



高级算法 设计与分析

2023年3月

吕志鹏

zhipeng.lv@hust.edu.cn

群名称:

群 号:



Chapter 7

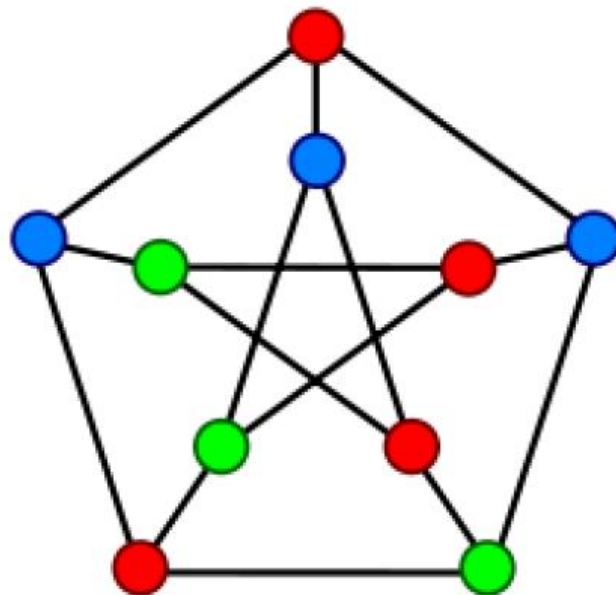
Graph coloring problem

图着色

7.1 问题概述

问题描述

给出一个无向图 $G = (V, E)$, 其中 V 为节点集合, E 为边集。图着色问题 (Graph coloring problem, GCP) 是为每一个节点分配一个颜色 c_i ($1 \leq c_i \leq k$), 要求相邻的节点之间染不同的颜色并且使用的 k (颜色数) 最小 (少)。





优化 or 判定?

优化版本 (**NP-hard**) :

寻找最小的颜色数 (k) ;

判定版本 (**NP-Complete**) (k -着色) :

给定 k 中颜色, 判断是否存在一种着色方式, 使得所有的相邻节点都满足约束。

注: 通过一系列 k 值逐渐减小的**GCP**判定版本, 可以求解**GCP**的优化版本, 因此它们是相互等价的。



求解过程：

1. 从一个初始颜色数 k 出发，求解 k -着色问题。一旦 k -着色问题被求解， $k = k - 1$ ，再次求解 k -着色问题。
 2. 重复动作，知道找不到一个合法的 k -着色。
- 由于 k 越小， k -着色问题的求解越困难，因此，以上描述的求解策略解决了一个困难度逐渐增加的 k -着色问题。
 - 在接下来的求解过程中，我们仅仅考虑 k -着色问题。



数学模型-判定版:

$$\max \quad 0$$

$$s. t. \quad \sum_{c=1}^k x_{ic} = 1, \forall i \in V$$

$$x_{ic} + x_{jc} \leq 1, \forall c \in K, (i, j) \in E$$

$$x_{ic} \in \{0, 1\}$$

- 给出 k 种颜色, x_{ic} 是决策变量, 为1表示节点 i 染颜色 c , 否则为0
- 第一条约束表示每一个节点只能染一种染色
- 第二条约束表示相邻的节点必须着不同的颜色



解的表示方式

■ 赋值表示：

- k -着色问题的解可以被表示为：每个节点所染的一系列

颜色集： $S = \{c_1, c_2, \dots, c_n\}$

其中， c_v 表示节点 v 所染的颜色，

且对于任意的 $(u, v) \in E, c_u \neq c_v$ 。

- 注：该方法表示比较自然，但并不直接和本质。



解的表示方式

■ 分组表示:

- k -着色问题的可行解可以被表示为一系列的**独立集**（一个独立集由一组非相邻节点集构成）。
- 具体表示如下：
 - $S = I_1 \cup I_2 \cup \dots \cup I_k$, 其中
 1. 对于任意的节点 j 和 l , 满足 $I_j \cap I_l = \emptyset$
 2. $I_1 \cup I_2 \cup \dots \cup I_k = V$
 3. 对于任意的 j , I_j 是一个独立集。
- 因此, k -着色问题变成了将 N 个节点**分割**为 k 个独立集。



应用

- 电台频率分配 (Mobile radio frequency assignment)
- 制定时间表：教育、交通、运动会
- 寄存器分配 (Register allocation)
- 人员排班 (Crew scheduling)
- 印刷电路板测试 (Printed circuit testing)
- 空中交通流量管理 (Air traffic flow management)
- 卫星调度 (Satellite range scheduling)
- 波长路由分配 (Routing and wavelength assignment in WDM networks)

Constructive Greedy Algorithms

- 是第一个解决图着色问题的启发式方法，它通过一个贪心函数逐个的引导节点染色。
- 该类算法是非常快速的，但解的质量不尽人意。
- 这类算法中最著名的算法包括：
 1. the **largest saturation degree** heuristic (DSATUR) (D. Brelaz, 1979)
 2. **recursive largest first** heuristic (RLF) (F.T. Leighton, 1979)
- 这些算法通常用于生成一个高质量的初始解。

■ Local Search Algorithms

- 一个典型的例子是Tabucol算法，这是禁忌搜索算法在图着色中的第一个应用。(Hertz, de Werra, 1987)
- 其他局部搜索算法包括：
 - Simulated Annealing (Johnson et al, 1991)
 - Iterated Local Search (Chiarandini and Stutzle, 2002)
 - Reactive Partial Tabu Search (Blochliger, Zufferey, 2008)
 - GRASP (Laguna, Marti, 2001)
 - Variable Neighborhood Search (Avanthay et al, 2003)
 - Variable Space Search (Hertz et al, 2008)
 - Clustering-Guided Tabu Search (Porumbel, Hao, 2009)
- 文献[Galinier, Hertz, 2006]对局部搜索算法做了详细调查
- 之后将会对著名的Tabucol算法做详细的介绍

■ Hybrid Evolutionary Algorithms

- 混合进化算法是近年来求解组合优化算法很有效的方法之一
- 混合进化算法是指将局部搜索算法嵌入到进化算法的框架中，以实现集中性（intensification）和多样性（diversification）之间的平衡，如：
 - Dorne, Hao, 1998
 - Galinier, Hao, 1999
 - Galinier, Hertz, Zufferey, 2008
 - Malaguti, Monaci, Toth, 2008
 - Porumbel, Hao, Kuntz, 2009



7.2 局部搜索

■ 搜索空间

- 这里使用 ***k -fixed penalty strategy***，该策略被用在很多着色算法中。
- 对于图 $G = (V, E)$ ，颜色数 k 是固定的，搜索空间包括所有可能的（合法或不合法）染色情况。
- k -着色可以表示为 $S = \{V_1, V_2, \dots, V_k\}$, V_i 表示所染 i 颜色的节点的集合。
- 因此，如果所有的 V_i 都是独立集，则 S 是一个合法的 k -着色。否则， S 是不合法（有冲突）的。



评估函数

- k -着色问题的**目标函数**：最小化**冲突边**（参见后面的**冲突数**）和在搜索空间内发现一个合法 k -着色。
- 给定一个 k -着色 $S = \{V_1, \dots, V_k\}$ ，评估函数 f 计算了由 S 造成的冲突数，

$$f(S) = \sum_{\{u,v\} \in E} \sigma_{uv}$$

$$\text{其中, } \sigma_{uv} = \begin{cases} 1, & \text{如果 } u \in V_i, v \in V_j, \text{ 且 } i = j, \\ 0, & \text{否则} \end{cases}$$



初始化颜色

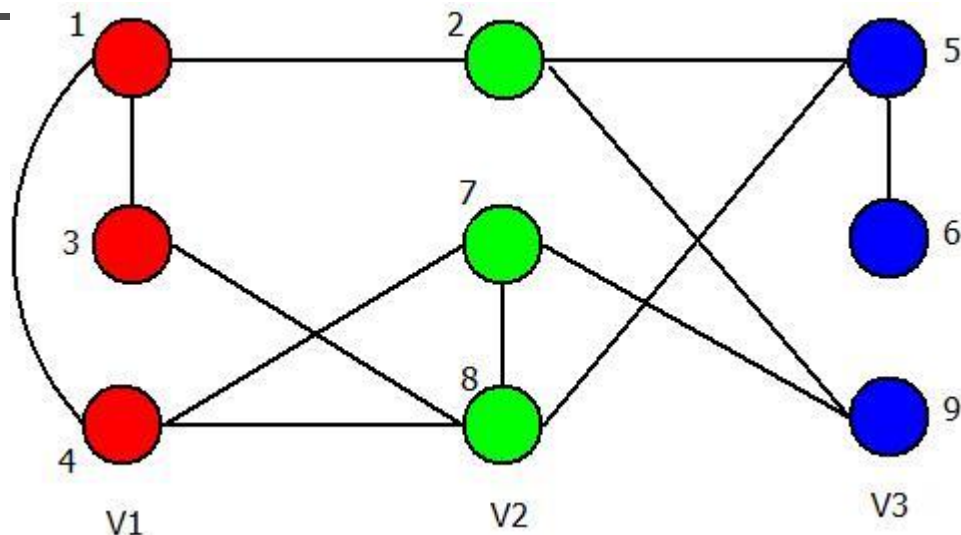
- 该算法的初始解是随机生成的，也就是说，对图中的每个节点随机分配一个从1到 k 的颜色。
- 其他初始构造算法：贪心构造算法，如DSATUR, RLF, DANGER等。
- 对于 k -着色问题，通过实验发现有效的局部搜索算法对于初始解不是很敏感。



邻域动作

- 给定 k -着色的邻域是通过将冲突节点 u 从原来的颜色类 V_i 移动到另一个颜色类 V_j (用 $\langle u, i, j \rangle$ 表示), 称为“**关键动作 (critical one-move)**”邻域。
- 节点是**冲突**的, 意味着它的相邻节点中至少有一个和它所染的颜色相同。
- 因此, 对于冲突数为 $f(S)$ 的 k -着色 S , 邻域大小的上界为 $O(f(S) \times k)$ 。

例:



$k=3$, 随机初始解

- 冲突数: $f = 4$
- 冲突对: $(1, 3), (1, 4), (7, 8), (5, 6)$
- 关键动作: 只考虑冲突节点1, 3, 4, 7, 8, 5, 6, 每个节点都可从当前颜色变到其它两种染色, 则总共 $7 \times 2 = 14$ 种动作。



邻域评估

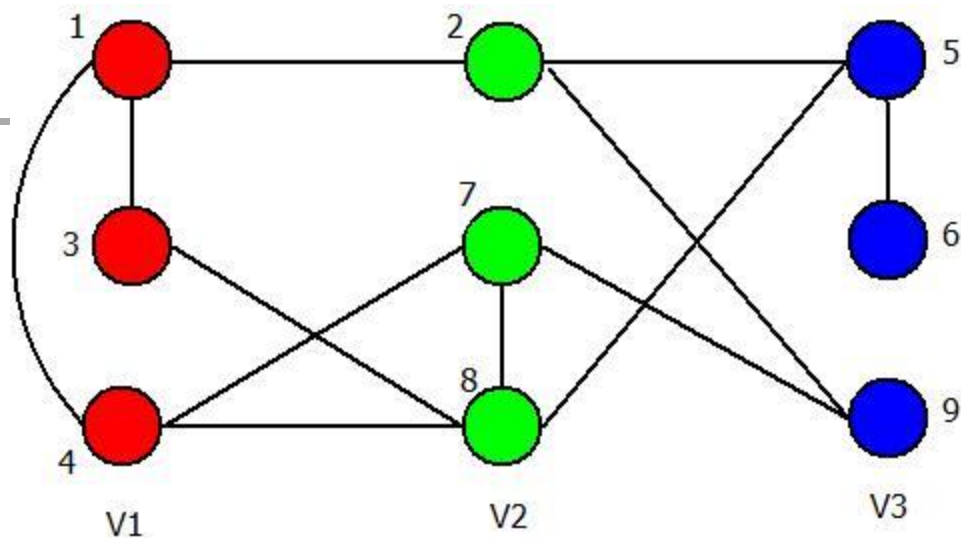
- 邻域评估: . . .

目的：找到最好的执行动作

- 采用了增量评估技术。
- 通过一种特殊的数据结构可以快速地计算出每一次移动对目标函数的影响。
- 每次执行动作时，只相应地更新受此动作影响的移动值。

邻接颜色表

Vertex	Red V_1	Green V_2	Blue V_3
1	<u>2</u>	1	0
2	1	<u>0</u>	1
3	<u>1</u>	1	0
4	<u>1</u>	2	0
5	0	2	<u>1</u>
6	0	0	<u>1</u>
7	1	<u>1</u>	1
8	2	<u>1</u>	1
9	0	2	<u>0</u>



• 如果节点 u 染了颜色 i ，矩阵 $M[u][i]$ ($N * k$) 用来评估节点 u 的邻居染颜色 i 的个数。

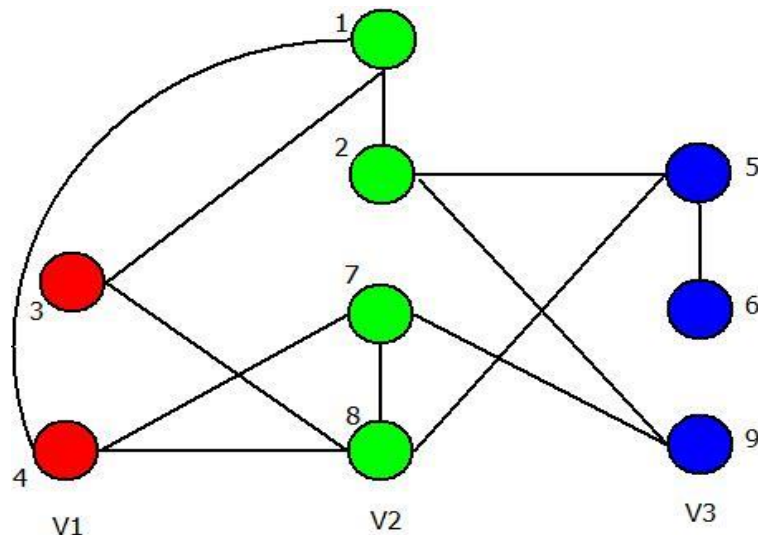
• 因此，一个动作 $\langle u, i, j \rangle$ 的增量值能够用下列式子快速计算：

$$\Delta(u, i, j) = M[u][j] - M[u][i]$$

更新颜色表

每次找冲突最大的节点

Vertex	Red V_1	Green V_2	Blue V_3
1	2	<u>1</u>	0
2	$1-1=0$	$0+1=1$	1
3	$1-1=0$	$1+1=2$	0
4	$1-1=0$	$2+1=3$	0
5	0	2	<u>1</u>
6	0	0	<u>1</u>
7	1	<u>1</u>	1
8	2	<u>1</u>	1
9	0	2	<u>0</u>



- 移动($1, v_1, v_2$):
 - 只有相邻的节点2、3和4受到影响，且仅有 v_1 和 v_2 列需要更新
 - 所有旧颜色(v_1)列减小1
 - 所有新颜色(v_2)列增加1

简单局部搜索

- 生成初始解 S ，计算 $f(S)$
- 初始邻接颜色表 M
- 当 存在改进动作 时
 - 构造 S 的邻居，标记为 $N(S)$
 - 计算所有关键动作的增量值 Δ
 - 用 Δ 找到最好的动作
 - 执行最好的动作： $f' = f + \Delta_{best}$
 - 更新邻接颜色表 M
- 结束

算法框架



■ 禁忌搜索-跳出局部最优

- 通常，我们禁忌的是解决方案的**属性**，而不是**解决方案本身**，因为禁忌解决方案本身代价太大。
- 对于禁忌表，一旦动作 $\langle u, i, j \rangle$ 被执行，节点 u 在之后的 tt （迭代步长）步迭代内将不能再次移回到颜色类 V_i 中。

■ 禁忌步长

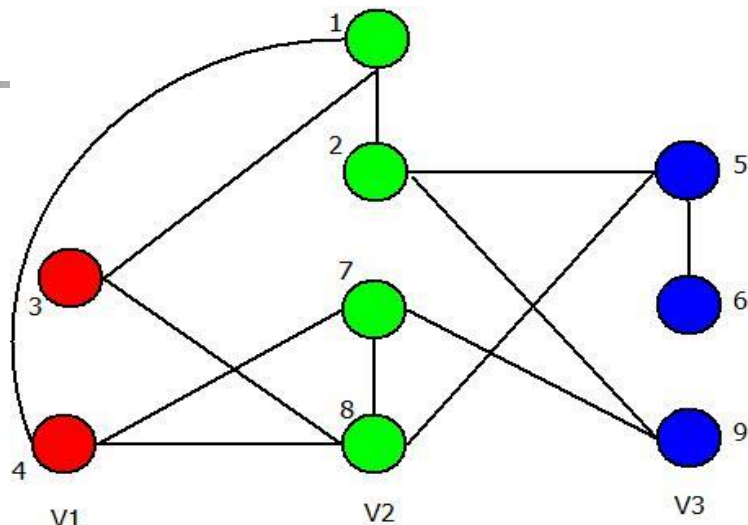
- 这里的禁忌步长 tt 是动态的

$$tt = f(S) + rand(10)$$

$rand(10)$ 是 $\{1, 2, \dots, 10\}$ 内的随机数。

禁忌表

Vertex	Red V1	Green V2	Blue V3
1	9	0	0
2	0	10	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0

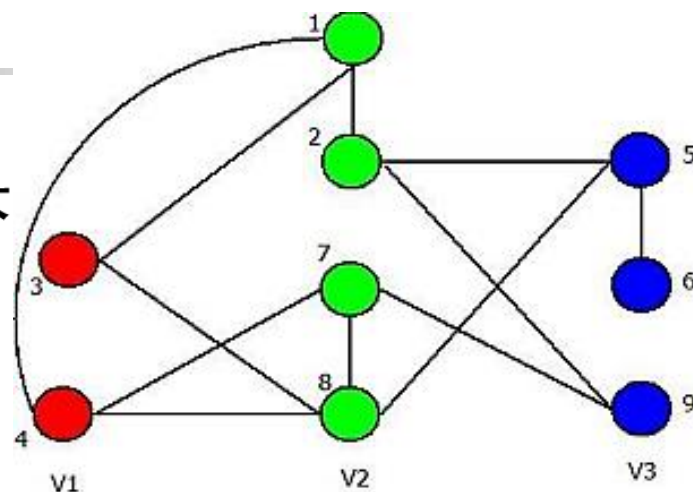


- 在搜索开始时，禁忌表初始化为零。
- 一旦执行动作 $\langle u, i, j \rangle$, $Table[u][i] = tt$
- 搜索过程中，每次迭代都需要对禁忌表中非零值-1
- 下次搜索时，通过检查是否 $Table[u][j] > 0$ 来判断动作 $\langle u, i, j \rangle$ 是否禁忌。


禁忌表的改进

- 上述 $Table[u][i]$ 记录的是**相对**禁忌长度。为什么不记录**绝对**禁忌长度？

Vertex	Red V1	Green V2	Blue V3
1	1+10	0	0
2	0	2+15	0
3	0	0	0
4	0	0	3+12
5	0	0	0
6	0	0	0
7	0	4+10	0
8	0	0	0
9	0	0	5+12



- 一旦执行了动作 $\langle u, i, j \rangle$,
 $Table[u][i] = tt + Iter$ (迭代次数)
- 在后面搜索中, 我们可以通过检查是否 $Table[u][j] > Iter$ 来判断动作 $\langle u, i, j \rangle$ 是否禁忌。
- 以上策略可以在 $O(1)$ 时间内更新禁忌表



禁忌搜索-整体框架

- 生成初始解 S , 计算 $f(s)$
- 初始化邻接颜色表 M .
- 当 停止条件不满足 时
 - 创建 S 的邻居, 定义为 $N(S)$
 - 计算当前邻域所有关键动作的 Δ 值
 - 用计算得到的 Δ 值寻找最好的禁忌和非禁忌动作
 - 如果特赦准则满足:
 - 执行最好的禁忌动作
 - 否则:
 - 执行最好的非禁忌动作
 - 更新 f 和邻接颜色表 M
- 结束

- 数据结构:

Sol[N], f, BestSol[N], Best_f, TabuTenure[N][K],
Adjacent_Color_Table[N][K](邻接颜色表)

- 子函数:

- Initialization():初始数据结构的值
- FindMove($u, v_i, v_j, delt$):寻找一个最好的禁忌或非禁忌动作.
- MakeMove($u, v_i, v_j, delt$):更新相关值.

- TabuSearch(){

- 初始化: u, v_i, v_j , 迭代次数 $iter=0$;
- Initialization();
- 当满足 $iter < MaxIter$ (最大迭代次数){
 - 寻找最好的邻域动作(FindMove($u, v_i, v_j, delt$));
 - 执行动作并更新相关值(MakeMove($u, v_i, v_j, delt$)));}}

- **FindMove**($u, v_i, v_j, delt$)
- { $i=1 \rightarrow n$
- 如果 $Adjacent_Color_Table[i][Sol[i]] > 0$ {
- $k=1 \rightarrow K$
- 如果 $k \neq Sol[i]$ {
- 计算动作的delt值 $\langle i, Sol[i], k \rangle$ {
- 如果 $iter < Table[i][k]$: 更新禁忌最好动作;
- 否则更新非禁忌最好动作;
- }
- }}
- 如果最好的禁忌动作满足特赦准则
- $\langle u, v_i, v_j, delt \rangle =$ 禁忌最好动作;
- 否则 $\langle u, v_i, v_j, delt \rangle =$ 非禁忌最好动作;
- }



Make Move

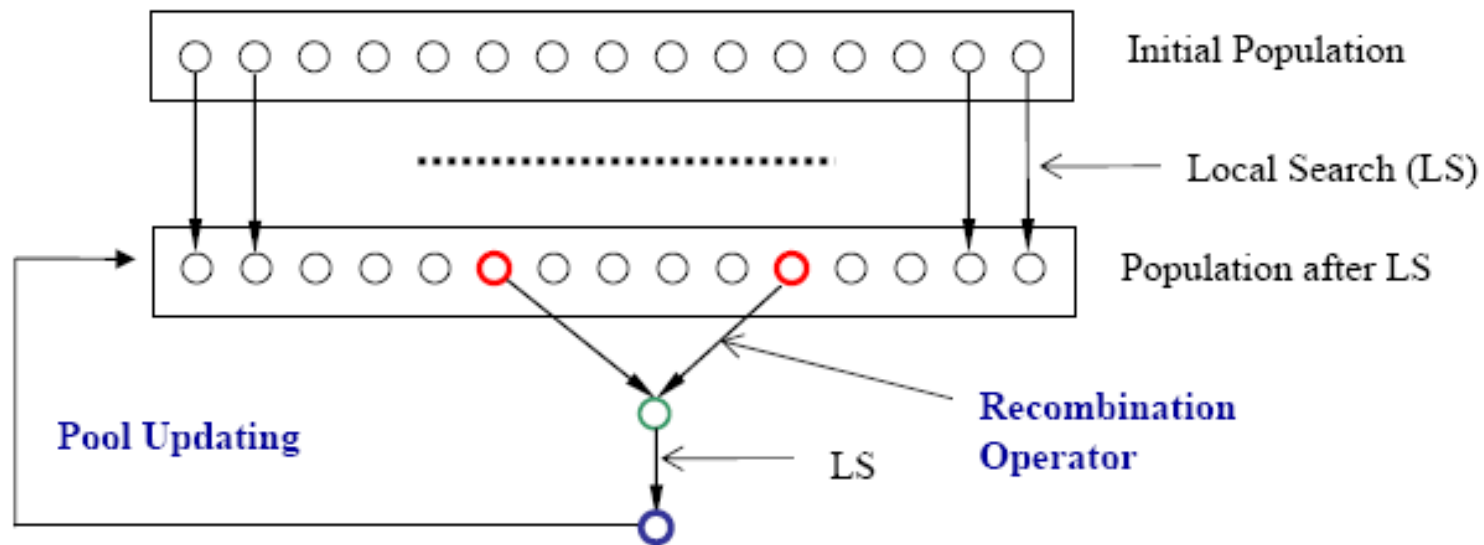
- $\text{MakeMove}(u, v_i, v_j, \text{delt})$
- {
- $\text{Sol}[u] = v_j ;$
- $f = f + \text{delt} ;$
- $\text{Table}[u][v_i] = \text{iter} + f + \text{rand}() \% 10 ;$
- 更新 `Adjacent_Color_Table`;
- }



提示

- **特赦准则：**如果一个邻域动作可以覆盖到目前为止找到的最佳解决方案，即使它处于禁忌状态，也会被接受。
- **原因：**禁忌表只存储了解的属性，而不会解决方案本身。
- **特赦准则——需要满足两个条件**
 1. 最好的禁忌动作好于历史最好解；
 2. 最好的禁忌动作好于当前邻域的非禁忌的最好动作。
- 对于禁忌和非禁忌动作，如果有多个最好的动作时，随机选择一个；

7.3 混合进化算法



Hybrid Evolutionary Algorithm (LS+EA)



The hybrid coloring algorithm

Data : *graph $G = (V, E)$, integer k*

Result : *the best configuration found*

begin

$P = \text{InitPopulation}(|P|)$

while *not Stop-Condition ()* **do**

$(s1, s2) = \text{ChooseParents}(P)$

$s = \text{Crossover}(s1, s2)$

$s = \text{LocalSearch}(s, L)$

$P = \text{UpdatePopulation}(P, s)$

end

Table 2. The crossover algorithm: an example.

parent $s_1 \rightarrow$	A B C	D E F G	H I J
parent $s_2 \rightarrow$	C D E G	A F I	B H J
offspring s			

$V_1 := \{D, E, F, G\}$
remove D, E, F and G

A B C		H I J
C	A I	B H J
D E F G		

parent s_1	A B C		H I J
parent $s_2 \rightarrow$	C	A I	B H J
offspring s	D E F G		

$V_2 := \{B, H, J\}$
remove B, H and J

A C		I
C	A I	
D E F G	B H J	

parent $s_1 \rightarrow$	A C		I
parent s_2	C	A I	
offspring s	D E F G	B H J	

$V_3 := \{A, C\}$
remove A and C

		I
	I	
D E F G	B H J	A C

交叉算子

- 一个合法的 k -着色是 k 个独立集的集合；
- 有了这个观点，如果能通过交叉算子尽可能地最大化独立集的大小，它将有助于将剩余的节点推到独立集中；
- 换句话说，更多的节点在 k 步内从父个体传递到后代，而只有较少的节点没有分配；
- 通过这种方式，获得的后代个体有更大的可能性成为合法的着色。

The GPX crossover algorithm

Data : configurations $s_1 = \{V_1^1, \dots, V_k^1\}$ and $s_2 = \{V_1^2, \dots, V_k^2\}$

Result : configuration $s = \{V_1, \dots, V_k\}$

begin

for $l(1 \leq l \leq k)$ **do**

 if l is odd, then $A := 1$, else $A := 2$

 choose i such that V_i^A has a maximum cardinality

$V_l := V_i^A$

 remove the vertices of V_l from s_1 and s_2

 Assign randomly the vertices of $V - (V_1 \cup \dots \cup V_k)$

end

交叉算子——伪代码

- 一个合法的 k -着色是 k 个独立集的集合；
- 有了这个观点，如果能通过交叉算子尽可能地最大化独立集的大小，它将有助于将剩余的节点推到独立集中；
- 换句话说，更多的节点在 k 步内从父个体传递到后代，而只有较少的节点没有分配；
- 通过这种方式，获得的后代个体有更大的可能性成为合法的着色。

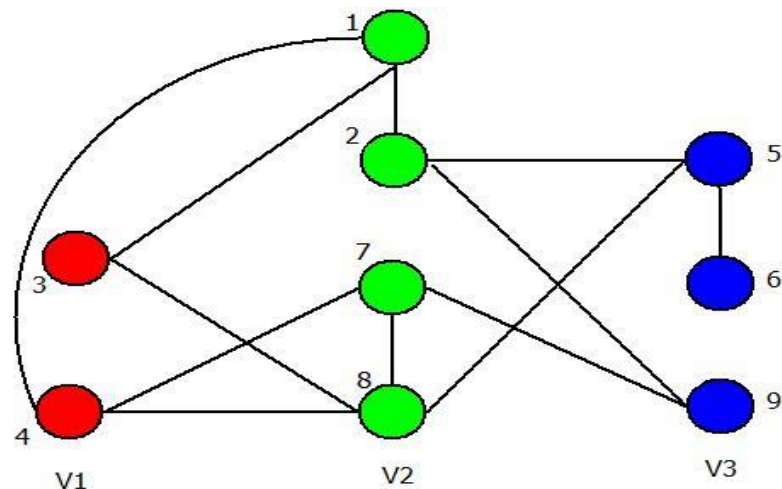
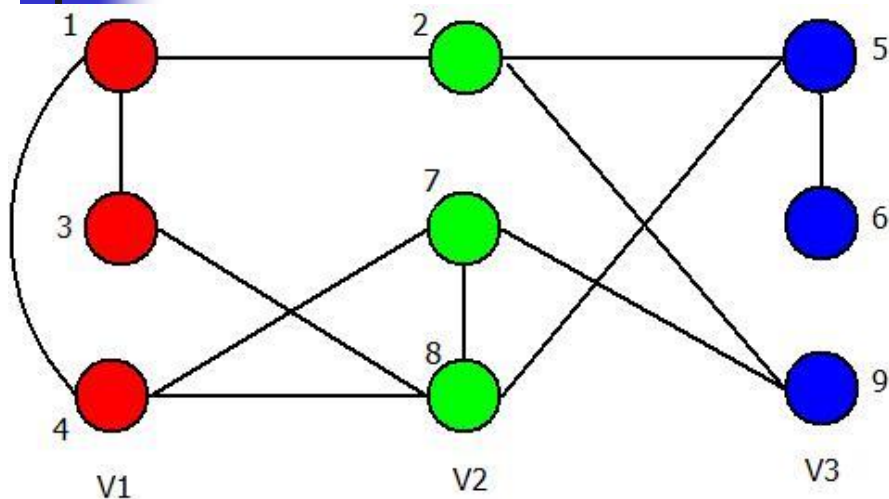


HEA算法框架

Algorithm 1 Pseudocode of the Hybrid Evolutionary Algorithm for k -Coloring

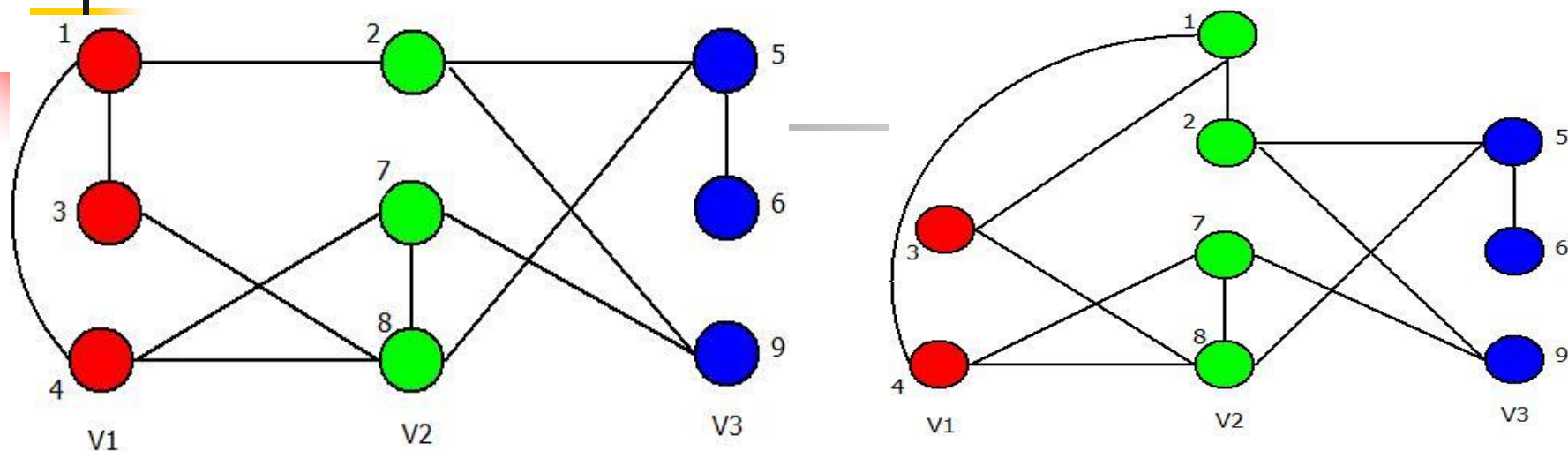
```
1: Input: Graph  $G$ 
2: Output: The best solution  $S^*$  found so far
3:  $\{S_1, \dots, S_p\} \leftarrow$  Initial Population
4: for  $i = \{1, \dots, p\}$  do
5:    $S_i \leftarrow$  Tabu Search( $S_i$ )
6: end for
7:  $S^* = \arg \min\{f(S_i), i = 1, \dots, p\}$ 
8: repeat
9:   Randomly choose two parent solutions  $\{S_{i1}, S_{i2}\}$ 
10:   $S_0 \leftarrow$  Crossover_Operator ( $S_{i1}, S_{i2}$ )
11:   $S_0 \leftarrow$  Tabu Search( $S_0$ )
12:  if  $f(S_0) < f(S^*)$  then
13:     $S^* = S_0$ 
14:  end if
15:   $\{S_1, \dots, S_p\} \leftarrow$  Pool Updating( $S_0, S_1, \dots, S_p$ )
16: until Stop condition met
```

7.4 图着色的高级玩法



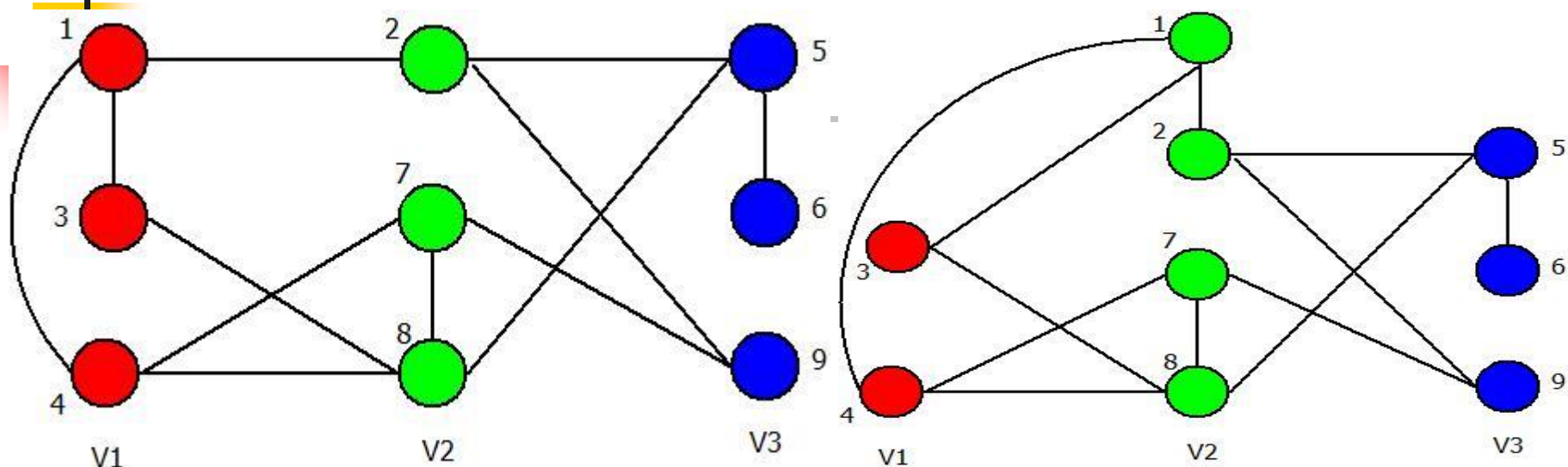
更新冲突节点:

- 动作($1, v_1, v_2$): 新的冲突节点是什么?
- 当节点2变为冲突节点时, 节点3和4变成了非冲突节点;
- 使用数据存储冲突节点, 每次移动后更新数组;
- 时间复杂度为 $O(1)$ 。



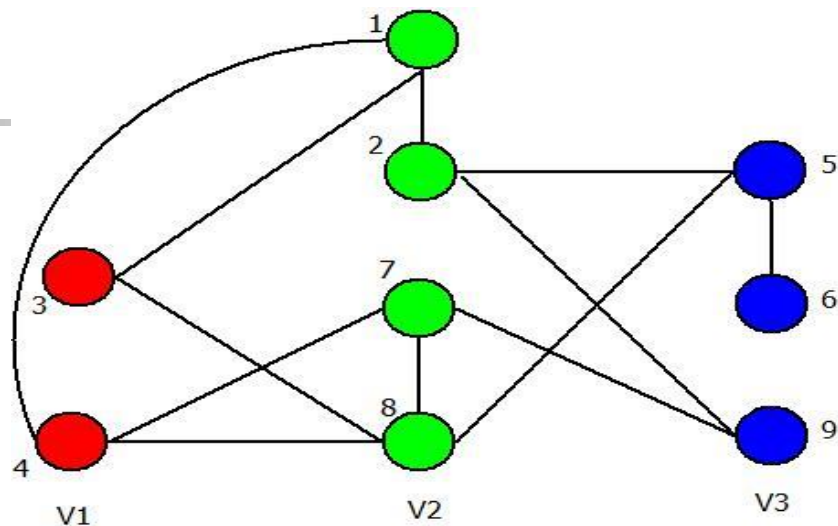
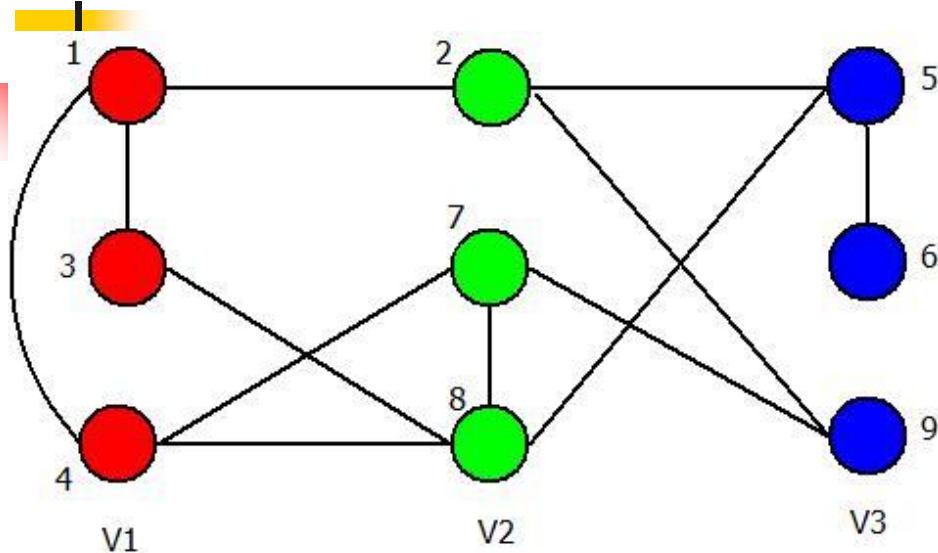
更新冲突节点：

- Num : 冲突节点数
- $Con[N]$: 冲突节点数组, 只有前 Num 有效
- $Pos[N]$: 记录任意一个节点在 $Conf$ 数组中的位置
- 加节点 V : $Conf[Num] = V$; $Pos[V] = Num + +$;
- 删节点 V : $Conf[Pos[V]] = Conf[Num]$;



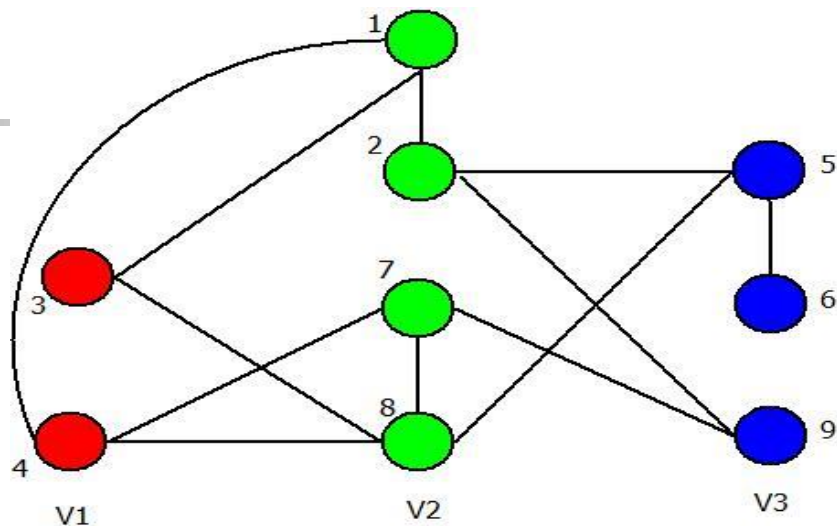
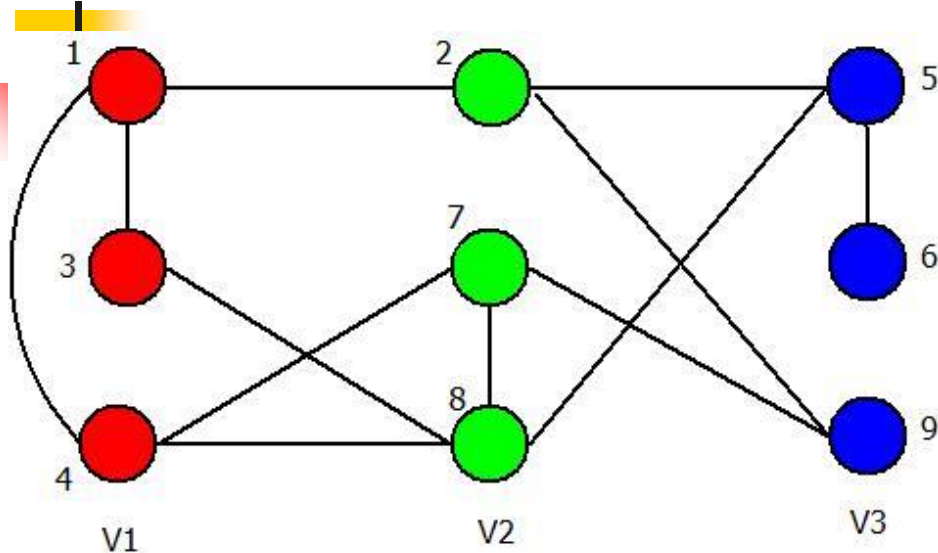
最优邻域动作如何快速找出来:

- 邻域大小: $m * (k - 1)$, m 为冲突节点数, k 为颜色数 (冲突节点维护技术: $N \rightarrow m$)
- 邻域动作时只选择禁忌最好的或者非禁忌最好的
- 上述动作后, 冲突节点减少2个, 增加1个, 原来14个邻域动作, 现在变为12个
- 哪些邻域动作发生了变化? 哪些动作的delt值发生了变化?



最优邻域动作如何快速找出来:

1. 减少的邻域动作: 节点3, 4到绿色、蓝色;
2. 增加的领域动作: 节点2到红色、蓝色;
3. **delt**值发生变化的领域动作: 无。
 - 如果节点7与节点1相连, 那么节点7移到红色的仇人数-1, 移到绿色+1。即节点1的邻居节点如果仍为冲突节点, 它到红色的仇人数-1, 移到绿色+1。
 - 既然大多数动作的**delt**值没有发生变化, 那么最优的**delt**值也有可能未发生变化



最优邻域动作如何快速找出来:

1. 减少的邻域动作直接删除
2. 增加的领域动作重新计算并加入
3. **delt**值发生变化的领域动作（即移动节点的邻居），分三种情况
 - a) 如果该节点在老颜色中，移到新颜色**delt**值+2，移到其它颜色**delt**值+1。
 - b) 如果该节点在新颜色中，移到老颜色**delt**值-2，移到其它颜色**delt**值-1。
 - c) 如果该节点在其它颜色中，移到老颜色**delt**值-1，移到新颜色**delt**值+1，其它不变。

最优邻域动作如何快速找出来:

- 桶排序技术（增、删、移动与之前的数据实现冲突节点维护一致）
- 辅助数据结构：Pos[N][k][k][2]

id	Delt值	邻域动作数	邻域动作（及delt值）
1	≤ -4	0	
2	-3	0	
3	-2	4
4	-1	1	...
5	0	10
6	1	12
7	2	12
8	3	25
9	≥ 4	30

最优邻域动作如何快速找出来：

- 每次从动作数非0的第一个桶中随机选一个动作
- 桶更新策略：
 1. 删掉的邻域动作要从桶中删掉；
 2. 增加的邻域动作要加到桶中；
 3. delt 值发生变化的动作要在桶中移动到对应的桶
- 该策略可以节约多少时间：
- 邻域动作大小： $m * (k - 1)$
- 更新的邻域动作数：(增、删冲突节点数) * $(k - 1)$ + 移动节点的邻居节点仍为冲突节点数 * 2

- 可以更懒吗？

- 惰性更新桶的策略：

1. 应该删除的邻域动作不用管
2. 新增加的邻域动作直接加到桶中；
3. delt 值发生变化的动作只在桶中加新的动作，旧动作不动

师徒进化算法

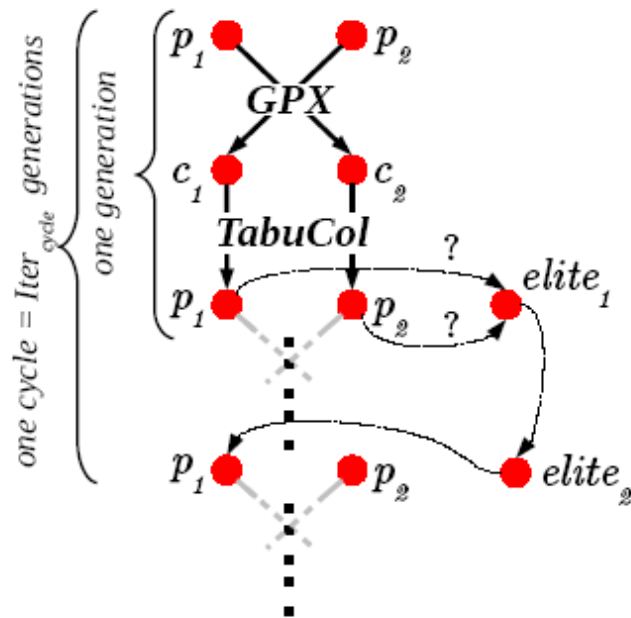


Fig. 2: Diagram of *HEAD*


Algorithm 2: *HEAD* - second version of *HEAD* with two extra elite solutions

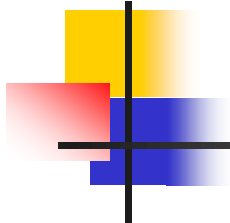
Input: k , the number of colors; $Iter_{TC}$, the number of *TabuCol* iterations; $Iter_{cycle} = 10$, the number of generations into one cycle.

Output: the best k -coloring found: *best*

```

1  $p_1, p_2, elite_1, elite_2, best \leftarrow \text{init}()$            /* initialize with
   random  $k$ -colorings */
2  $generation, cycle \leftarrow 0$ 
3 do
4    $c_1 \leftarrow GPX(p_1, p_2)$ 
5    $c_2 \leftarrow GPX(p_2, p_1)$ 
6    $p_1 \leftarrow TabuCol(c_1, Iter_{TC})$ 
7    $p_2 \leftarrow TabuCol(c_2, Iter_{TC})$ 
8    $elite_1 \leftarrow \text{saveBest}(p_1, p_2, elite_1)$  /* best  $k$ -coloring of
   the current cycle */
9    $best \leftarrow \text{saveBest}(elite_1, best)$ 
10  if  $generation \% Iter_{cycle} = 0$  then
11     $p_1 \leftarrow elite_2$            /* best  $k$ -coloring of the
   previous cycle */
12     $elite_2 \leftarrow elite_1$ 
13     $elite_1 \leftarrow \text{init}()$ 
14     $cycle ++$ 
15   $generation ++$ 
16 while  $nbConflicts(best) > 0$  and  $p_1 \neq p_2$ 
  
```

- 
- 局部搜索算法500.5能算到48种颜色吗?
 - 500.5能算够非常稳定地算到47种颜色吗?
 - 局部搜索算法500.5能算到47种颜色吗?
 - 禁忌算法的局限和新实现：格局检测
 - 节点是否可以回到老颜色应该取决于老颜色中它的“格局”是否发生了变化
 - 仅“格局检测”是不够的，还需要加权和平滑技术的配合使用



Thank You!