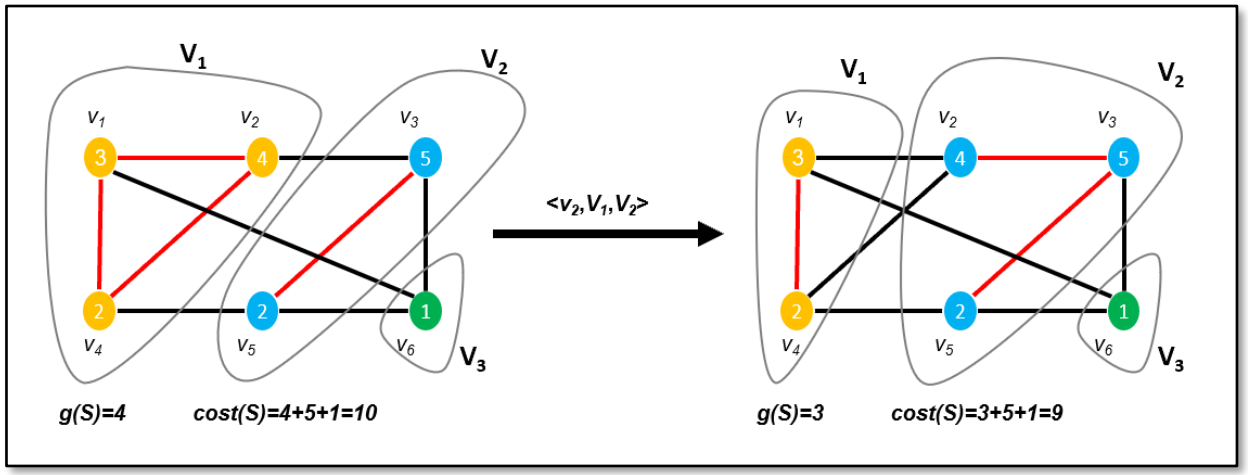
- $cc[v]$ is set to 0;
- if move<v,V_i,V_j> is a good operation, $cc[y]$ is set to 1 for $y \in N(v)$.

Example:

After performing $< \boldsymbol{v_2}, \boldsymbol{V_1}, \boldsymbol{V_2} >$

Update cc values of v_2 and its neighbors

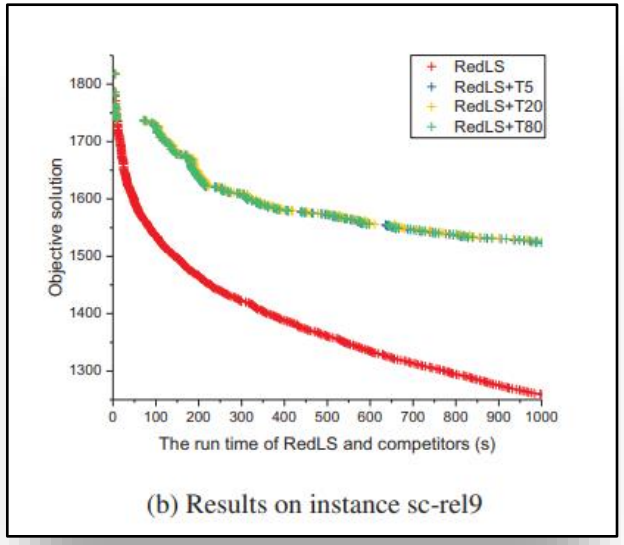
$$\begin{cases} cc[v_2] := 0 \\ cc[v_1] := cc[v_3] := cc[v_4] := 1 \end{cases}$$



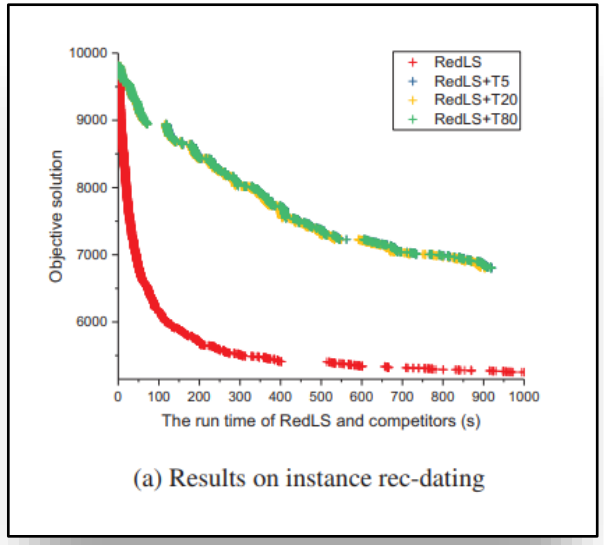
Configuration Checking for Graph Coloring

Our CC based local search WGCP algorithm:

- On COLOR and matrix decomposition benchmarks (59 instances): outperforms AFISA [Sun et al. 2018] and MWSS [Cornaz, Furini, and Malaguti 2017] on **48** and **55** instances, respectively.
- On large graphs from Network Data Repository (65 with more than 10^5 vertices): obtains better solutions on **60** instances.
- Compare with alternative versions replacing CC with tabu (under different t parameters)



(b) Results on instance sc-rel9



(a) Results on instance rec-dating

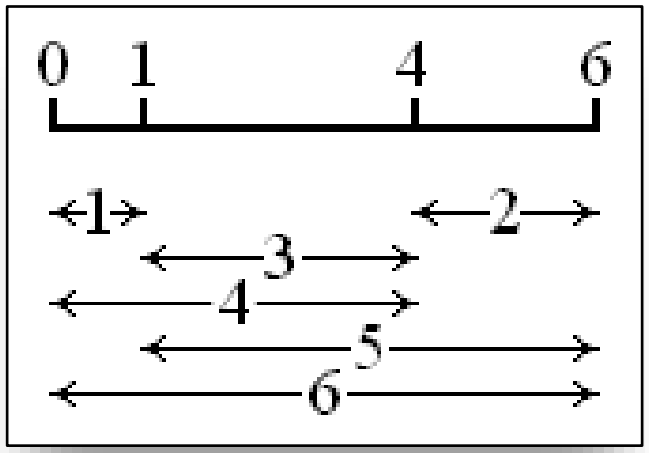
Particularly better on sparse graphs.

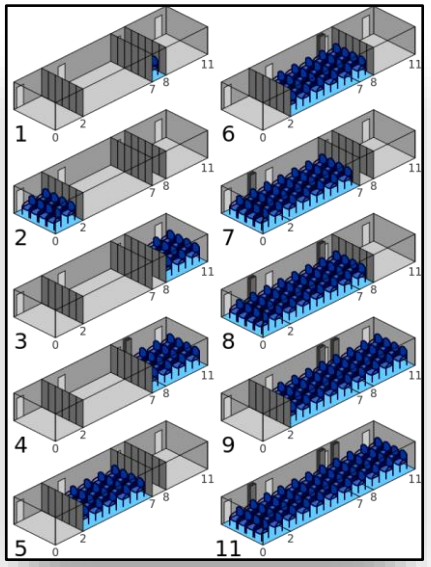
Optimal Golomb Ruler

A Golomb ruler G of order m and length n comprises m integers $x_1 < x_2 < \cdots < x_m$ such that each difference $d_{ij} = x_i - x_j$ where $i > j$ is unique.

Note that by definition $n = x_m - x_1$ and without loss of generality, one can easily assume $x_1 = 0,\ x_m = n$.

OGR is optimal if its length is the minimum for the given order.





Example of a conference room with proportions of a [0, 2, 7, 8, 11] Golomb ruler, making it configurable to 10 different sizes.

Optimal Golomb Ruler

Finding an optimal Golomb ruler (OGR) is an extremely challenging problem.

- the search for a 19 marks OGR took approximately 36,200 CPU hours on a Sun Sparc workstation (1998)
- it took 5 years to find an OGR of 27-mark by 19,919 participants. It has been stated by Cotta et al. about the distributed.net projects (2007)

Table 1 A 16-mark optimal Golomb ruler with marks $(x_k)_{1 \le k \le m}$

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
x_k	0	1	4	11	26	32	56	68	76	115	117	134	150	163	168	177

Configuration Checking for OGR

A constraint-based local search algorithm uses both tabu and CC performs very fast on solving OGR problems.

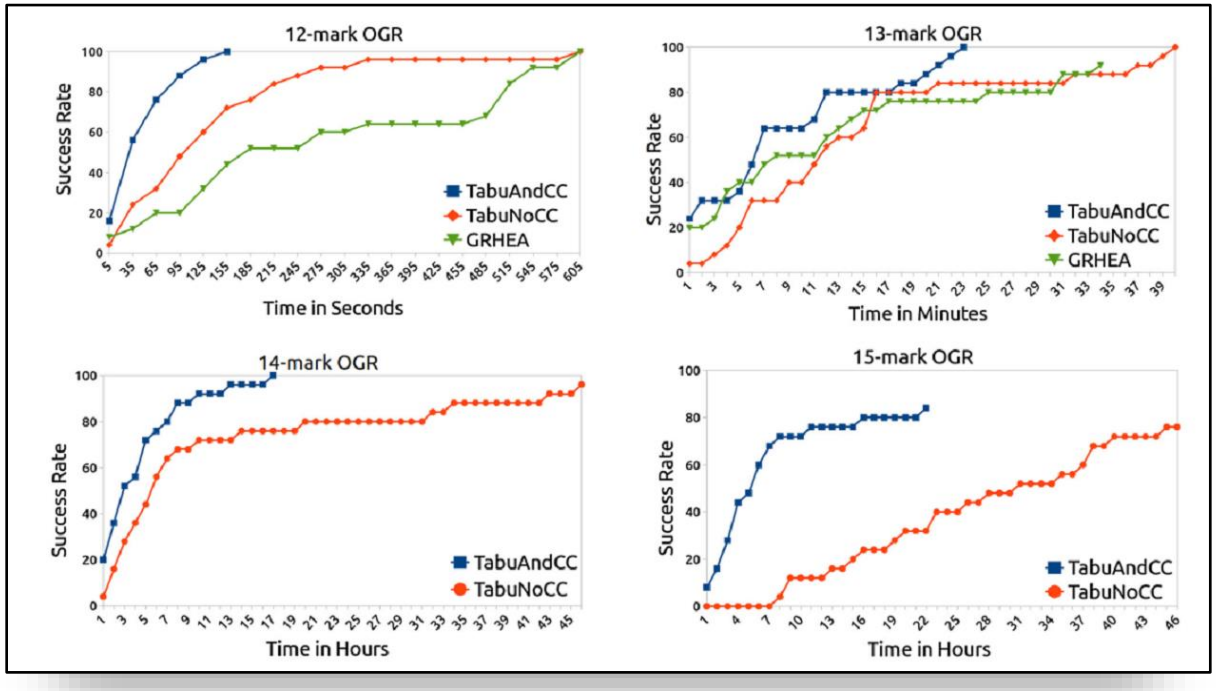
- Finds the optimal solution of 19-mark OGR in 40 CPU hours. Much faster.

CC is used when the domain of a variable xi becomes one (i.e., containing only its current value).

- This might happen because the marks are all sorted and the domain of xi is dynamically restricted [keeping feasibility].
- the variable is kept locked for any future changes until any of its neighbouring variables have changed their values.

Configuration Checking for OGR

The version that uses both tabu and CC better than only using tabu



"the use of CC effectively reduces the number of restarts required during search and thus mitigates the cycling problem of local search for OGR."

The algorithm restarts when the platueau search steps achieve a threshold.

```
05    if (plateauSize > MaxPlateauSize)
06        Restart(solution), plateauSize = 0
```

Md. Masbaul Alam, M. A. Hakim Newton, Abdul Sattar: Constraint-based search for optimal Golomb rulers. J. Heuristics (2017)

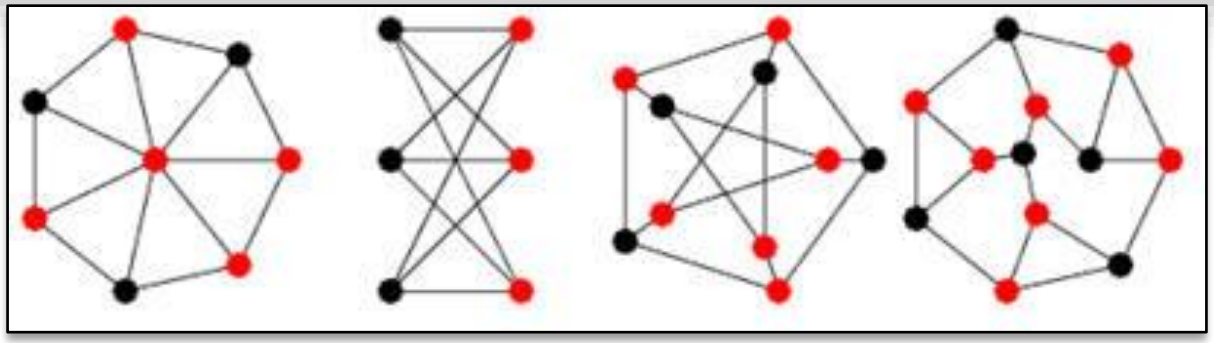
3

PART THREE

Configuration Checking for Subset Problems

The Vertex Cover Problem

Vertex Cover: Given an undirected graph G = (V,E), a vertex cover is a subset $S \subseteq V$ such that every edge in G has at least one endpoint in S.



Red vertices constitute a vertex cover

- Minimum Vertex Cover (MinVC): to find a vertex cover of the smallest size.
- K-Vertex-cover: find a k-sized vertex cover for a given k.

A Local Search Framework for MinVC

An iterative framework for MinVC

- A popular framework is to solve MinVC by solving its decision problem iteratively.

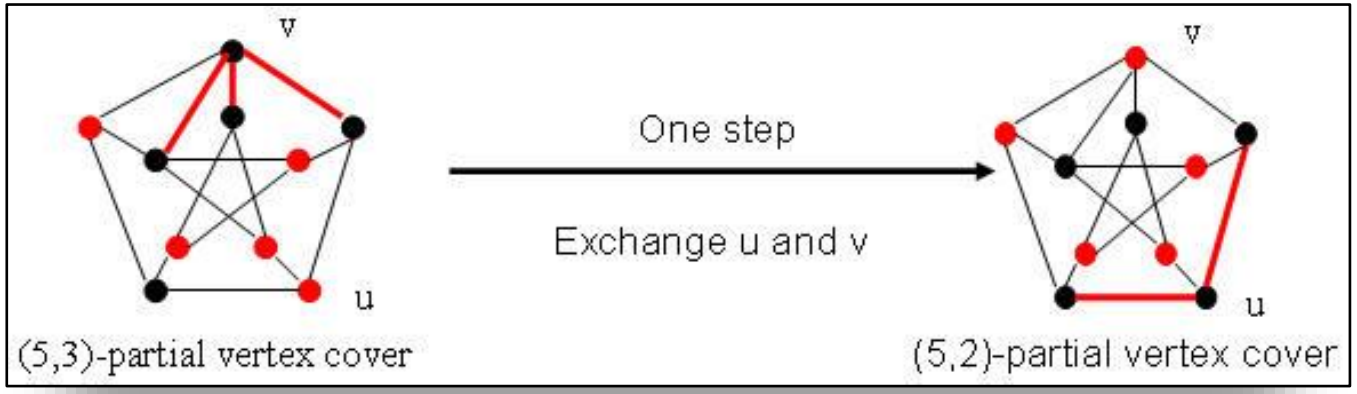
→ whenever finding a k-sized vertex cover, the algorithm goes on to search for a (k-1)-sized vertex cover.

- So, the core is to solve the K-vertex-cover problem.

A Local Search Framework for MinVC

Solving the K-vertex-cover problem

- Maintain a candidate solution S of k vertices, and modify the set by exchanging vertices, trying to cover all edges.



Choose which two vertices to exchange?

- Which to remove
- Which to add

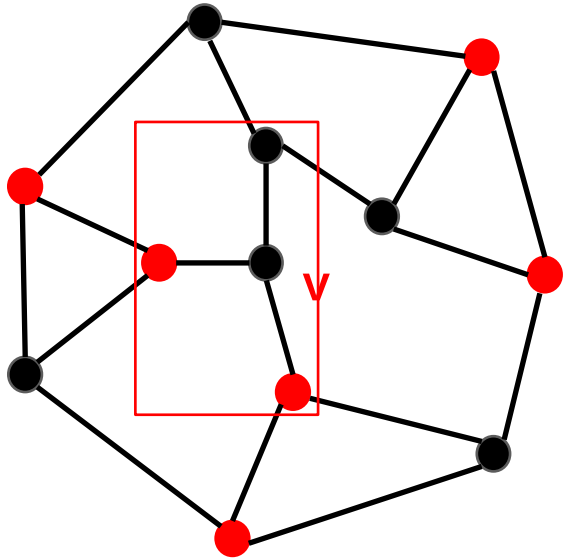
Scoring function:

$$score(v) = cost(C) - cost(C')$$

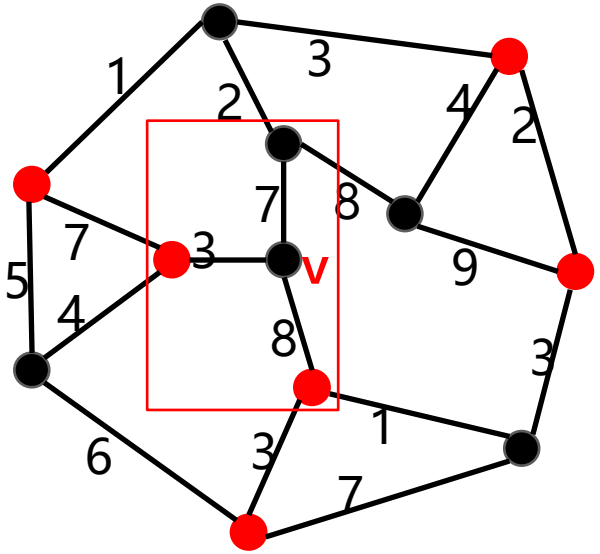
C' is obtained from C by changing the state of v

Configuration Checking for Vertex Cover

Configuration of v



Only considering N(v)



Considering N(v) and edge weights, when employing edge weighting tech.

Configuration Checking for Vertex Cover

Exchange

choose a vertex $u \in C$ greedily w.r.t score;

$C := C \setminus \{u\}$;

choose an uncovered edge e randomly

choose a vertex $u \in e$ satisfying CC criterion, and with higher score;

$C := C \cup \{v\}$;

Updating rules for CC array :

- Rule 1: all CC[v] are initialized to 1.
- Rule 2: When removing a solution vertex v, CC[v] is reset to 0.
- Rule 3: When a vertex v changes its state, for each $y \in N[v]$, CC[y] is set to 1.
- Rule 4 (optional): When updating the weight of edge e(u, v), CC[u] and CC[v] are set to 1.

Configuration Checking for Vertex Cover

NuMVC

- a popular state of the art local search algorithm for MinVC. Currently, it is among best-performing solvers on popular benchmarks, even it was published 8 years ago.

Table 1: Experiment results of NuMVC and NuMVC without CC on the representative DIMACS and BHOSLIB benchmarks.

Instance	$\|V\|$	initial size	NuMVC			NuMVC without CC		
			#iter	#repeat	#repeat/#iter	#iter	#repeat	#repeat/#iter
brock800_4	800	774	10000000	185445	**0.01854**	10000000	1627269	0.16273
C2000.9	2000	1920	10000000	72708	**0.00727**	10000000	1337073	0.13371
DSJC1000.5	1000	985	51024	18	**0.00035**	15588	2166	0.13895
gen400_p0.9_75	400	325	3403	0	**0.00000**	2516	19	0.00755
hamming10-4	1024	984	46332	163	**0.00352**	19080	3676	0.19266
keller6	3361	3302	429550	716	**0.00167**	1101418	245155	0.22258
MANN_a81	3321	2221	10000000	5355	**0.00054**	10000000	526714	0.05267
p_hat1500-3	1500	1406	17699	728	**0.04113**	77922	48932	0.62796
frb59-26-1	1534	1475	10000000	43559	**0.00436**	10000000	2559511	0.25595
frb59-26-2	1534	1475	10000000	43968	**0.00440**	10000000	2594888	0.25949
frb59-26-3	1534	1475	10000000	44915	**0.00449**	10000000	2412643	0.24126
frb59-26-4	1534	1475	10000000	45687	**0.00457**	10000000	2433095	0.24331
frb59-26-5	1534	1475	10000000	45615	**0.00456**	10000000	2514596	0.25146
frb100-40	4000	3900	10000000	2800	**0.00028**	10000000	902380	0.09024

NuMVC uses greedy heuristics when choosing vertices, which tends to make it easily stuck at local optima, while the equipment of CC significantly reduces the cycling.

Shaowei Cai, Kaile Su, Chuan Luo, Abdul Sattar: NuMVC: An Efficient Local Search Algorithm for Minimum Vertex Cover. J. Artif. Intell. Res. (2013)

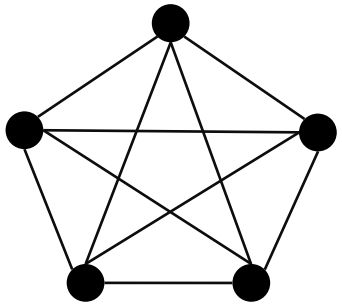
Max Clique and Its Relaxed Versions

Clique, k-plex and quasi-clique

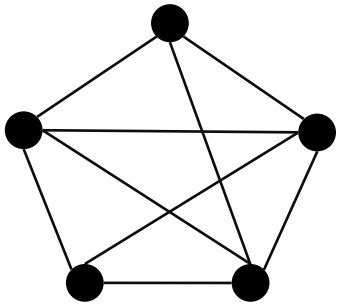
- K-plex: a subset $S \subseteq V$ is a k-plex, if $|N(v) \cap S| \geq |S| - k$ for all $v \in S$. (each vertex at most misses k-1 connections)
- λ-quasi-clique: is a subset $S \subseteq V$ such that the edge density of the subgraph induced by S is at least λ.

Each vertex at most misses 1 connection in the clique

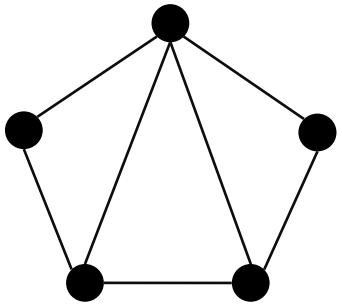
edge density of the subgraph is 7 / 10 = 0.7.



5-clique



2-plex with 5 vertices



0.7-quasi clique

Configuration Checking for Max (Weight) Clique

Three operators:

- Add: add a vertex while maintaining feasibility
- Swap: remove a solution vertex and add another vertex, keeping feasibility
- Remove: remove some solution vertices

CC is used as an encouraging mechanism on good operations

- The configuration of a vertex is defined as its neighborhood.
- When a good operation is performed, it causes CC updating.
- Otherwise, no CC updating.

Configuration Checking for Max (Weight) Clique

SCCWalk:

- state of the art performance on a wide range of benchmarks

Comparative results for SCCWalk and five competitors on the classical benchmarks.

Benchmark	*#inst*	ReTS1 *#win*	ReTS2 *#win*	FastWCIq *#win*	RRWL *#win*	WLMC *#win*	SCCWalk *#win*
DIMACS	80	78	77	63	77	63	**79**
BHOSLIB	40	38	38	0	29	1	**40**
WDP	22	17	17	11	17	15	**22**
DGCA	7	5	5	2	6	5	**7**
Total	149	138	137	76	129	84	**148**

Comparative results for SCCWalk4L and five competitors on the massive graphs.

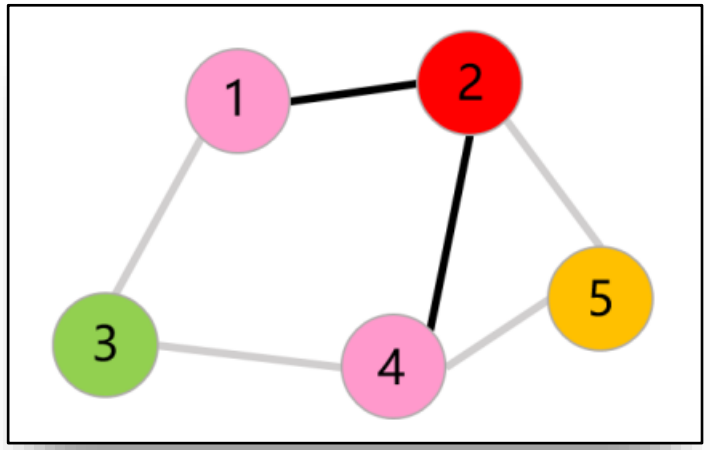
Benchmark	*#inst*	ReTS1 *#win*	ReTS2 *#win*	FastWCIq *#win*	RRWL *#win*	WLMC *#win*	SCCWalk4L *#win*
real-world massive graphs	137	48	48	123	133	**137**	**137**
large-scaled FRB	20	7	3	0	0	0	**20**
Total	157	55	51	123	133	137	**157**

Wang et. al. SCCWalk: An efficient local search algorithm and its improvements for maximum weight clique problem. Artif. Intell. (2020)

Local Search for Max K-plex

Three operators:

- *Add: add a vertex while maintaining S feasible*
- *Swap: remove a solution vertex and add another vertex, keeping S feasible*
- *Remove: remove some solution vertices*



$S = \{1,2,4\}$

$SaturatedSet = \{1,4\}$

$AddSet(S) = \{3\}$

$SwapSet(S) = \{5\}$

Dynamic Configuration Checking

Recall that the original CC becomes ineffective on dense instances, as analyzed on random k-SAT formulas.

Address this drawback: use dynamic threshold CC

Configuration: the neighborhood of v.
Updating Rule

- A vertex v is locked (CC[v]:=0) when it is removed from S.
- When a vertex changes its state, CC[y]++ for each $y \in N(v)$.
- A vertex v is unlocked if CC[v]>threshold[v].
    - The threshold[v] depends on the degree of v and the frequency of v being selected.

→ to increase the forbidding strength on the vertices that are frequently selected.

Configuration Checking for K-plex and Quasi-clique

Dynamic CC based local search algorithms push the state of the art for Max K-plex and Quasi-clique problems

- good performance on graphs with various densities.
- Best-performing on DIMACS, BHOSLIB and large graphs from Network Repository.

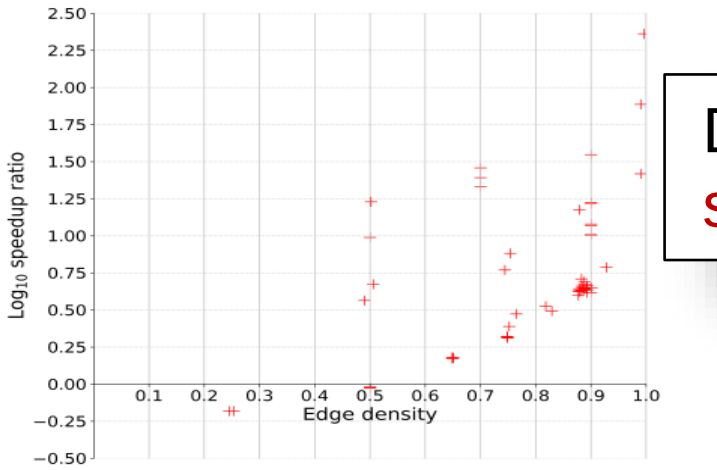


Figure 2: Speedup ratio varying with edge density

Dynamic CC enables self-adaptive forbidding strength, robust on graphs with various density.

Chen et. al: Local Search with Dynamic-threshold Configuration Checking and Incremental Neighborhood Updating for Maximum k-plex Problem. AAAI (2020)
Chen et. al: NuQClq: An Effective Local Search Algorithm for Maximum Quasi-Clique Problem. AAAI (2021)

4

PART FOUR

Conclusions and Challenges

Conclusions and Challenges

Configuration Checking is a generic idea for local search algorithms.

- Core idea: for a solution component (a variable, or a vertex), if after the value is changed or it is removed, its circumstance information remains unchanged, then this component (the value) is forbidden to be put in the solution again.

CC particularly works well for assignment problems and subset problems.

- You may have a try if you are solving similar problems.

Conclusions and Challenges

A list of problems using CC:

SAT // winners in SAT COMPs
MaxSAT //winners in MaxSAT Evaluations
SMT(IDL) // YicesLS 1st in QF_IDL theory of SMT COMP 2021
Abstract Argumentation

Graph Coloring Problem
MaxClique and variants
Vertex Cover and variants
Dominating Set
Set Cover
Packing Problems

Black and White Problem
Combinatorial Auction
Golomb Rule Problem
…

Nearly 70 papers,
pushing state of the art in many combinatorial problems.

Conclusions and Challenges

The intuition of CC is that by reducing cycling on local structure, we reduce the cycling of local search

- Or, is it the main reason for its power?
- Do we have better understanding on this idea?
- Any theoretical/experimental analysis?

It has been successfully used to assignment problems and subset problems

- Can it be used to other kinds of combinatorial problems, e.g., permutation problems?

THANK YOU

caisw@ios.ac.cn