Lecture12_Local Search

1.introduction

- solve problems approximately
- aims at a local optimum

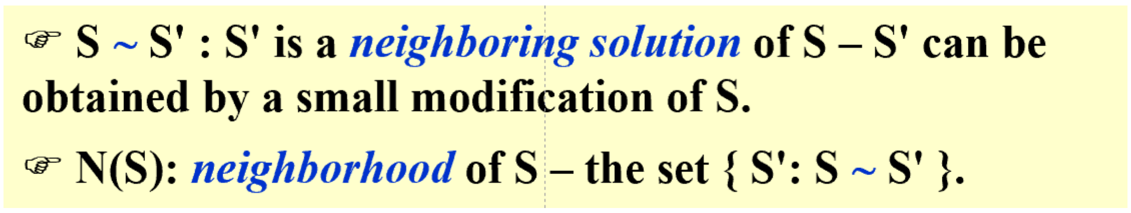
local:

- Define neighborhoods in the feasible set
- A local optimum is a best solution in a neighborhood

search:

- Start with a feasible solution and search a better one within the neighborhood
- A local optimum is achieved if no improvement is possible

neighbor relation:



☞ **S ~ S' : S' is a *neighboring solution* of S – S' can be obtained by a small modification of S.**

☞ **N(S): *neighborhood* of S – the set { S': S ~ S' }.**

```
SolutionType Gradient_descent()//梯度下降算法
{   Start from a feasible solution S □ FS ;
    MinCost = cost(S);
    while (1) {
        S’ = Search( N(S) ); /* find the best S’ in N(S) */
        CurrentCost = cost(S’);
        if ( CurrentCost < MinCost ) {
            MinCost = CurrentCost;    S = S’;
        }
        else  break;
    }
    return S;
}
```

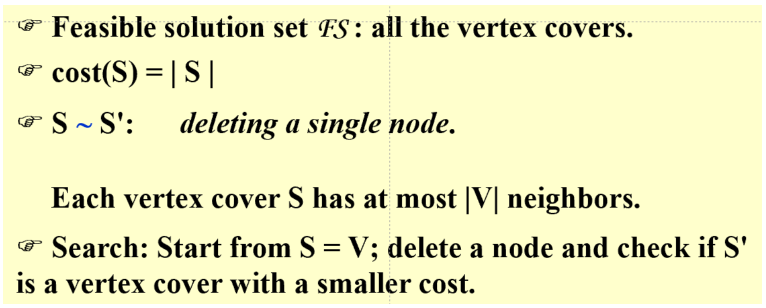
2.vertex cover

（decision version:最优解|V|，这里设置一个K，K>|V|，K=|V'|）

optimization version:

Vertex cover problem: Given an undirected graph G = (V, E). Find a minimum subset S of V such that for each edge (u, v) in E, either u or v is in S.

定义vertex 集合； 定义cost(); 定义S'（局部最优）；


☞ Feasible solution set $\mathcal{FS}$: all the vertex covers.

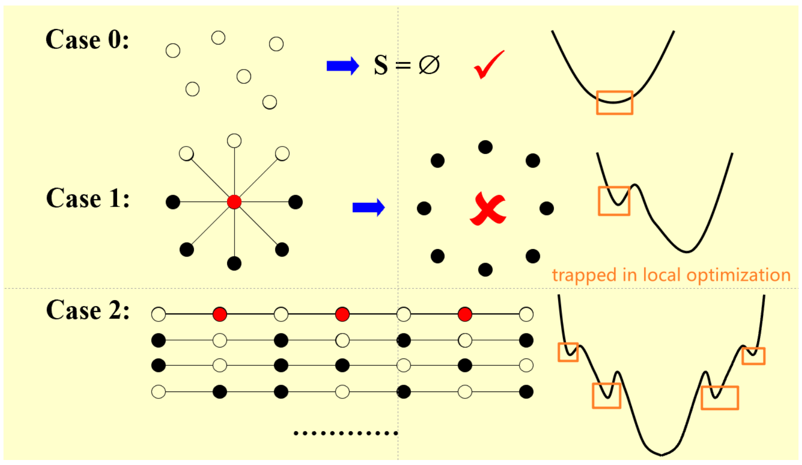
☞ cost(S) = | S |

☞ S ~ S': *deleting a single node.*

Each vertex cover S has at most |V| neighbors.

☞ Search: Start from S = V; delete a node and check if S' is a vertex cover with a smaller cost.

存在一些不work的情况



try to improve...

The Metropolis Algorithm

```
SolutionType Metropolis()
{   Define constants k and T;//T:temperature
    Start from a feasible solution S ∈ FS ;
    MinCost = cost(S);
    while (1) {
        S’ = Randomly chosen from N(S);
        CurrentCost = cost(S’);
        if ( CurrentCost < MinCost ) {
            MinCost = CurrentCost;    S = S’;
        }
        else {
            With a probability  e^(-Δcost/(kT)), let S = S’;
            else  break;
        }
    }
    return S;
}
```

Simulated Annealing(模拟退火)

The material is cooled very gradually from a high temperature, allowing it enough time to reach equilibrium（平衡） at a succession of intermediate lower temperatures.（相对较低温度）

Cooling schedule: T = { T1 , T2 , ... } 降温

3.Hopfield Neural Networks

input:

- Graph G = (V, E) with integer edge weights w (positive or negative).
- If we < 0, where e = (u, v), then u and v want to have the same state;if we > 0 then u and v want different states.
- The absolute value |we| indicates the strength of this requirement.

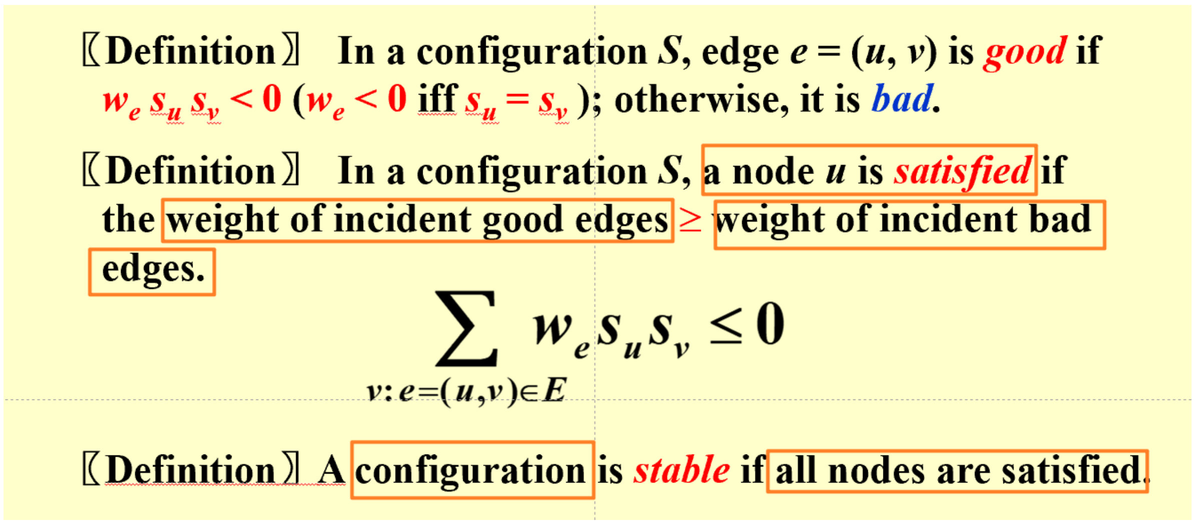
output:

A configuration(配置) S of the network – an assignment of the state s u to each node u

(There may be no configuration that respects the requirements imposed by all the edges.) 比如每一条边的weight 都是positive

so we need to find a configuration that is sufficiently good~

sufficiently good:



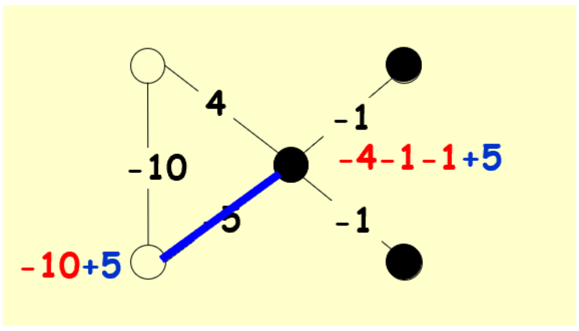
〖Definition〗 In a configuration S, edge $e = (u, v)$ is *good* if $w_e\, s_u\, s_v < 0$ ($w_e < 0$ iff $s_u = s_v$); otherwise, it is *bad*.

〖Definition〗 In a configuration S, a node u is *satisfied* if the weight of incident good edges $\geq$ weight of incident bad edges.

$$\sum_{v:e=(u,v)\in E} w_e s_u s_v \leq 0$$

〖Definition〗 A configuration is *stable* if all nodes are satisfied.

eg: satisfied

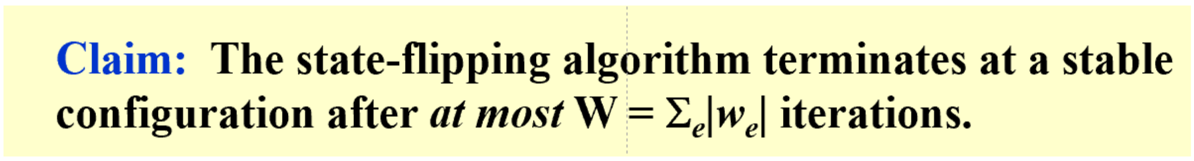


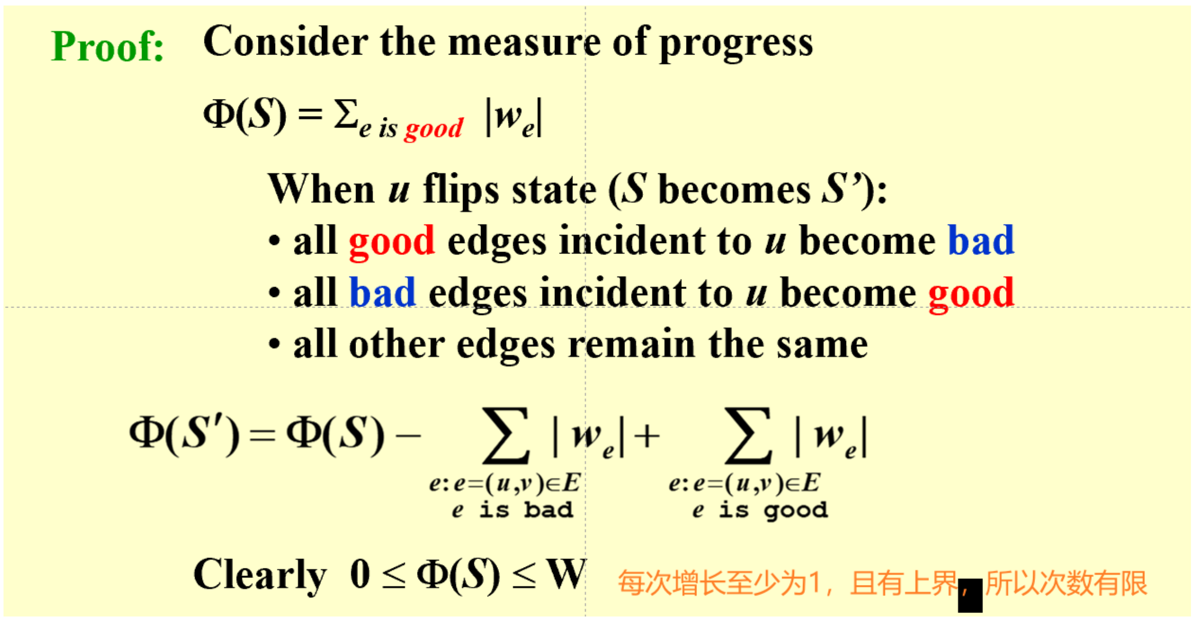
Does a Hopfield network always have a stable configuration, and if so, how can we find one?

state-flipping algorithm

```
ConfigType State_flipping()
{
    //初始化所有节点一个颜色
    Start from an arbitrary configuration S;
    while ( ! IsStable(S) ) {
        u = GetUnsatisfied(S);
        su = - su;
    }
    return S;
}
```

will it alwalys terminate? yes!


Claim: The state-flipping algorithm terminates at a stable configuration after *at most* $W = \Sigma_e|w_e|$ iterations.


Proof: Consider the measure of progress

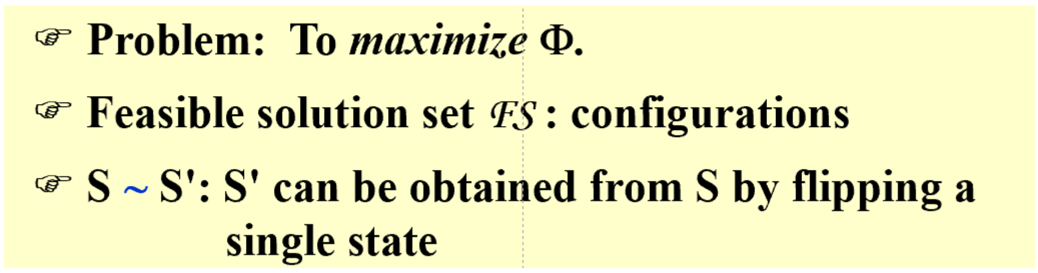
$\Phi(S) = \Sigma_{e \text{ is good}} |w_e|$

When u flips state (S becomes S'):
- all good edges incident to u become bad
- all bad edges incident to u become good
- all other edges remain the same

$$\Phi(S') = \Phi(S) - \sum_{\substack{e:e=(u,v)\in E \\ e \text{ is bad}}} |w_e| + \sum_{\substack{e:e=(u,v)\in E \\ e \text{ is good}}} |w_e|$$

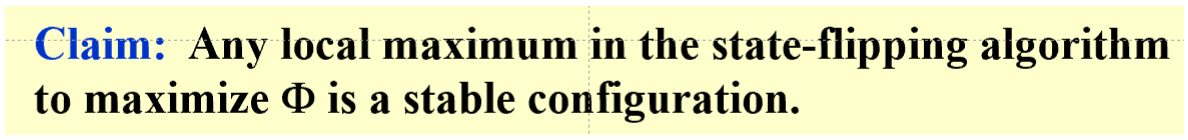
Clearly $0 \le \Phi(S) \le W$ 每次增长至少为1，且有上界，所以次数有限

related to local search


☞ Problem: To *maximize* Φ.

☞ Feasible solution set $\mathcal{FS}$: configurations

☞ S ~ S': S' can be obtained from S by flipping a single state

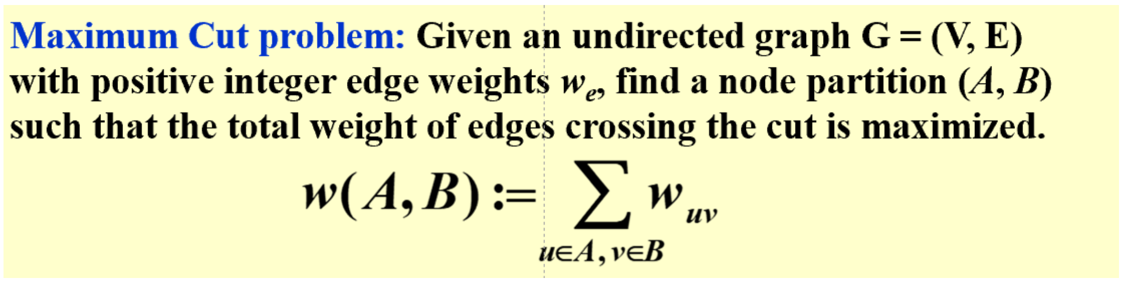

Claim: Any local maximum in the state-flipping algorithm to maximize Φ is a stable configuration.

Is it a polynomial time algorithm? no O(W) ~ O (2^n)

Still an open question: to find an algorithm that constructs stable states in time **polynomial in n and logW** (rather than n and W), or in a number of primitive arithmetic operations that is **polynomial in n alone**, independent of the value of W.

4.The Maximum Cut Problem.

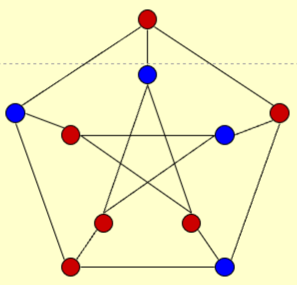
definition:



Maximum Cut problem: Given an undirected graph G = (V, E) with positive integer edge weights w_e, find a node partition (A, B) such that the total weight of edges crossing the cut is maximized.

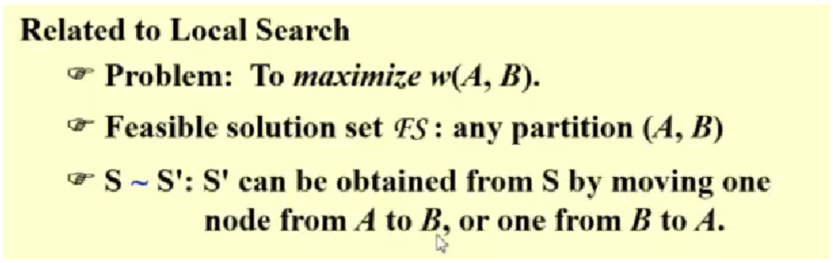
$$w(A,B) := \sum_{u\in A,\, v\in B} w_{uv}$$

eg:



- **Toy application**
  - *n* activities, *m* people.
  - Each person wants to participate in two of the activities.
  - Schedule each activity in the morning or afternoon to maximize number of people that can enjoy both activities.
- **Real applications** Circuit layout, statistical physics

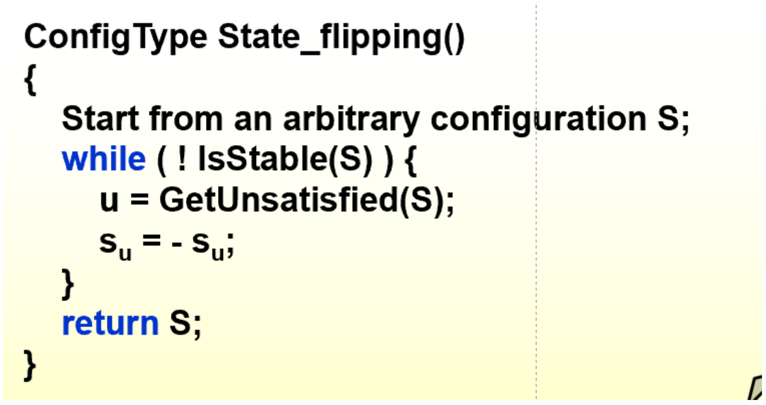
related to local search



Related to Local Search

- Problem: To *maximize* $w(A, B)$.
- Feasible solution set $\mathcal{FS}$: any partition (A, B)
- S ~ S': S' can be obtained from S by moving one node from A to B, or one from B to A.

可以用Hopfield Neural Networks 解决（maximizes 所有good edge)

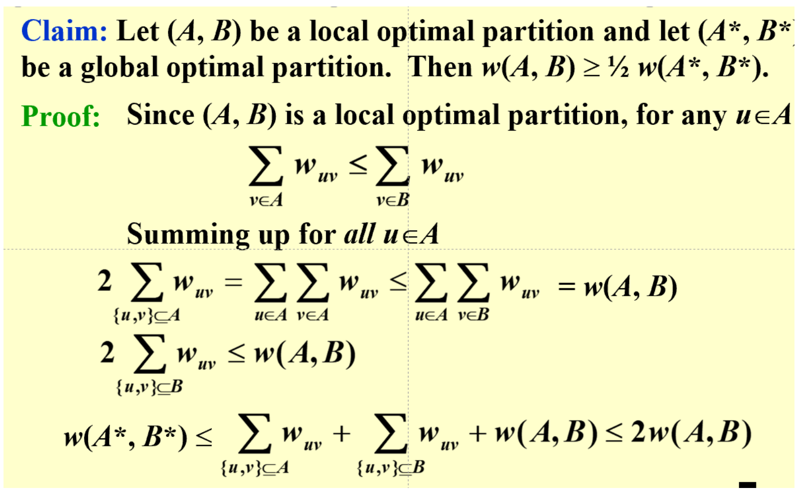


```
ConfigType State_flipping()
{
    Start from an arbitrary configuration S;
    while ( ! IsStable(S) ) {
        u = GetUnsatisfied(S);
        $s_u$ = - $s_u$;
    }
    return S;
}
```

问题是：1）时间可能不是polynomial? 2）how good is this local optimum? 3）try a better local?

4.1 how good is this local optimum?

局部最优至少是整体最优的一半



Claim: Let (A, B) be a local optimal partition and let $(A^*, B^*$
be a global optimal partition. Then $w(A, B) \geq ½\, w(A^*, B^*)$.

Proof: Since (A, B) is a local optimal partition, for any $u \in A$

$$\sum_{v \in A} w_{uv} \leq \sum_{v \in B} w_{uv}$$

Summing up for *all* $u \in A$

$$2 \sum_{\{u,v\} \subseteq A} w_{uv} = \sum_{u \in A} \sum_{v \in A} w_{uv} \leq \sum_{u \in A} \sum_{v \in B} w_{uv} = w(A, B)$$

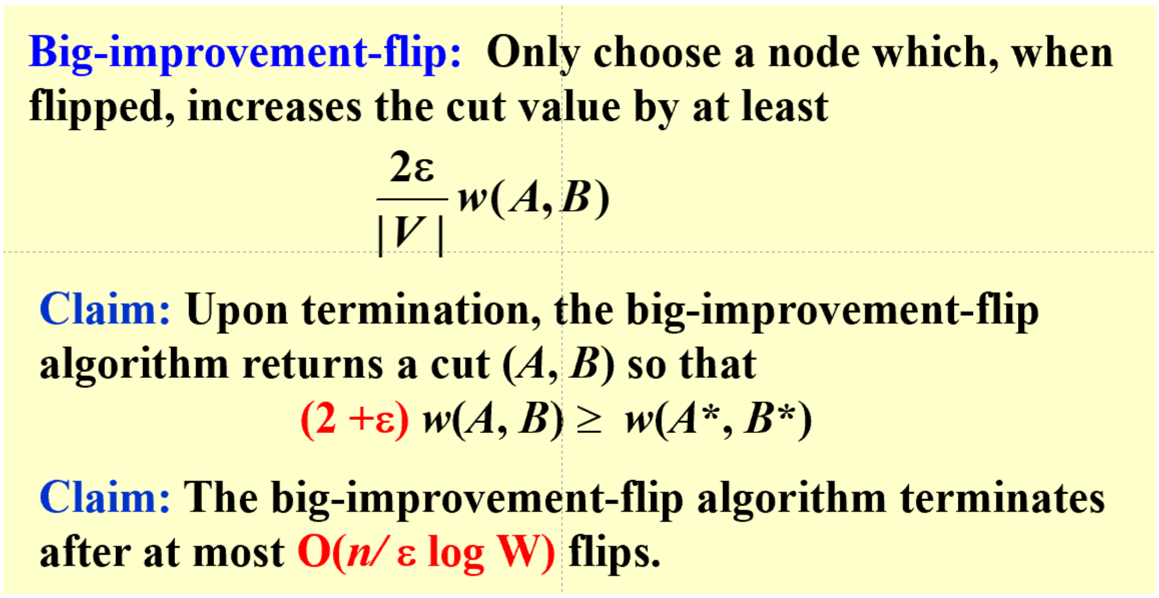
$$2 \sum_{\{u,v\} \subseteq B} w_{uv} \leq w(A, B)$$

$$w(A^*, B^*) \leq \sum_{\{u,v\} \subseteq A} w_{uv} + \sum_{\{u,v\} \subseteq B} w_{uv} + w(A, B) \leq 2w(A, B)$$

4.2 时间可能不是polynomial?

big-improvement-flip

stop the algorithm when there are no "big enough" improvements.



Big-improvement-flip: Only choose a node which, when flipped, increases the cut value by at least

$$\frac{2\varepsilon}{|V|} w(A, B)$$

Claim: Upon termination, the big-improvement-flip algorithm returns a cut (A, B) so that

$$(2 + \varepsilon)\, w(A, B) \geq w(A^*, B^*)$$

Claim: The big-improvement-flip algorithm terminates after at most $O(n/\varepsilon \log W)$ flips.

根据时间简单描述证明:

1.每次flip至少增加(1+epsilon/n)倍，其实是（1+2*epsilon/n)倍

\2. n/epsilon次flip之后，总增长至少是2倍。利用(1+1/x)^x >= 2, 如果x>=1

\3. 总量不超过W，而cut翻倍的次数不能超过logW

4.3 try a better local?

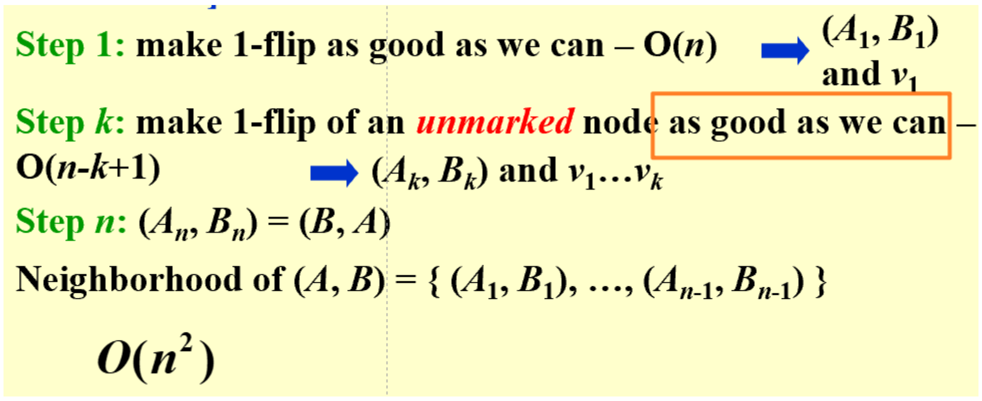
The neighborhood of a solution should be rich enough that we do not tend to get stuck in bad local optima;

but the neighborhood of a solution should not be too large, since we want to be able to efficiently search the set of neighbors for possible local moves.

single-flip -> k-flip   θ (n^k) for searching in neighbors

K-L heuristic generate a rich neighborhood~

最多n steps



Step 1: make 1-flip as good as we can – O(n) $\Rightarrow$ (A_1, B_1) and v_1

Step k: make 1-flip of an ***unmarked*** node $\boxed{\text{as good as we can}}$ – O(n-k+1) $\Rightarrow$ (A_k, B_k) and $v_1 \dots v_k$

Step n: (A_n, B_n) = (B, A)

Neighborhood of (A, B) = { (A_1, B_1), …, (A_{n-1}, B_{n-1}) }

$$O(n^2)$$