Tabu Search

Lecture # 4

Esmaeil Nourani

Horse with wings !!

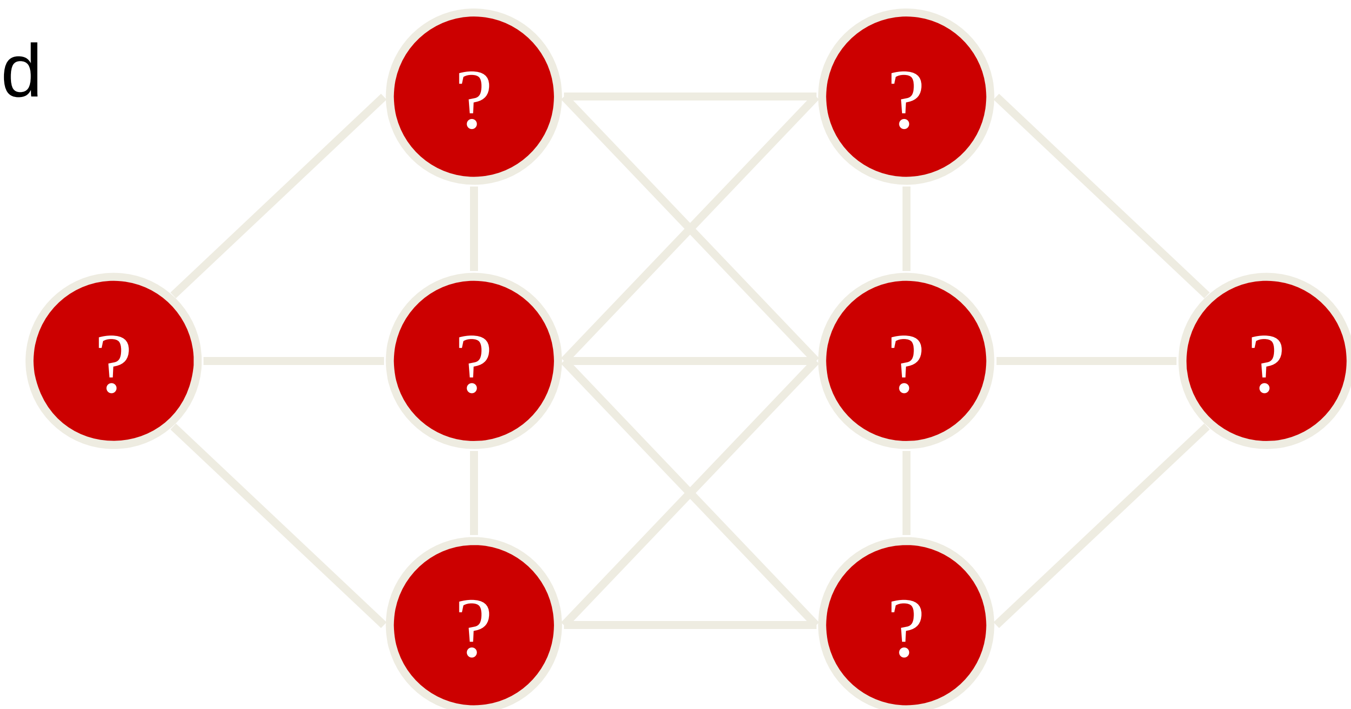
Some possibilities of escaping local optima within a single run of an algorithm :

- An additional parameter (called temperature) that changes the probability of moving from one point of the search space to another. (simulated Annealing)
- A memory, which forces the algorithm to explore new areas of the search space.(Tabu Search)

You have
5 minutes!

Crystal Maze

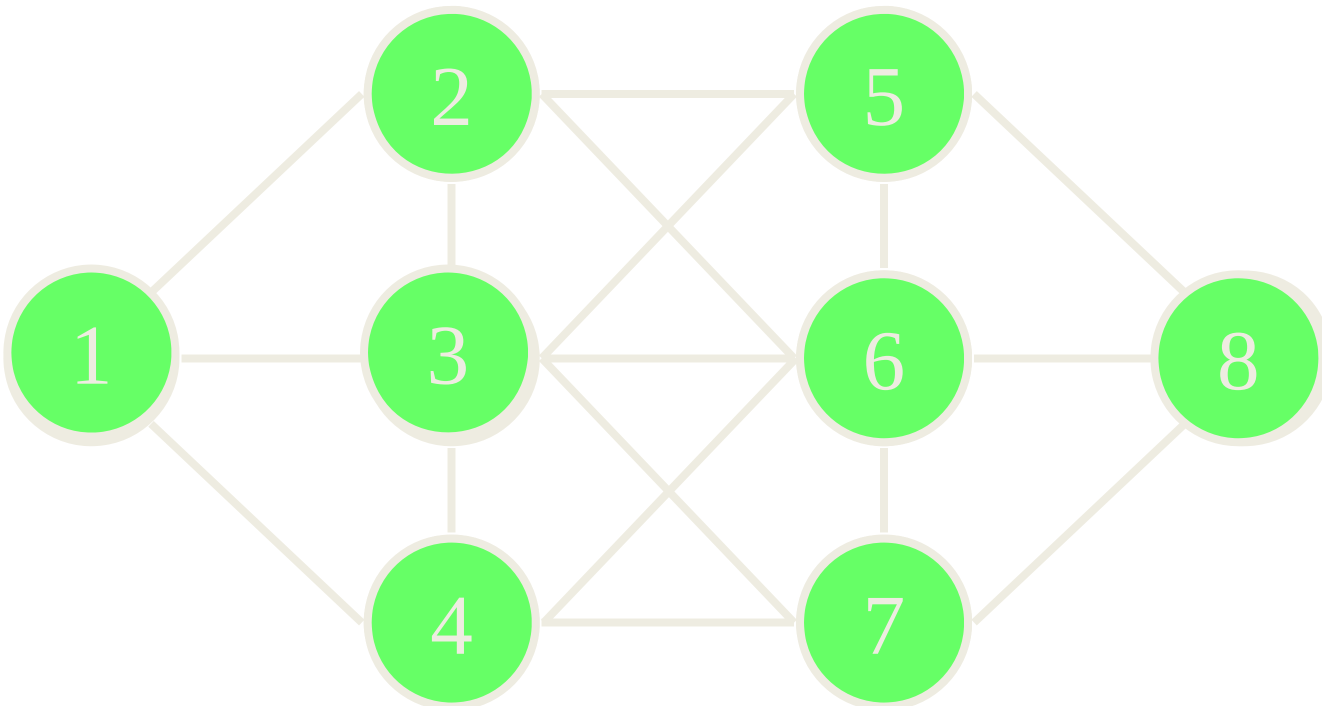
- Place the numbers 1 through 8 in the nodes such that:
  - Each number appears exactly once
  - No connected nodes have consecutive numbers



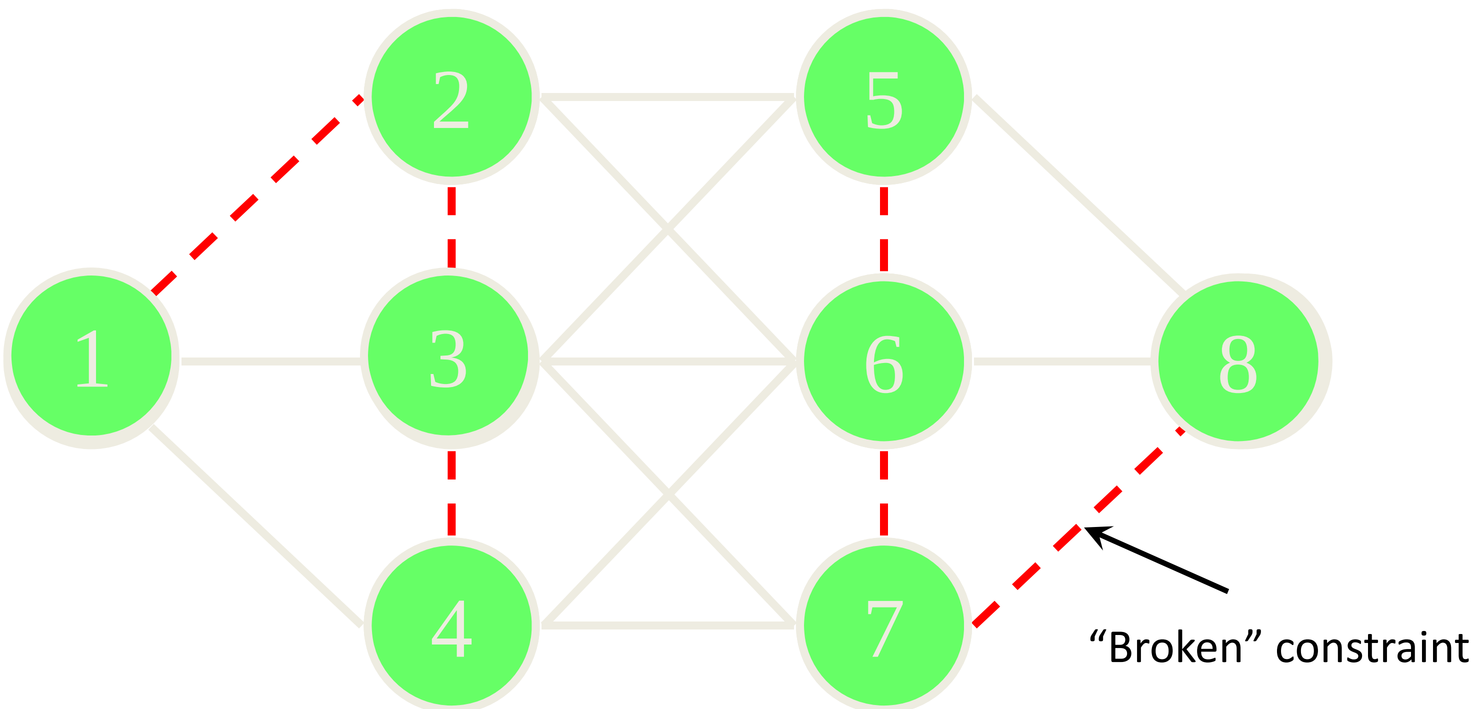
Local Search Idea

- Randomly assign values (even if the constraints are "broken")
  - Initial state will probably be infeasible
- Make "moves" to try to move toward a solution

Random Initial Solution



Random Initial Solution



Cost = # of broken constraints

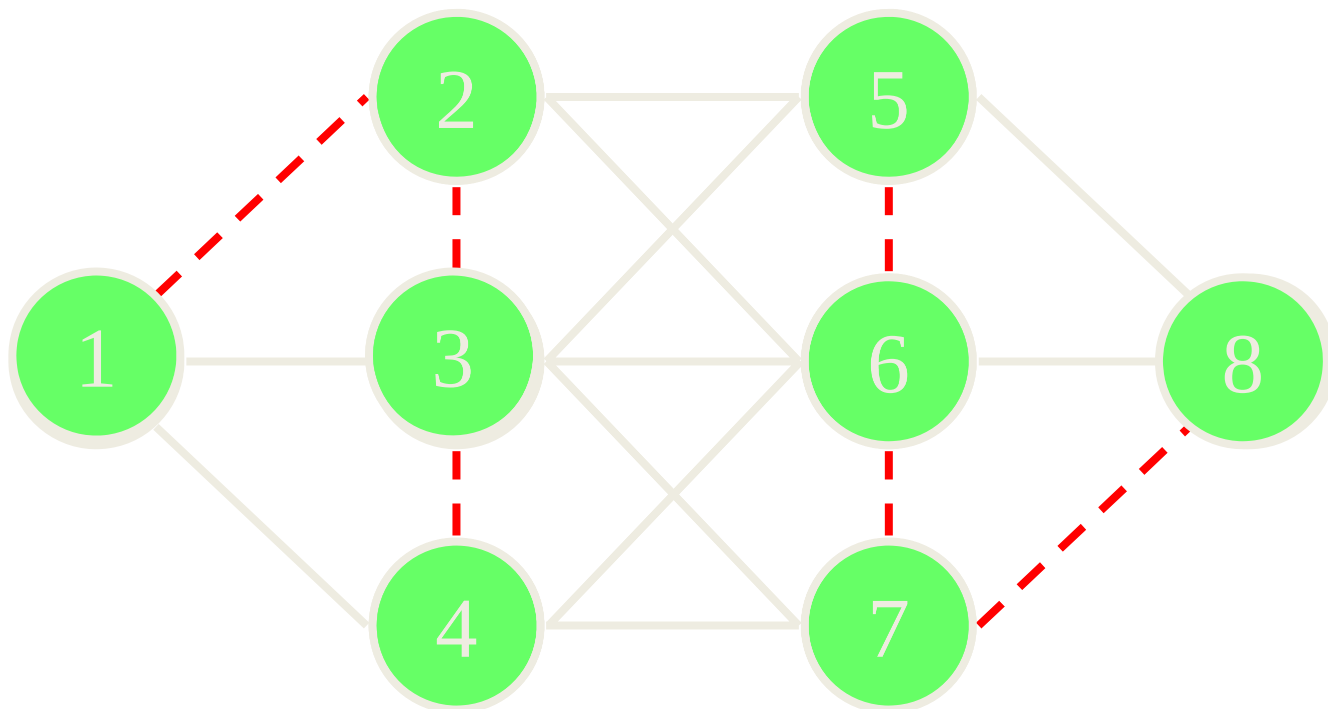
What Should We Do Now?

- Move:
  - Swap two numbers
- Which two numbers?
  - Randomly pick a pair
  - The pair that will lead to the biggest decrease in cost
    - Cost: number of broken constraints

What Should We Do Now?

- Move:
  - Swap two numbers
- Which two numbers?
  - Randomly pick a pair
  - The pair that will lead to the biggest decrease in cost
    - Cost: number of broken constraints

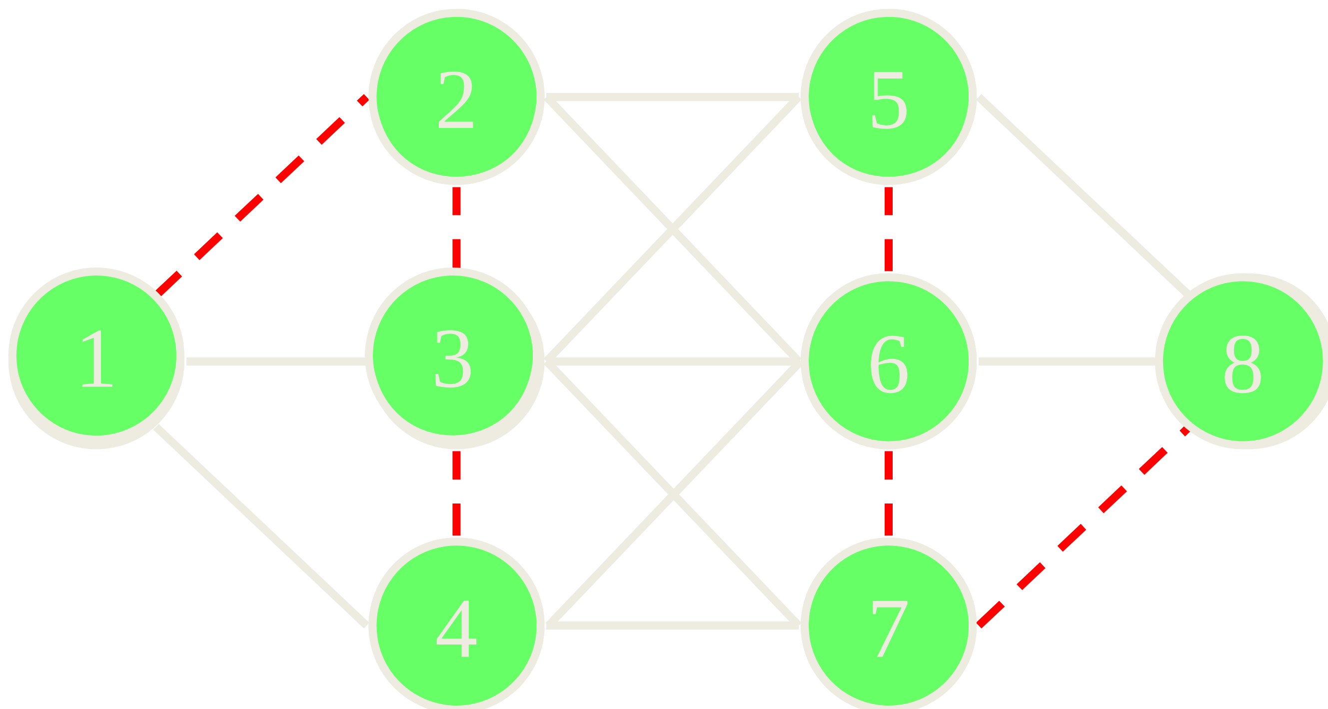
Random Initial Solution



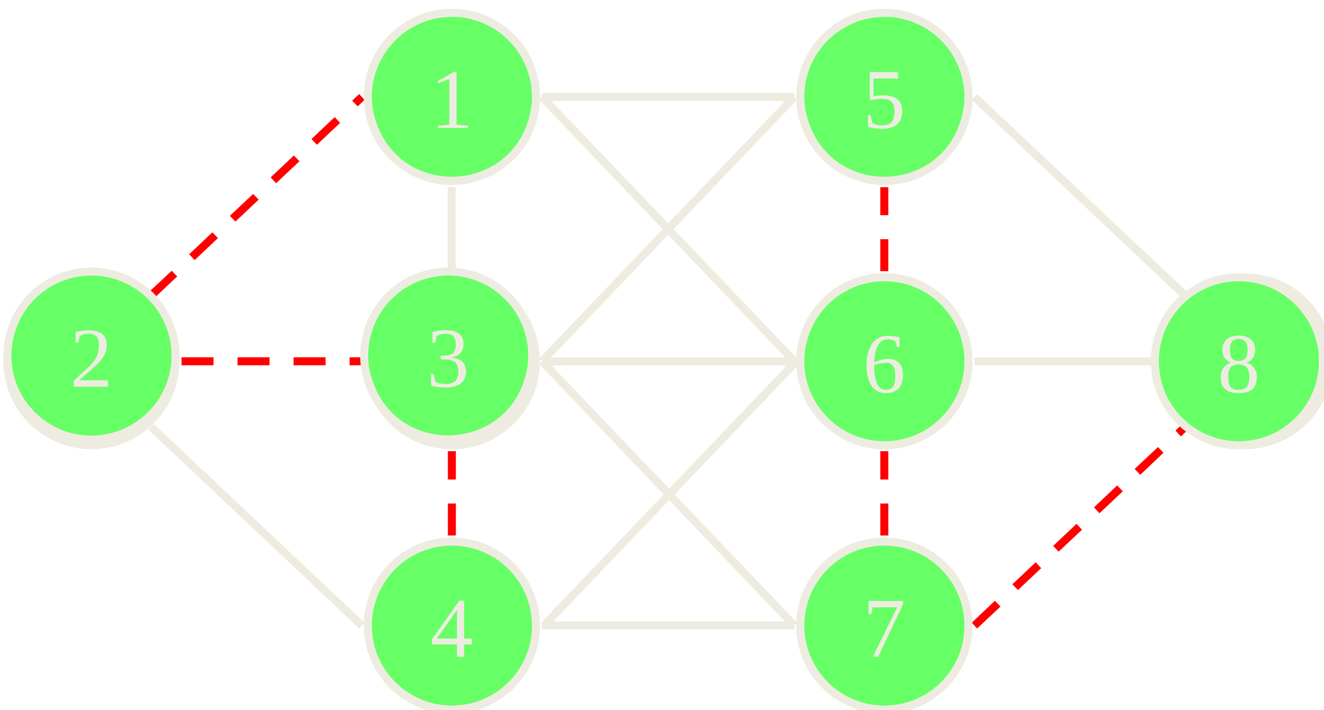
Cost Difference Table

	1	2	3	4	5	6	7	8
1	0							
2		0						
3			0					
4				0				
5					0			
6						0		
7							0	
8								0

Random Initial Solution



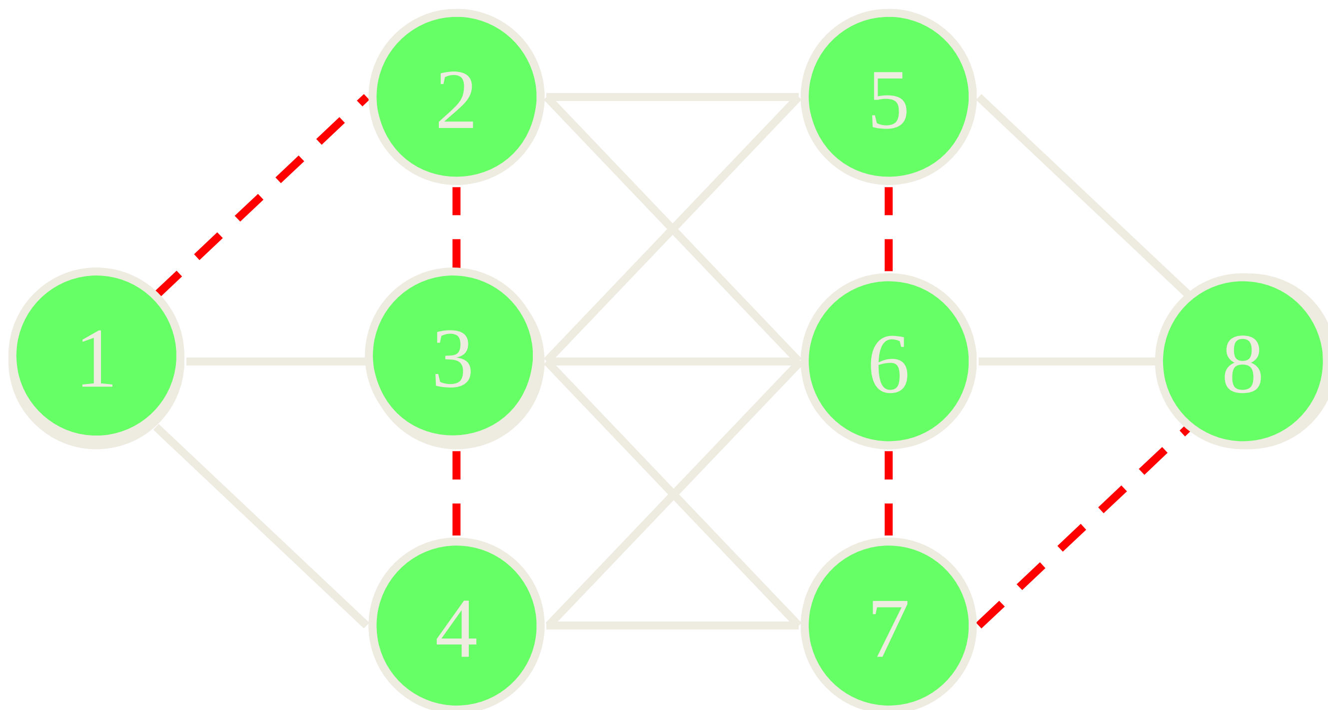
Swap 1 & 2



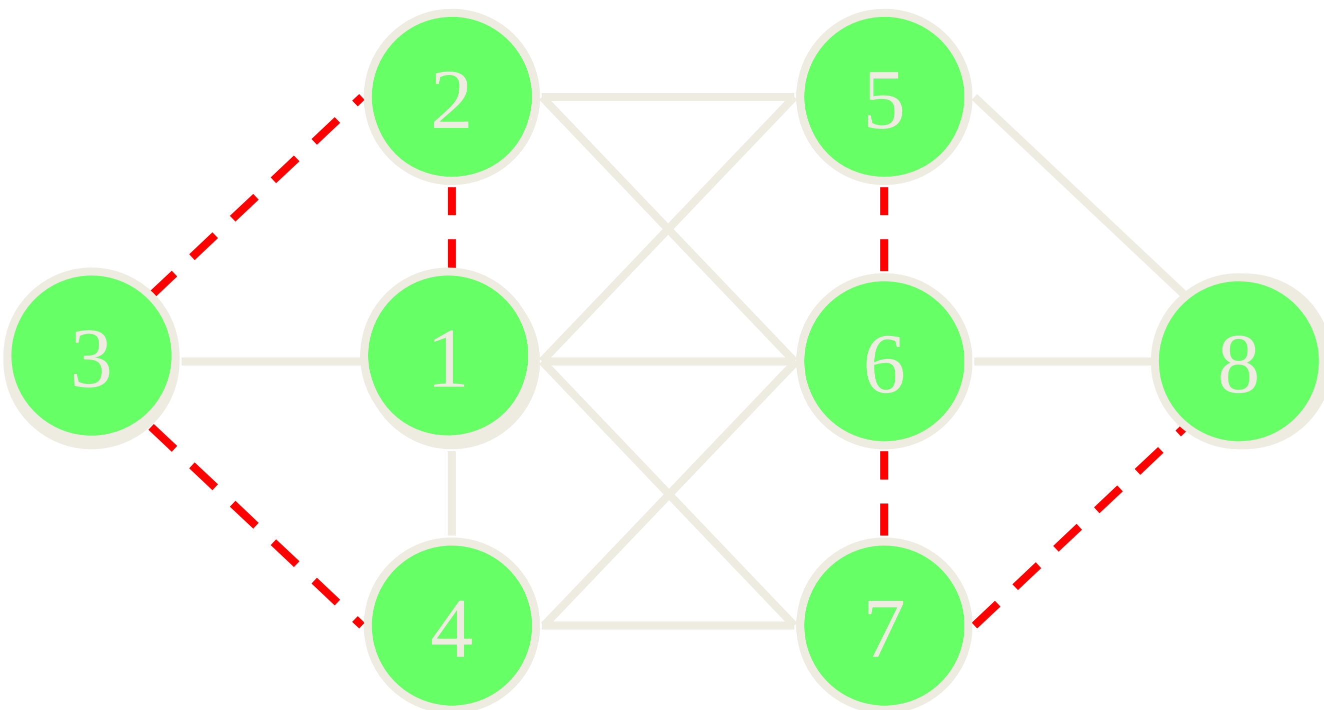
Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	0						
2		0						
3			0					
4				0				
5					0			
6						0		
7							0	
8								0

Random Initial Solution



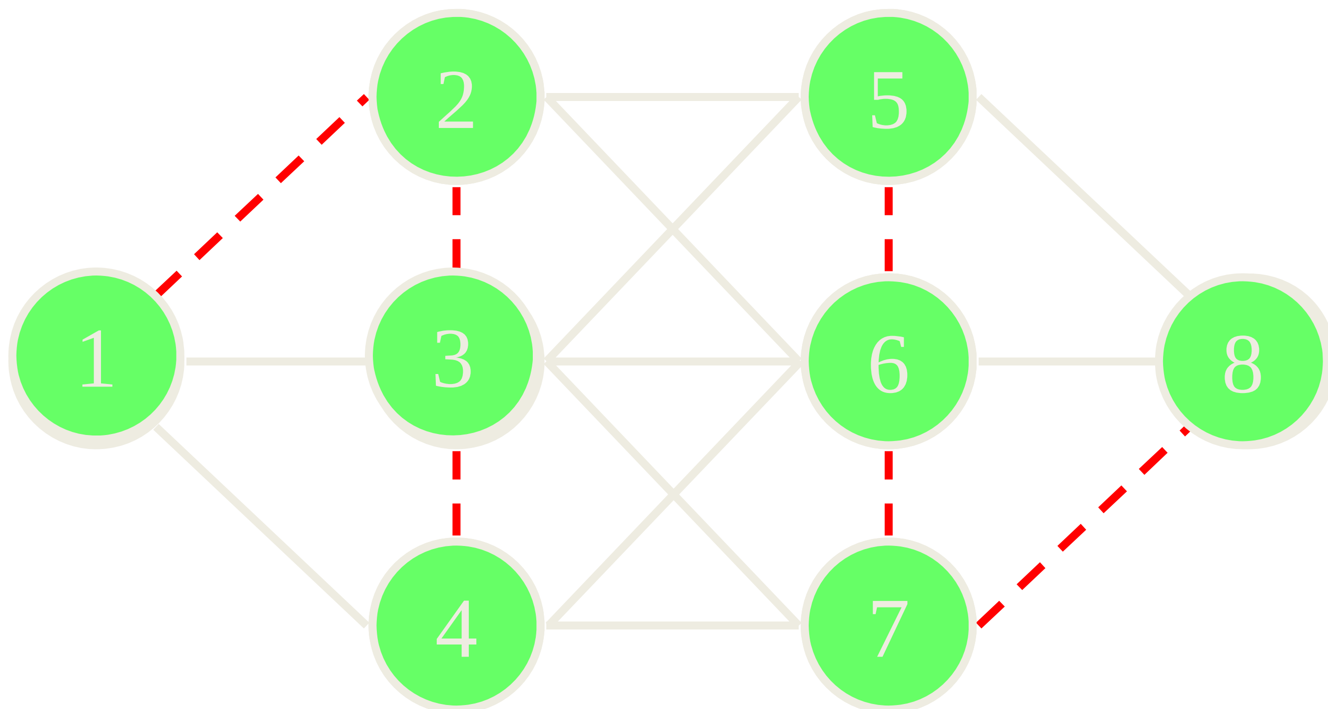
Swap 1 & 3



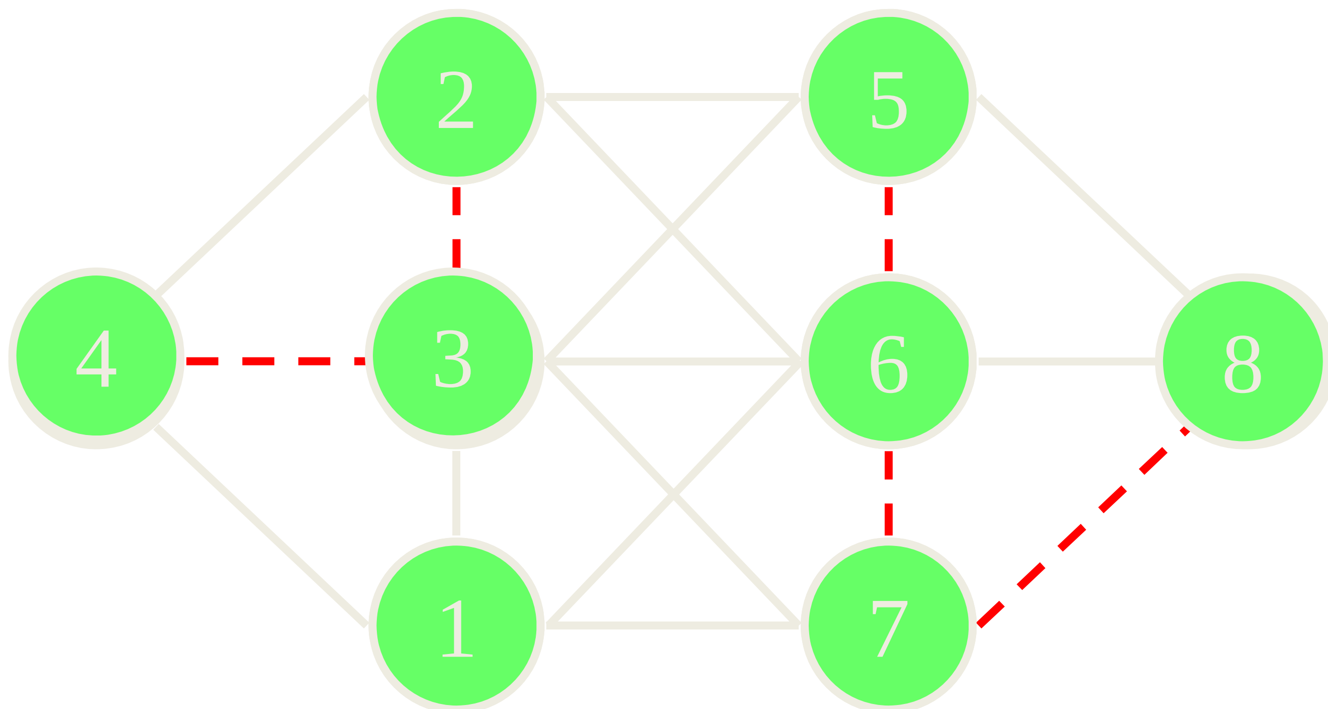
Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	0	0					
2		0						
3			0					
4				0				
5					0			
6						0		
7							0	
8								0

Random Initial Solution



Swap 1 & 4



Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	0	0	-1				
2		0						
3			0					
4				0				
5					0			
6						0		
7							0	
8								0

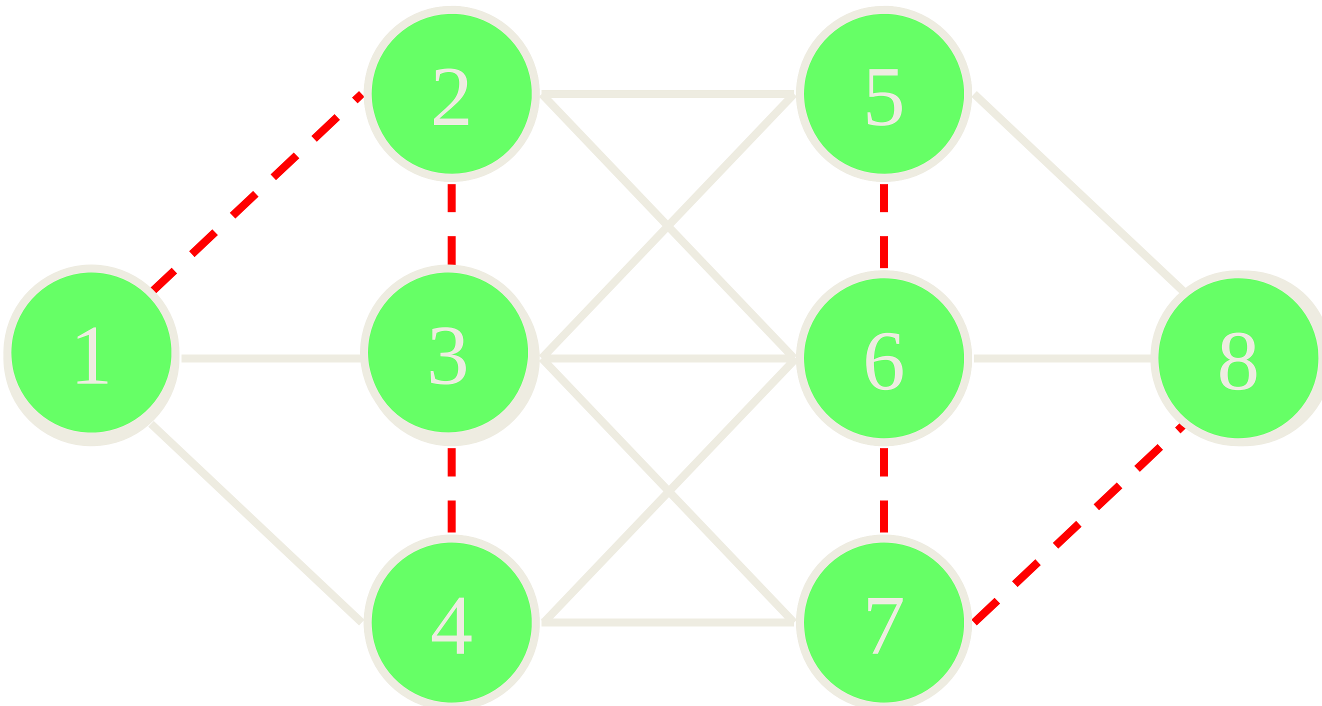
Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	0	0	-1	0	-2	-3	-2
2		0	-1	1	-1	-2	-1	-3
3			0	0	0	0	-1	0
4				0	0	0	-1	0
5					0	0	1	-1
6						0	-1	0
7							0	0
8								0

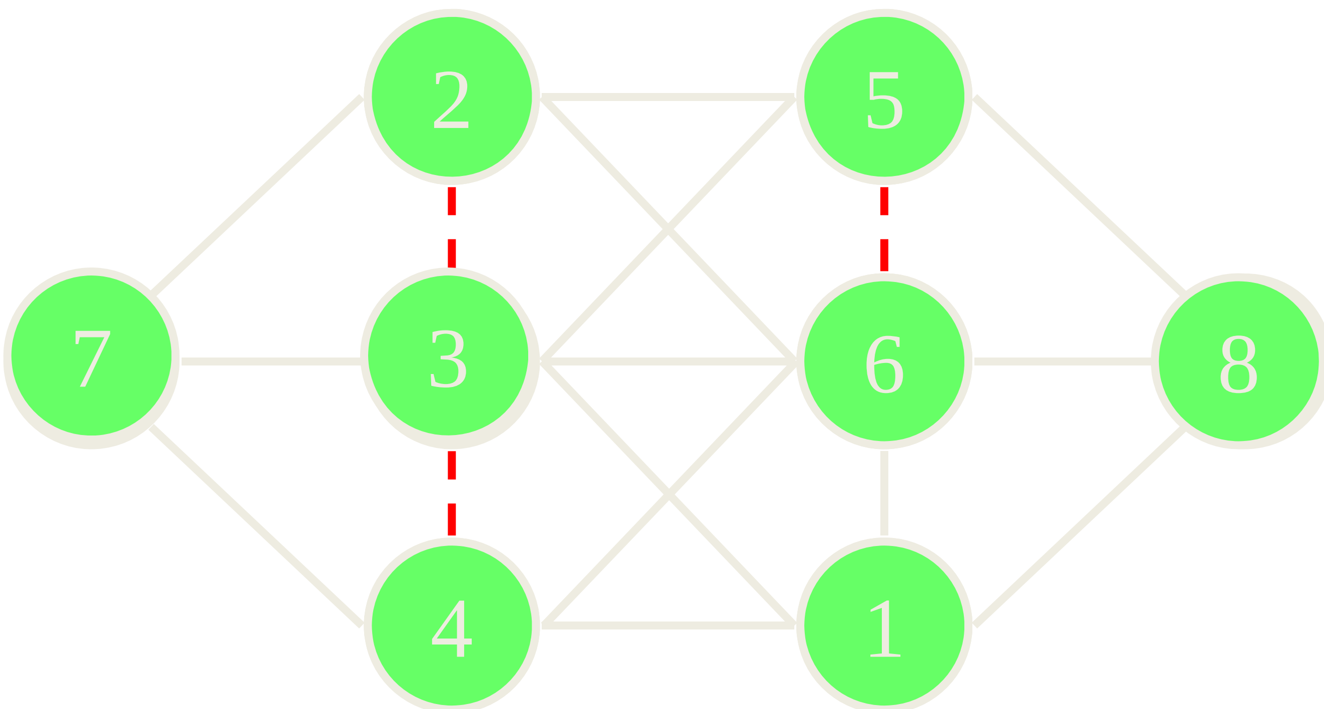
Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	0	0	-1	0	-2	-3	-2
2		0	-1	1	-1	-2	-1	-3
3			0	0	0	0	-1	0
4				0	0	0	-1	0
5					0	0	1	-1
6						0	-1	0
7							0	0
8								0

Current State



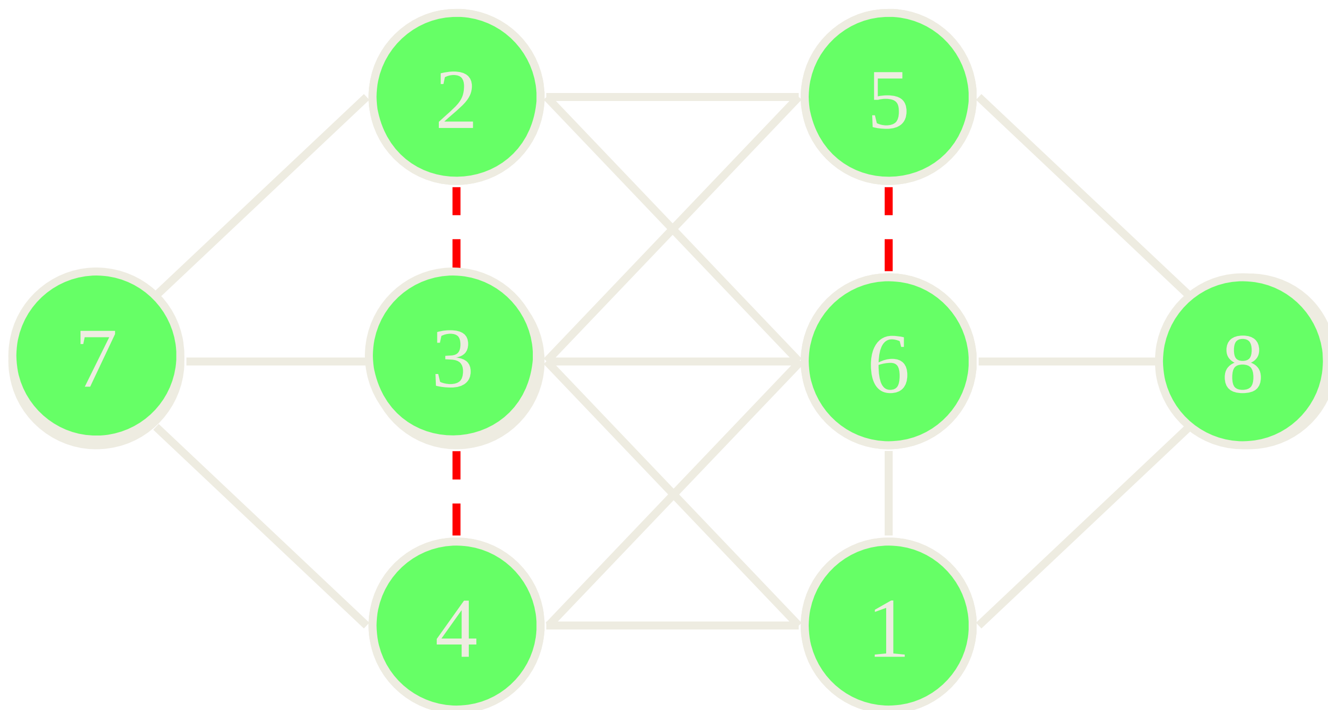
Swap 1 & 7: Cost 3



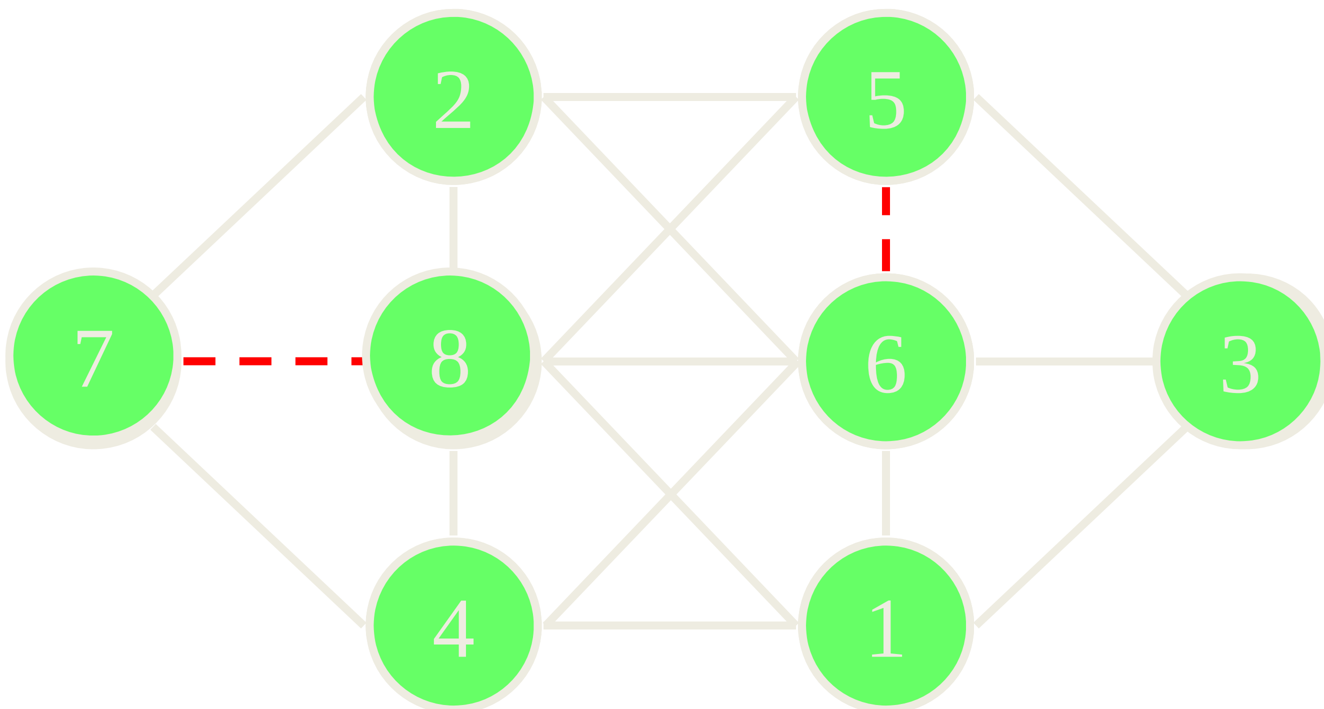
New Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	0	0	0	2	0	3	0
2		0	0	2	0	1	1	1
3			0	0	0	1	1	-1
4				0	0	1	1	1
5					0	1	2	0
6						0	0	0
7							0	1
8								0

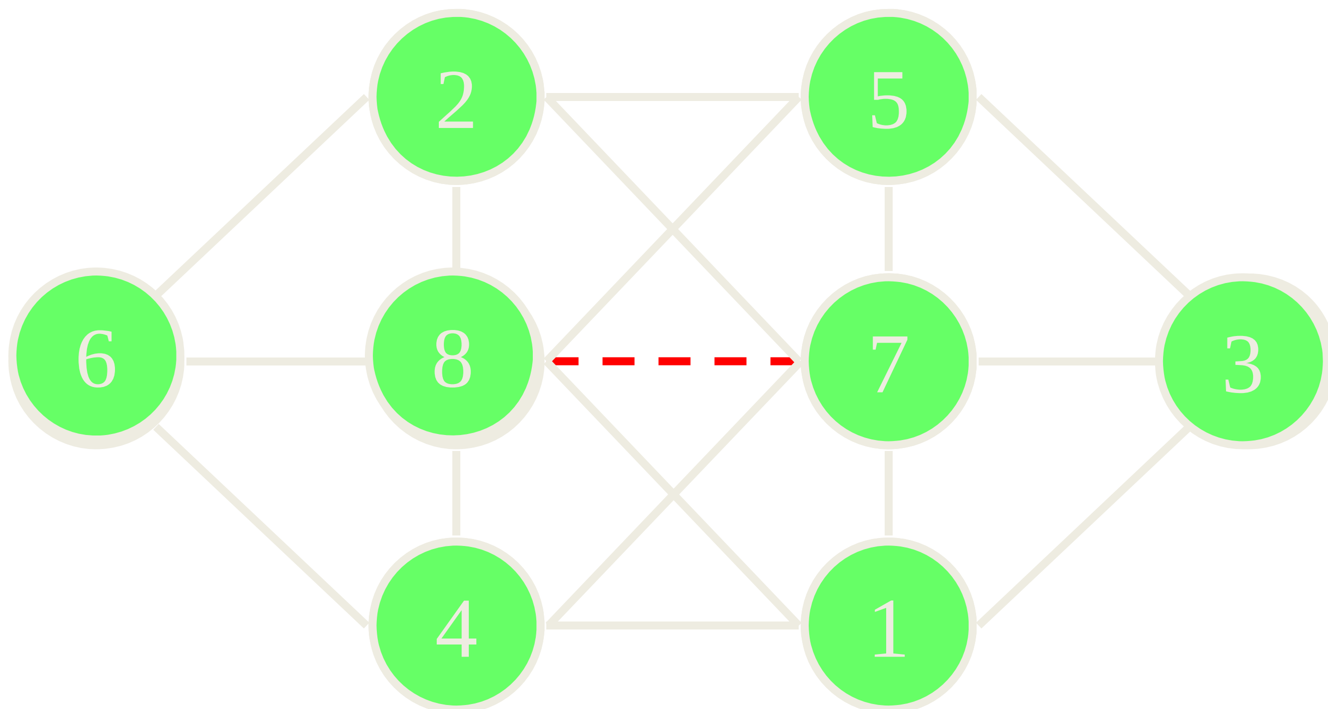
Current State



Swap 3 & 8: Cost 2



Swap 6 & 7: Cost 1



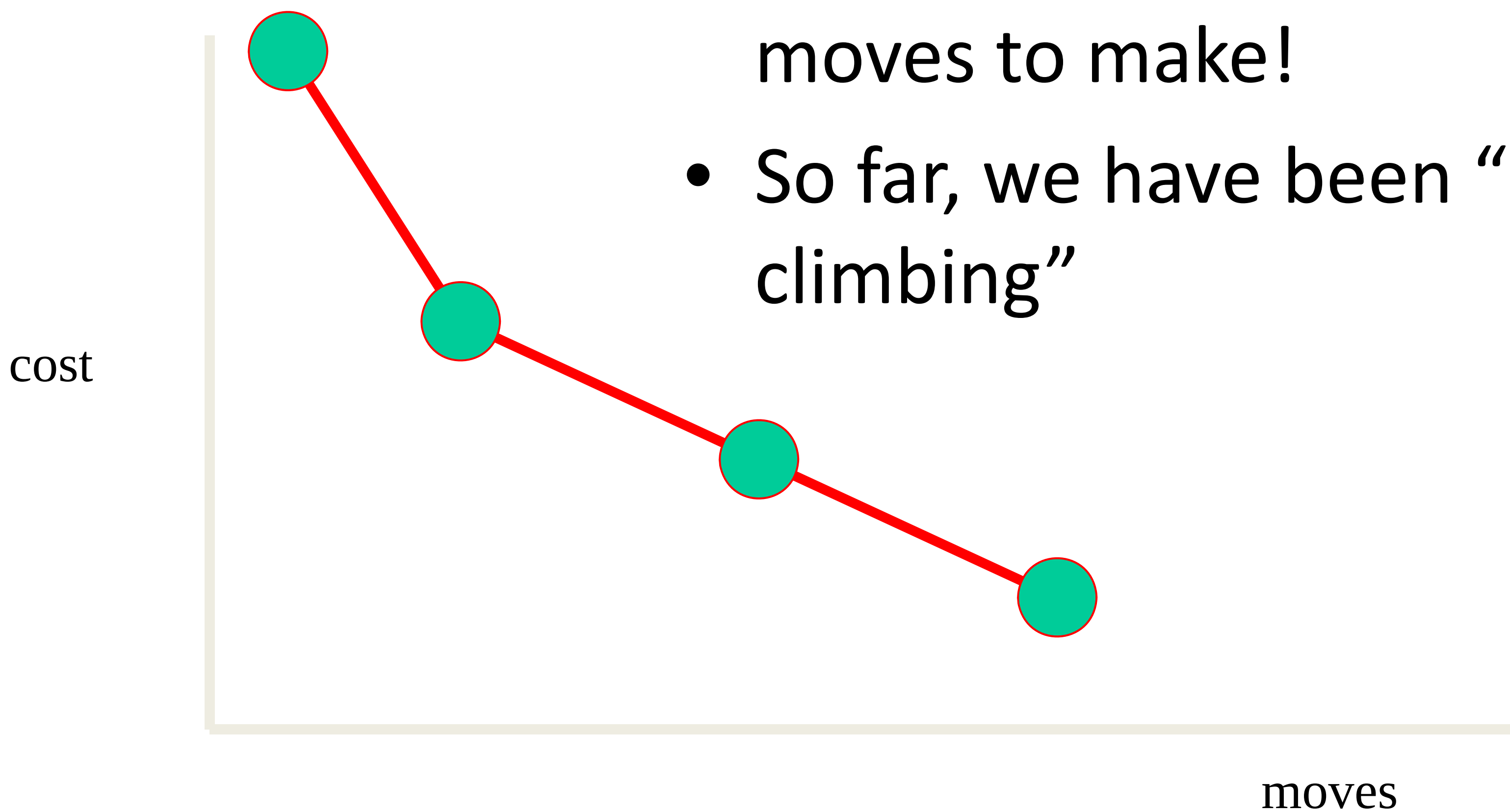
Moves

- Initial State: Cost 6
- Swap 1 & 7: Cost 3
- Swap 3 & 8: Cost 2
- Swap 6 & 7: Cost 1

Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	1	1	1	2	2	1	1
2		0	1	2	2	1	3	1
3			0	1	1	4	1	2
4				0	2	1	3	1
5					0	2	1	2
6						0	1	1
7							0	1
8								0

Now what?



- There are no improving moves to make!
- So far, we have been "hill-climbing"

Now what?

- Options:
  - Restart from a new random state
  - Take the least worse move (increase cost by minimal amount)
  - Try a new style of local search

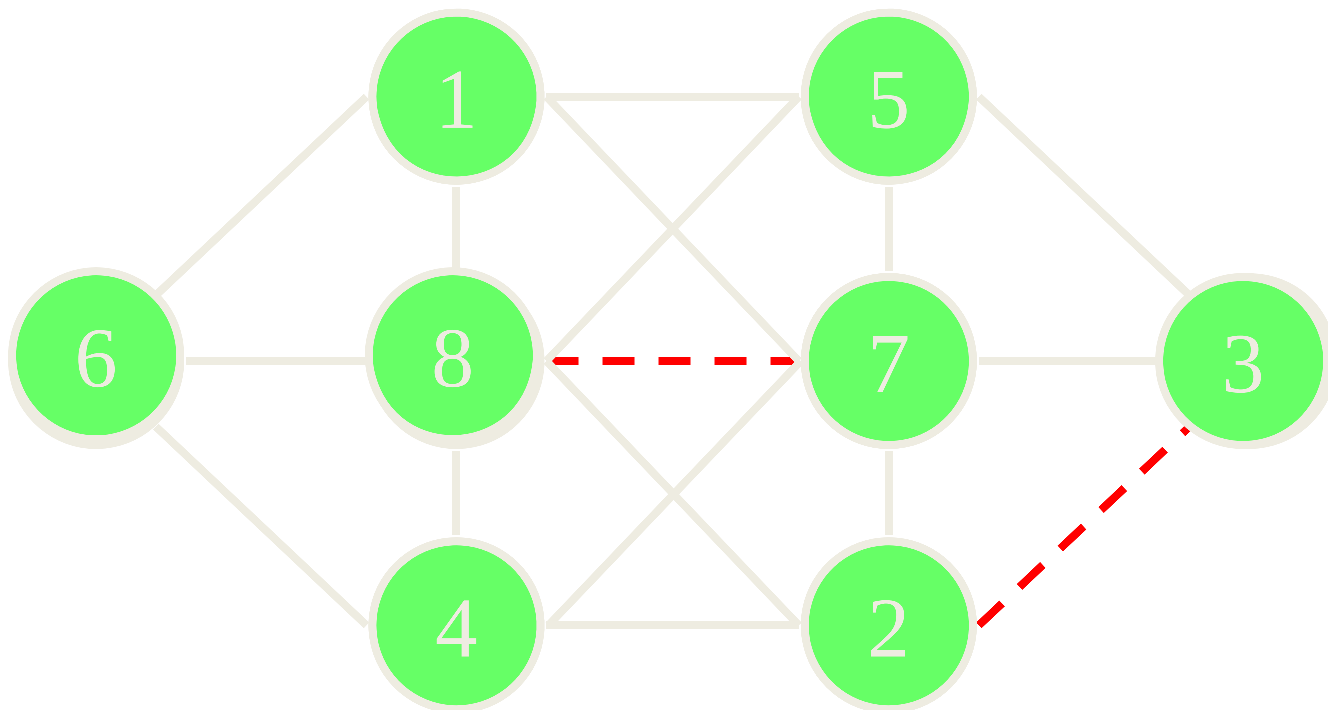
Now what?

- Options:
  - Restart from a new random state
  - Take the least worse move (increase cost by minimal amount)
  - Try a new style of local search

Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	1	1	1	2	2	1	1
2		0	1	2	2	1	3	1
3			0	1	1	4	1	2
4				0	2	1	3	1
5					0	2	1	2
6						0	1	1
7							0	1
8								0

Swap 1 & 2: Cost 2



Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	-1	0	2	1	2	2	1
2		0	1	0	2	1	1	0
3			0	1	-1	2	0	1
4				0	2	1	3	1
5					0	1	1	2
6						0	1	1
7							0	1
8								0

Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	-1	0	2	1	2	2	1
2		0	1	0	2	1	1	0
3			0	1	-1	2	0	1
4				0	2	1	3	1
5					0	1	1	2
6						0	1	1
7							0	1
8								0

Moves

- Initial State: Cost 6
- Swap 1 & 7: Cost 3
- Swap 3 & 8: Cost 2
- Swap 6 & 7: Cost 1
- Swap 1 & 2: Cost 2

Moves

- Initial State: Cost 6
- Swap 1 & 7: Cost 3
- Swap 3 & 8: Cost 2
- Swap 6 & 7: Cost 1
- Swap 1 & 2: Cost 2
- Swap 1 & 2: Cost 1

Moves

- Initial State: Cost 6
- Swap 1 & 7: Cost 3
- Swap 3 & 8: Cost 2
- Swap 6 & 7: Cost 1
- Swap 1 & 2: Cost 2
- Swap 1 & 2: Cost 1
- Swap 1 & 2: Cost 2
- Swap 1 & 2: Cost 1 … and so on

Now what?

- Options:
  - Restart from a new random state
  - Take the least worse move (increase cost by minimal amount)
  - Try a new style of local search

Now what?

- Options:
  - Restart from a new random state
  - Take the least worse move (increase cost by minimal amount)
  - Try a new style of local search

Tabu Search

- A type of local search
- Start with some (maybe random) initial state
- Look at the moves in the “neighborhood” and take the best one
- Remember the last *k* moves (“tabu list”) so you don’t undo them

Local Search

```
Procedure local search
    begin
        x = some initial starting point in S
        while improve(x) != 'no' do
                x = improve(x)
        return(x)
    end
```

Simulated Annealing

```
Procedure simulated Annealing
    begin
        x = some initial starting point in S
        while not termination-condition do
            x = improve?(x,T)
            update(T)
        return(x)
    end
```

Tabu Search

```
Procedure Tabu Search
   begin
         x = some initial starting point in S
         while not termination-condition do
                  x = improve?(x,H)
                  update(H)
         return(x)
   end
```

Simulated Annealing and Tabu search

Tabu Search is almost identical to simulated annealing with respect to the structure of the algorithm.

The function “improve(x,T)” change to “improve(x,H)” and returns an accepted solution y form the neighborhood of x , the acceptance is based on the history of the search H.

Tabu Search

Fred Glover in 1986, employs a different approach to doing exploration → Tabu Search

Tabu” is an alternate spelling for “taboo”.

Original Paper: Fred Glover, 1986, Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research, 5, 533–549.*

Tabu Search

The main idea behind tabu search is very simple. A "memory" forces the search to explore new areas of the search space.

Tabu search is basically deterministic.

We can memorize some solutions that have been examined recently and these become tabu points to be avoided in making decisions about selecting the next solution.

Tabu Search

- **Tabu Search, by Fred Glover, employs a different approach to doing exploration: it keeps around** a history of recently considered candidate solutions (known as the *tabu list) and refuses to return to* those candidate solutions until they're sufficiently far in the past.
- Original Paper:
- Fred Glover, 1986, Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research, 5, 533–549.*

Tabu List

- The simplest approach to Tabu Search is to maintain a **tabu list *L*, *of some maximum length l,*** of candidate solutions we've seen so far. Whenever we adopt a new candidate solution, it goes in the tabu list. If the tabu list is too large, we remove the oldest candidate solution and it's no longer taboo to reconsider.

Algorithm: Tabu Search

```
l ← Desired maximum tabu list length
n ← number of tweaks desired to sample the gradient

S ← some initial candidate solution
Best ← S
L ← {} a tabu list of maximum length l          ▷ Implemented as first in, first-out queue
Enqueue S into L
repeat
    if Length(L) > l then
        Remove oldest element from L
    R ← Tweak(Copy(S))
    for n − 1 times do
        W ← Tweak(Copy(S))
        if W ∉ L and (Quality(W) > Quality(R) or R ∈ L) then
            R ← W
    if R ∉ L and Quality(R) > Quality(S) then
        S ← R
        Enqueue R into L
    if Quality(S) > Quality(Best) then
        Best ← S
until Best is the ideal solution or we have run out of time
return Best
```

Work Space

- Tabu Search :

- continues
- Discrete

Work Space

- Tabu Search really only works in discrete spaces. What if your search space is real-valued numbers?

Work Space

- Tabu Search really only works in discrete spaces. What if your search space is real-valued numbers?

- In this situation, one approach is to consider a solution to be a member of a list if it is "sufficiently similar" to an existing member of the list. The similarity distance measure will be up to you

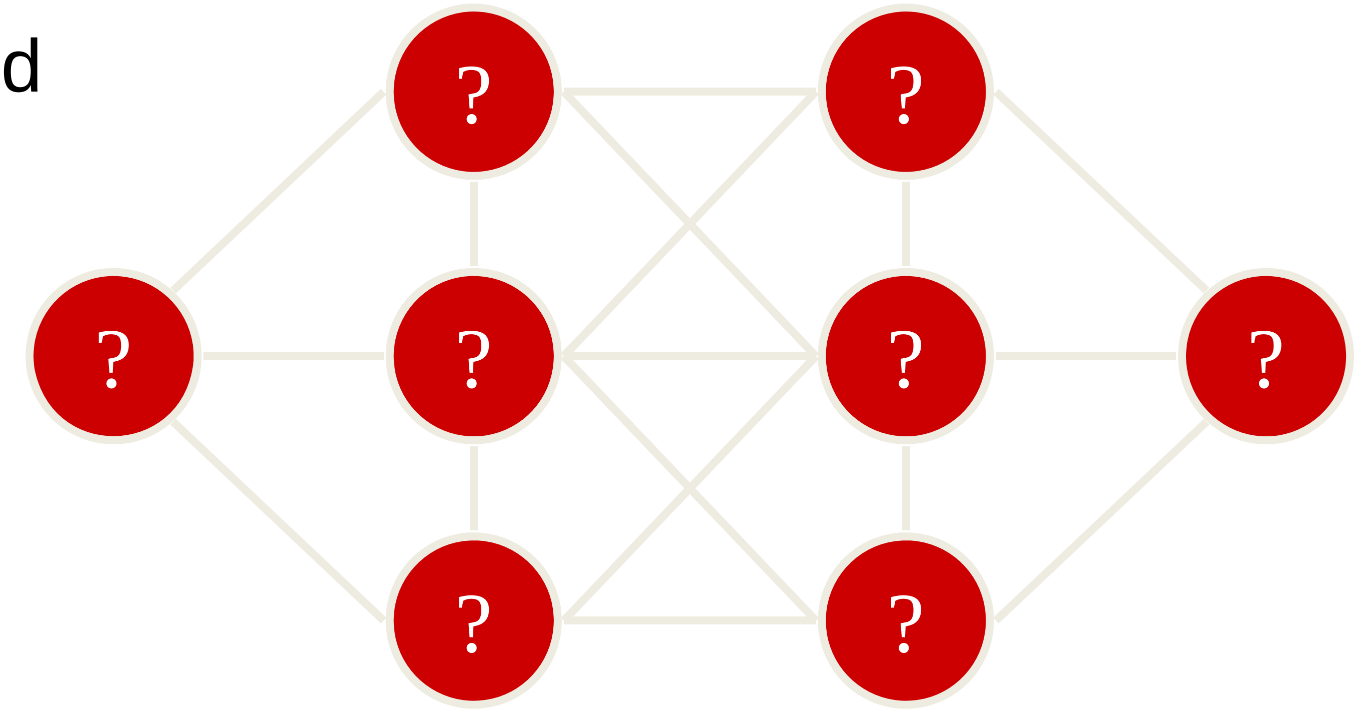
Large Problem

- so, the big problem with Tabu Search is that if your search space is very large, and particularly if it's of high dimensionality, it's easy to stay around in the same neighborhood, indeed on the same hill, even if you have a very large tabu list. There may be just too many locations. An alternative approach is to create a tabu list not of candidate solutions you've considered before, but of *changes you've made recently to certain features.*

Crystal Maze

- Place the numbers 1 through 8 in the nodes such that:
  - Each number appears exactly once

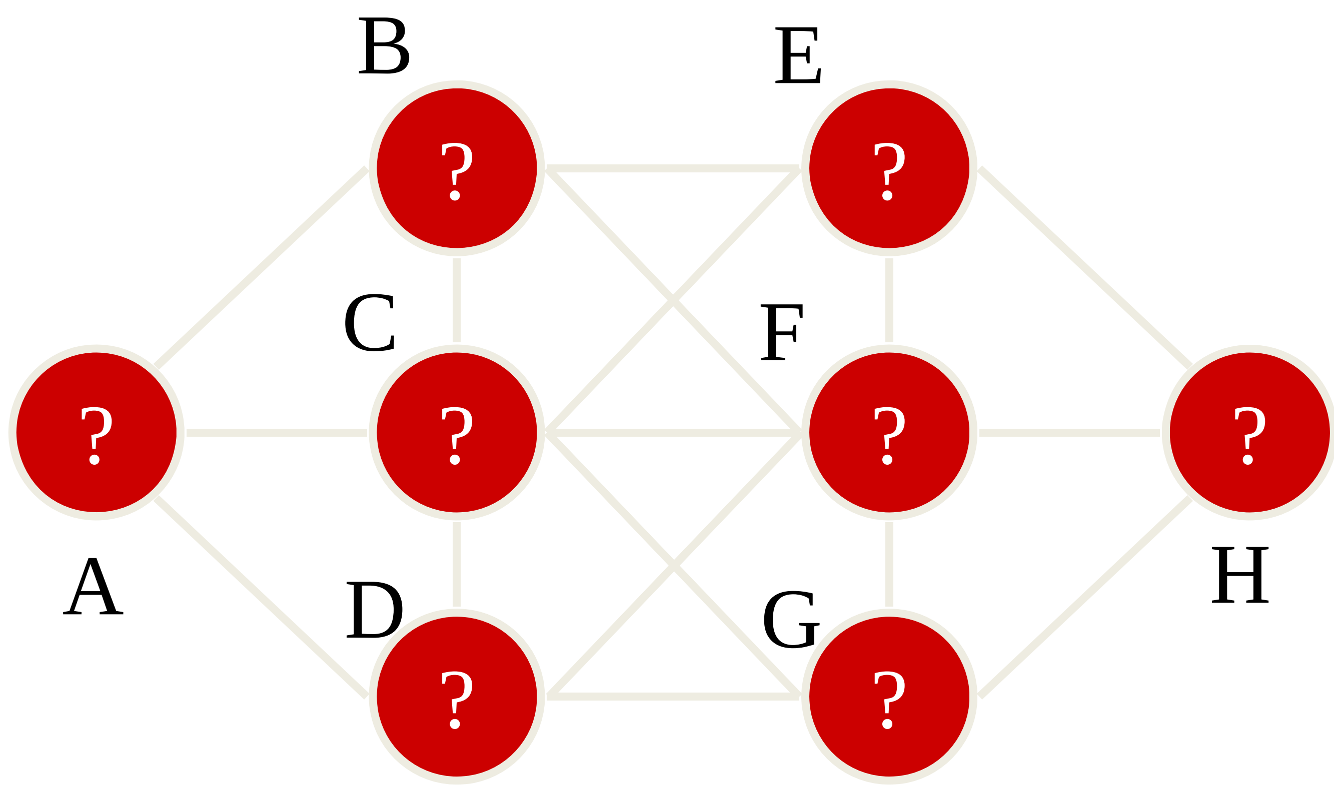
– No connected nodes have consecutive numbers



Tabu Search Idea

- Local search but:
  - Keep a small list of the moves we made so that we don’t revisit the same state
  - Keep a list of 4 pairs: the nodes that the numbers were in before we moved them

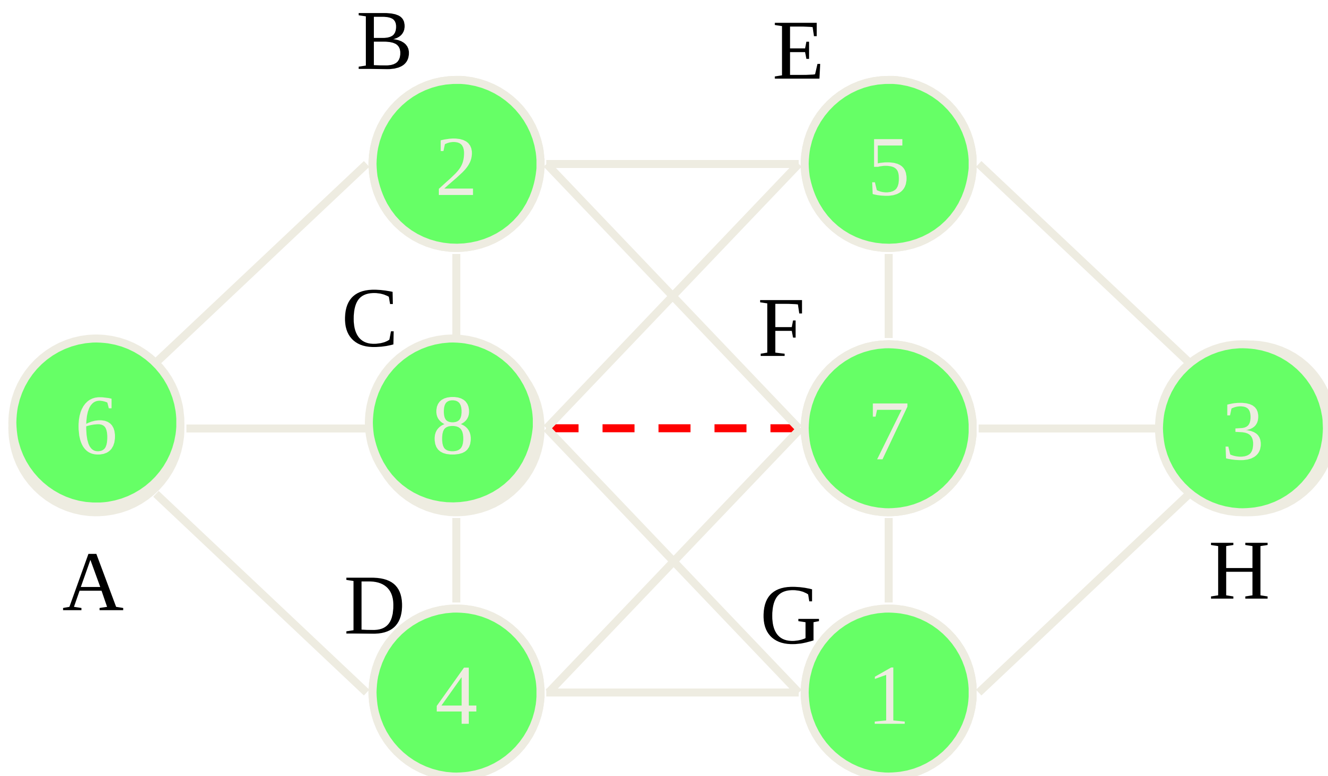
Assign Labels to Nodes



Return to a State Before We Started Cycling

- Initial State: Cost 6
- Swap 1 & 7: Cost 3
- Swap 3 & 8: Cost 2
- Swap 6 & 7: Cost 1

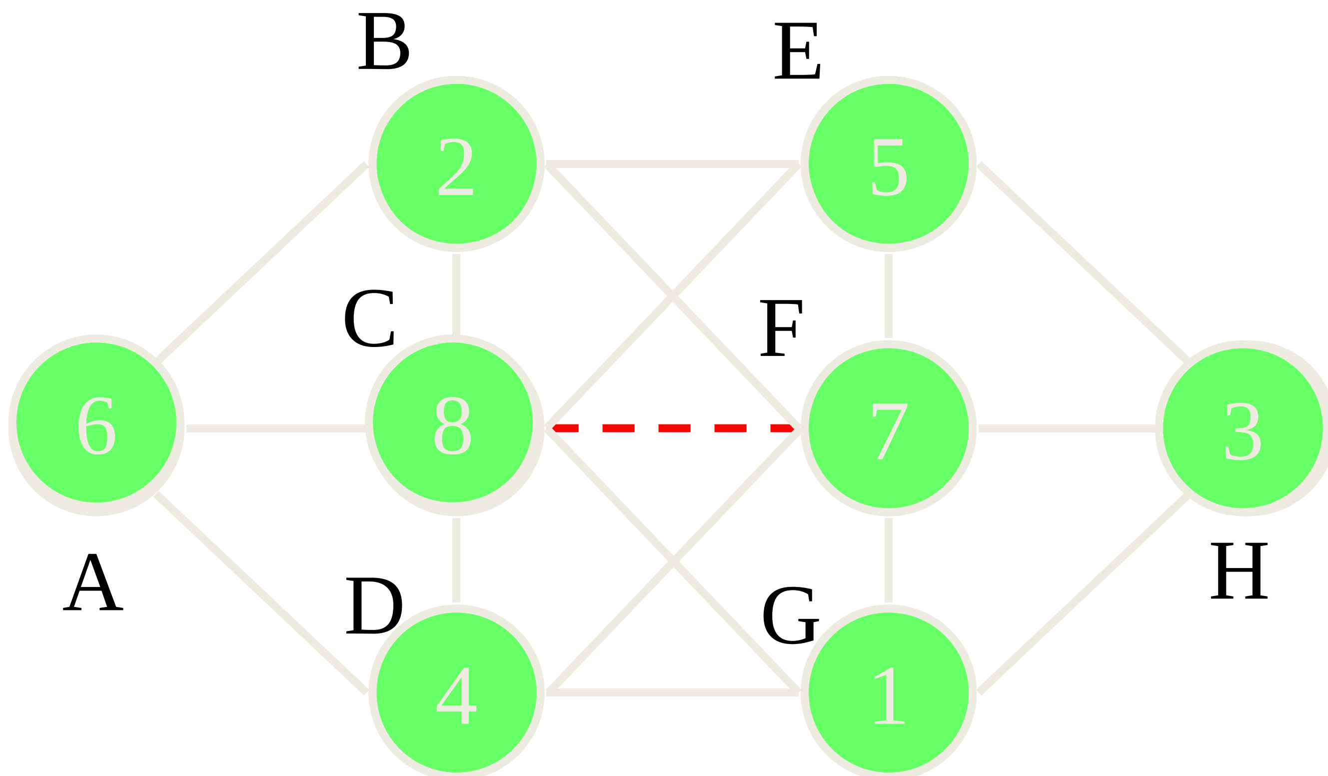
Just Swapped 6 & 7: Cost 1



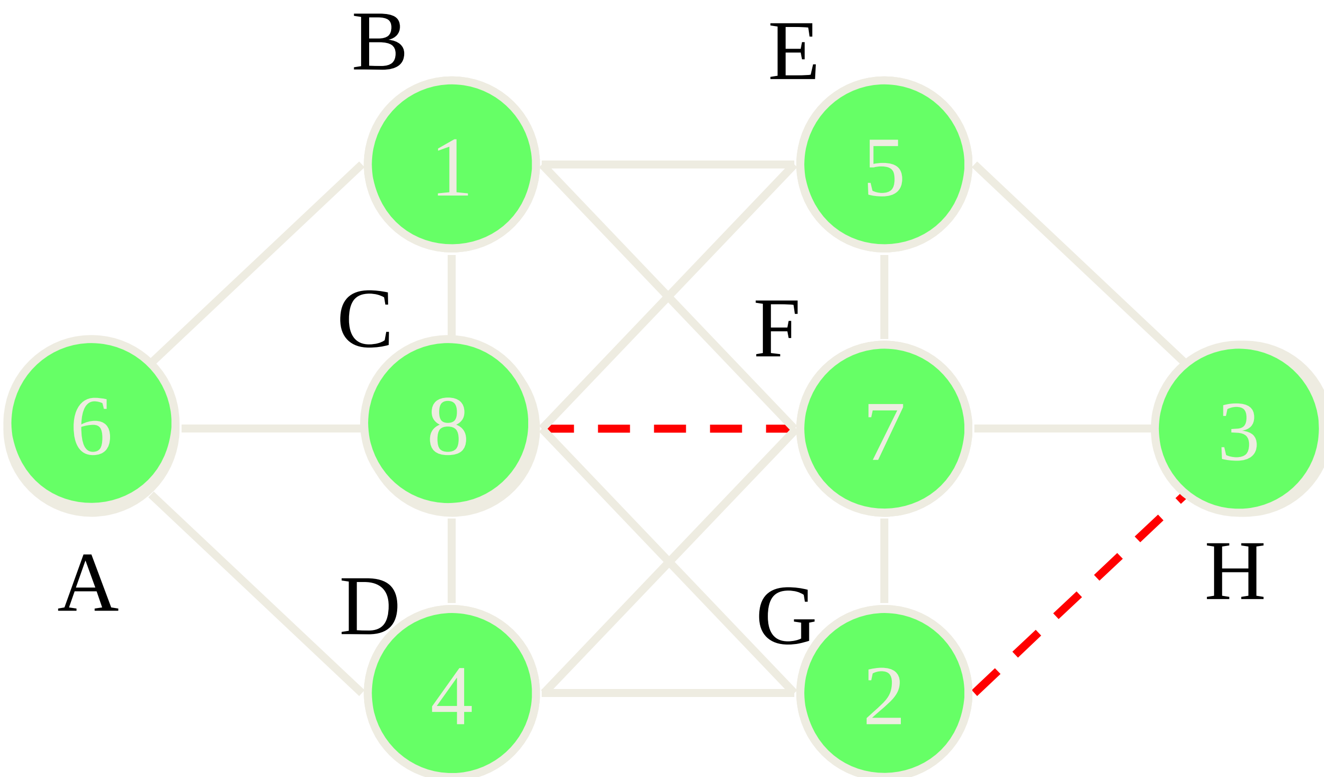
Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	1	1	1	2	2	1	1
2		0	1	2	2	1	3	1
3			0	1	1	4	1	2
4				0	2	1	3	1
5					0	2	1	2
6						0	1	1
7							0	1
8								0

Just Swapped 6 & 7: Cost 1



Swap 1 & 2: Cost 2



– Tabu: [(1G,2B)]

Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	-1	0	2	1	2	2	1
2		0	1	0	2	1	1	0
3			0	1	-1	2	0	1
4				0	2	1	3	1
5					0	1	1	2
6						0	1	1
7							0	1
8								0

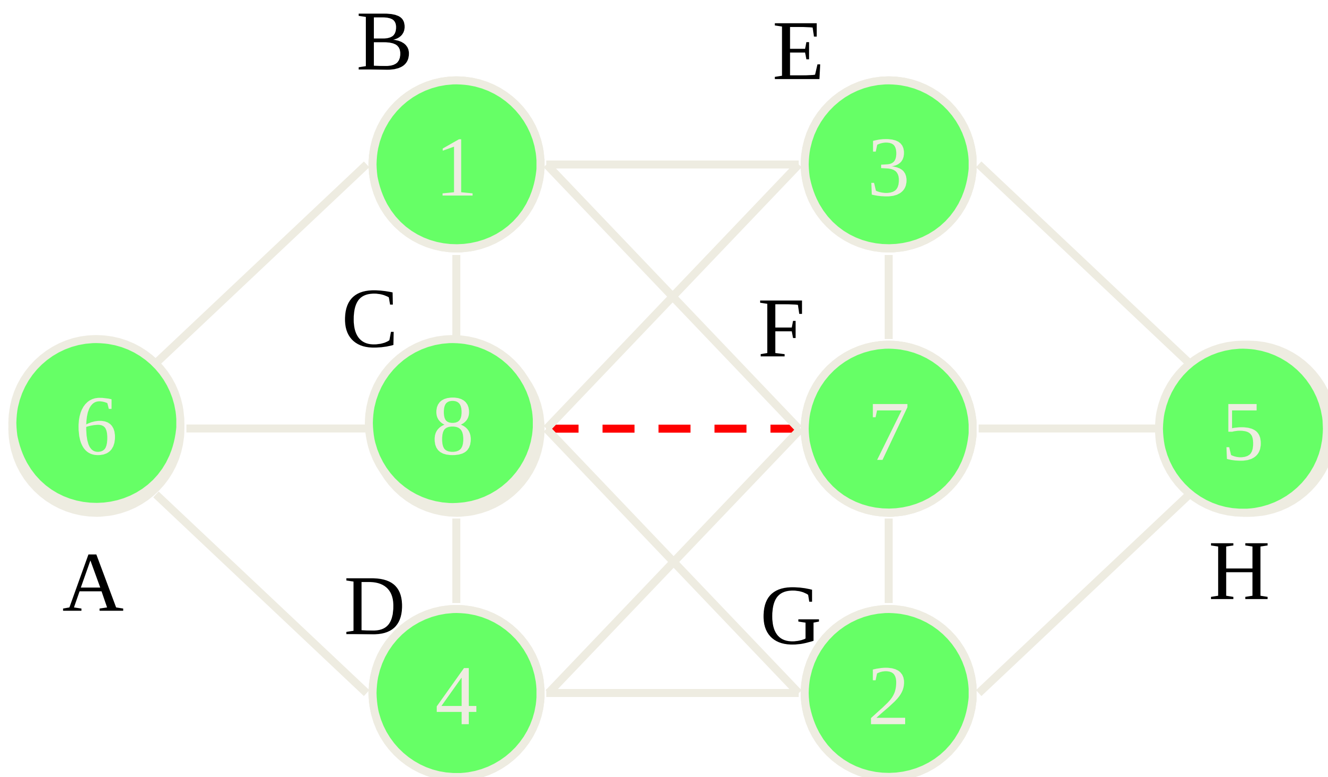
Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	X	0	2	1	2	2	1
2		0	1	0	2	1	1	0
3			0	1	-1	2	0	1
4				0	2	1	3	1
5					0	1	1	2
6						0	1	1
7							0	1
8								0

Cost Difference Table

	1	2	3	4	5	6	7	8
1	0	X	0	2	1	2	2	1
2		0	1	0	2	1	1	0
3			0	1	-1	2	0	1
4				0	2	1	3	1
5					0	1	1	2
6						0	1	1
7							0	1
8								0

Swap 3 & 5: Cost 1

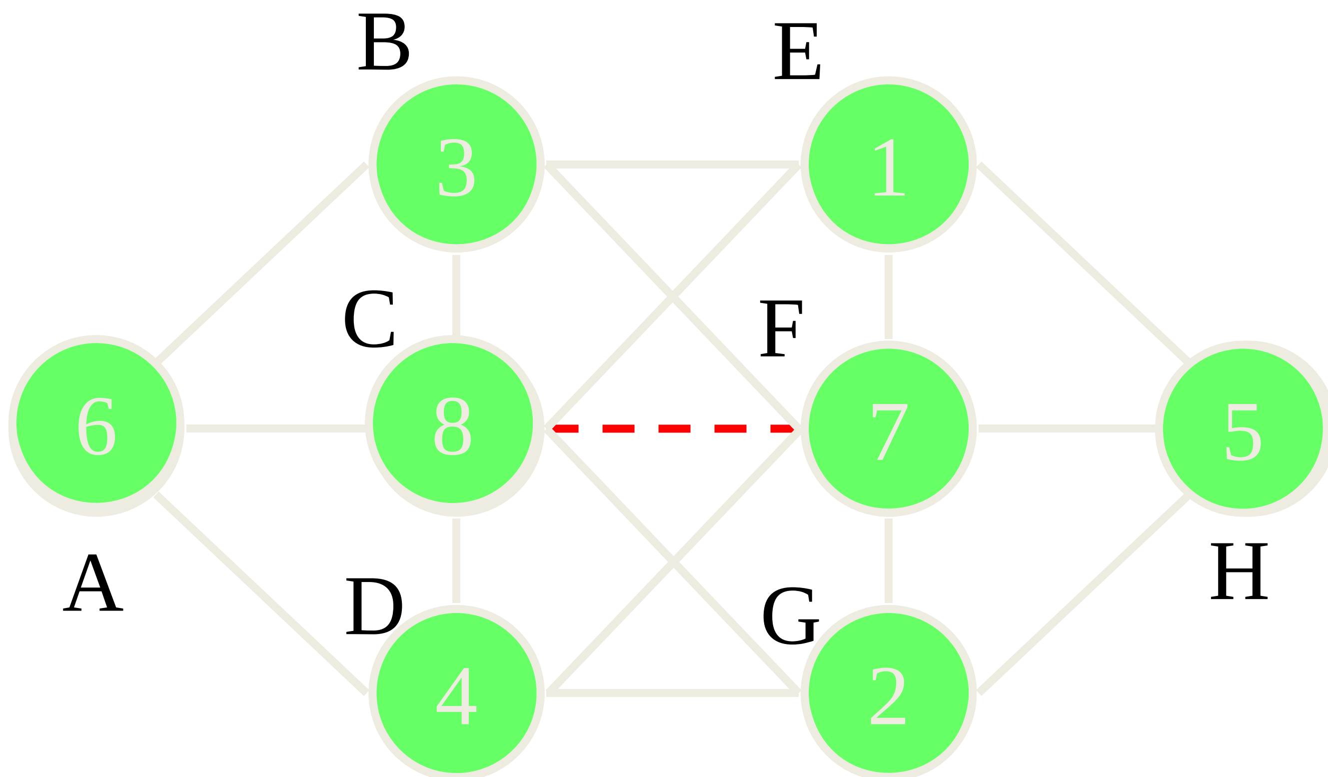


– Tabu: [(3H,5E),(1G,2B)]

Cost Difference Table

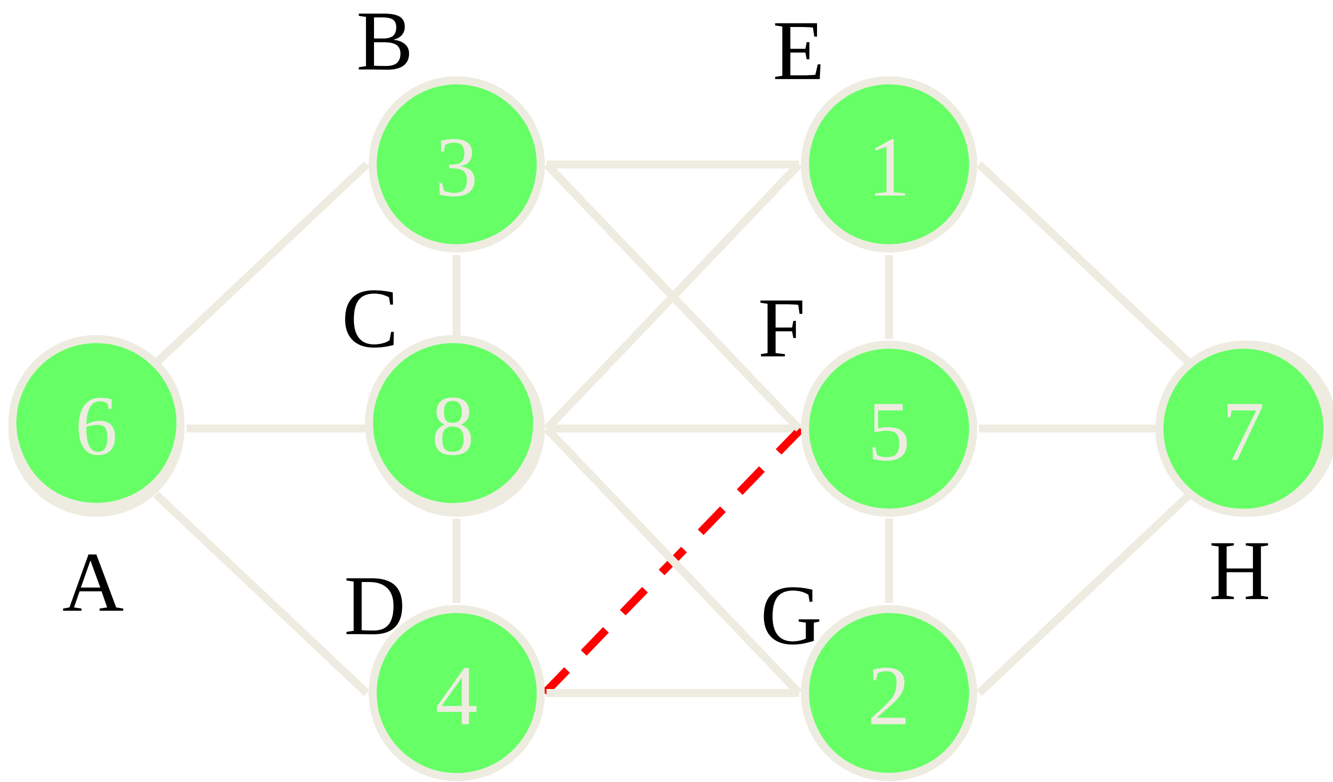
	1	2	3	4	5	6	7	8
1	0	X	0	2	2	1	2	1
2		0	2	1	2	3	2	2
3			0	2	X	3	2	2
4				0	2	1	3	1
5					0	2	0	2
6						0	1	0
7							0	1
8								0

Swap 1 & 3: Cost 1



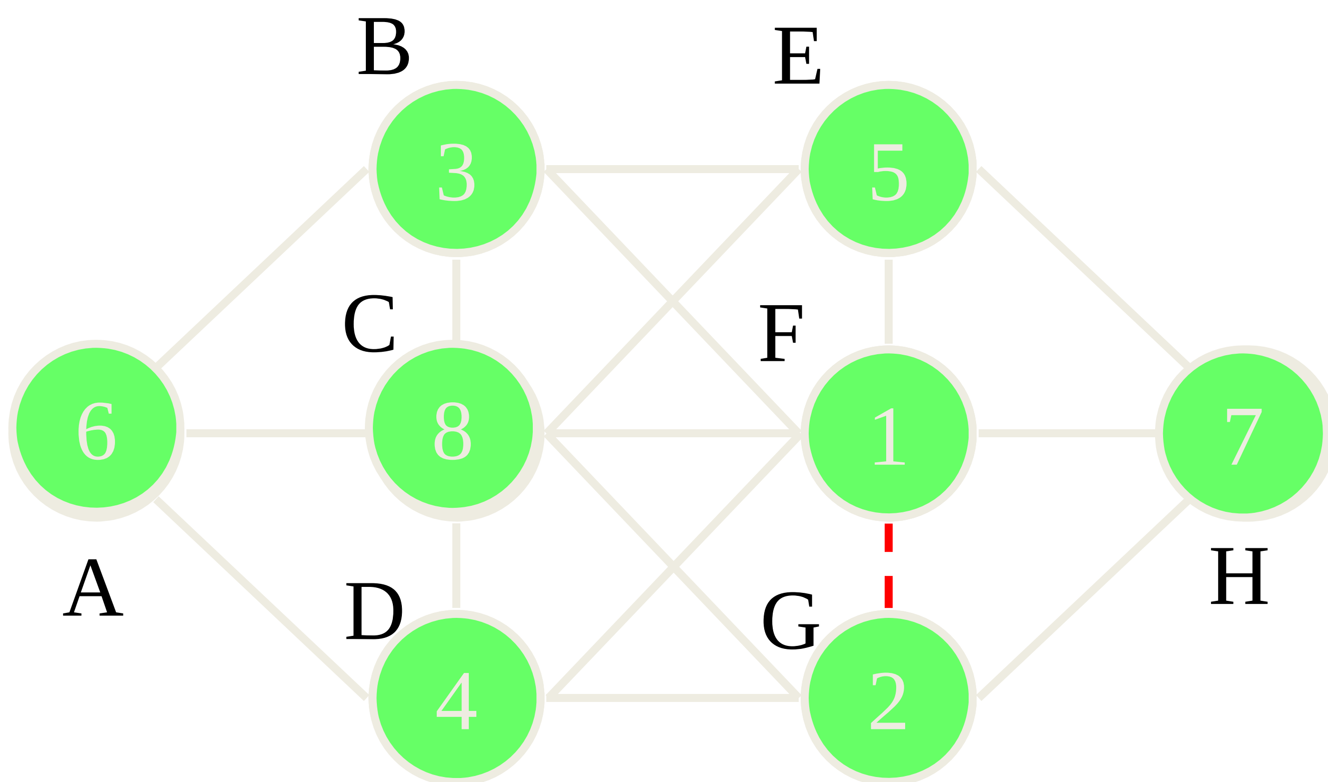
– Tabu: [(1B,3E),(3H,5E),(1G,2B)]

Swap 5 & 7: Cost 1



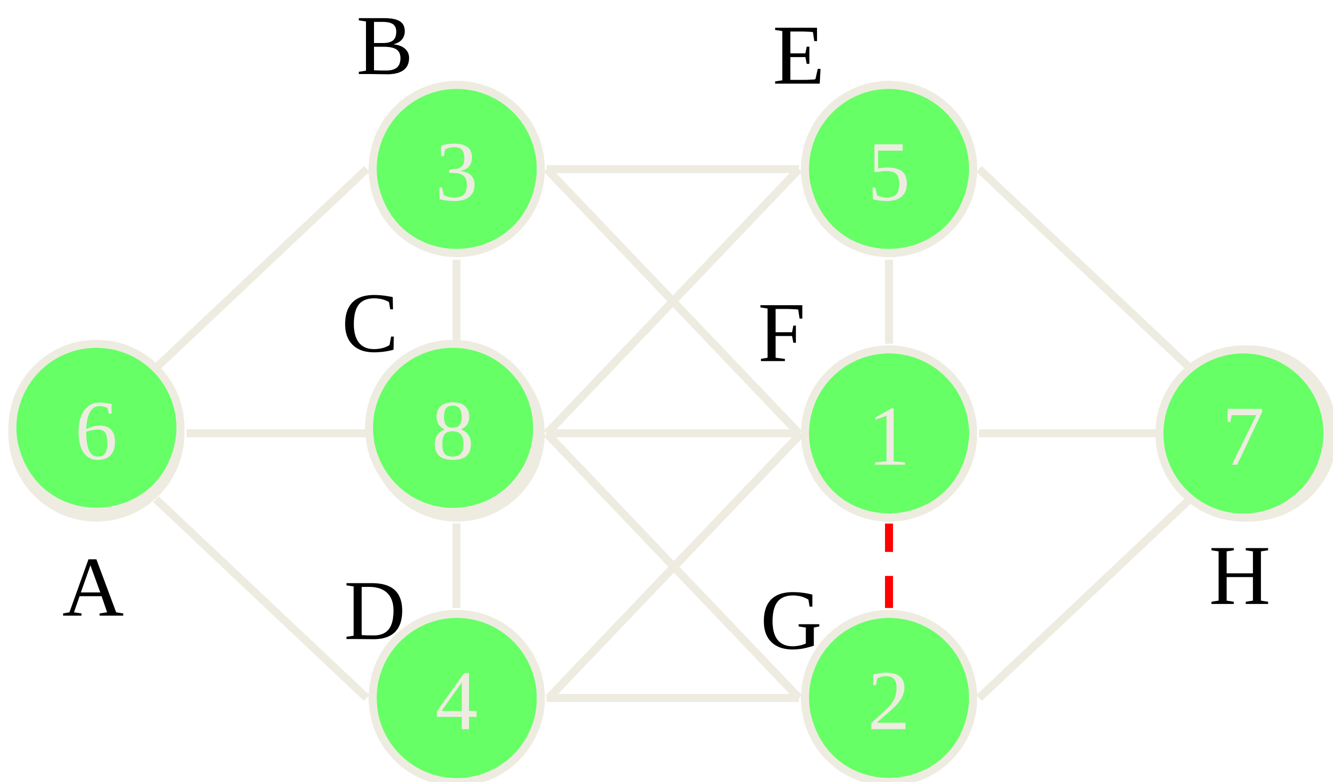
– Tabu: [(5H,7F),(1B,3E),(3H,5E),(1G,2B)]

Swap 1 & 5: Cost 1



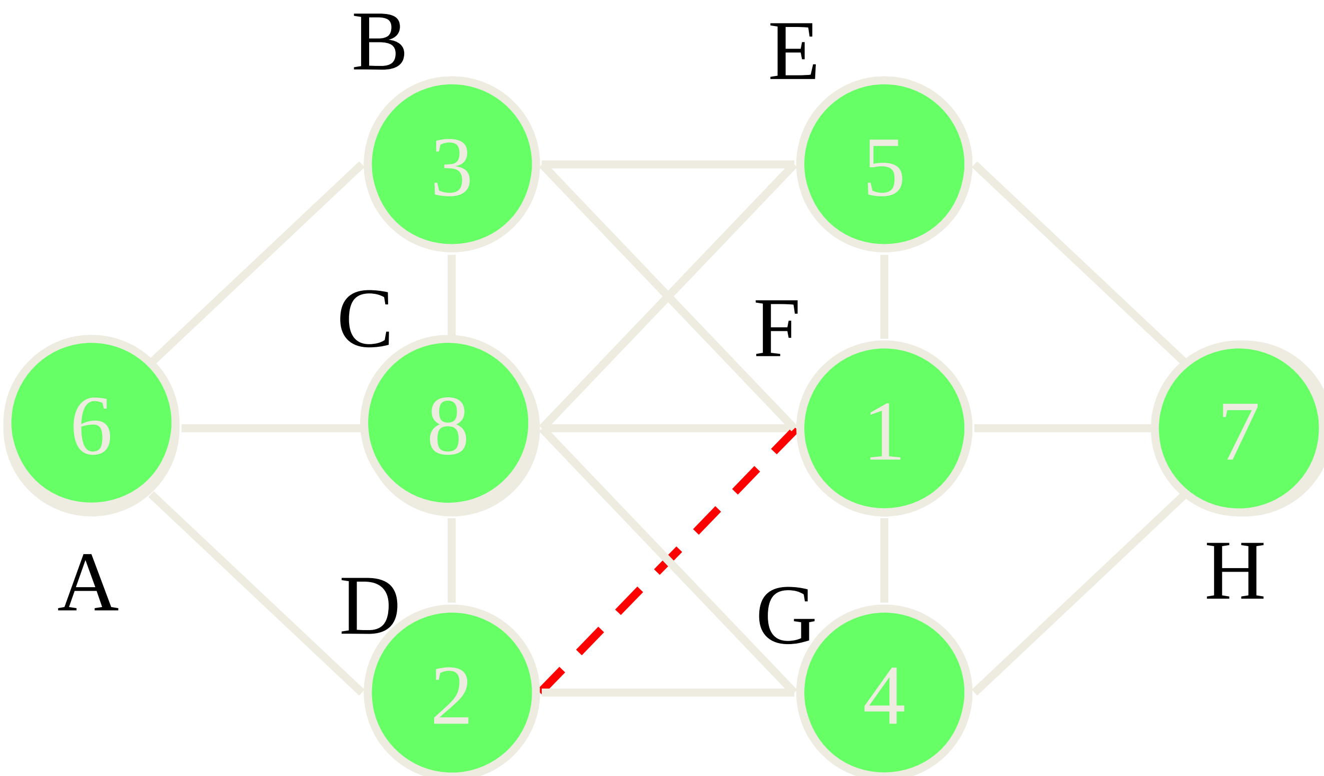
– Tabu: [(1E,5F),(5H,7F),(1B,3E),(3H,5E),(1G,2B)]

Swap 1 & 5: Cost 1



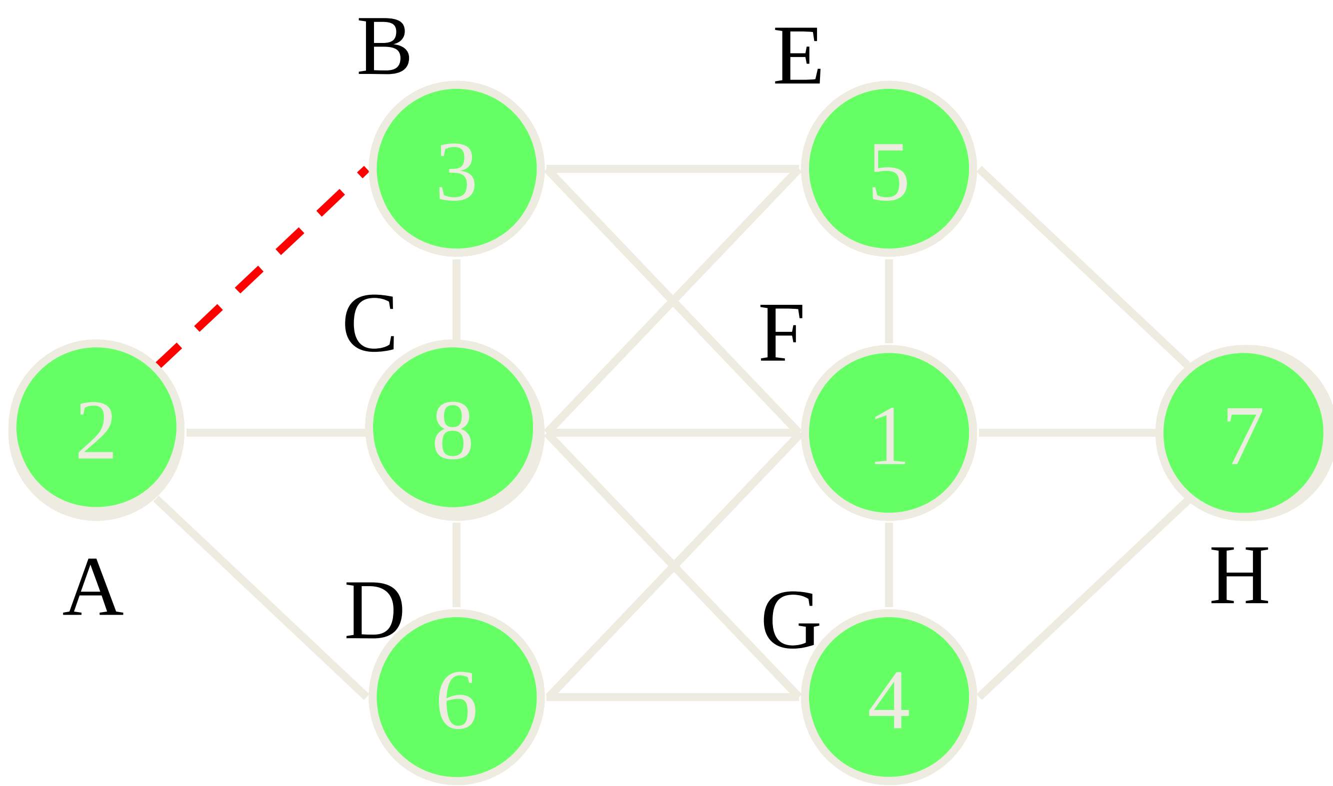
– Tabu: [(1E,5F),(5H,7F),(1B,3E),(3H,5E)]

Swap 2 & 4: Cost 1



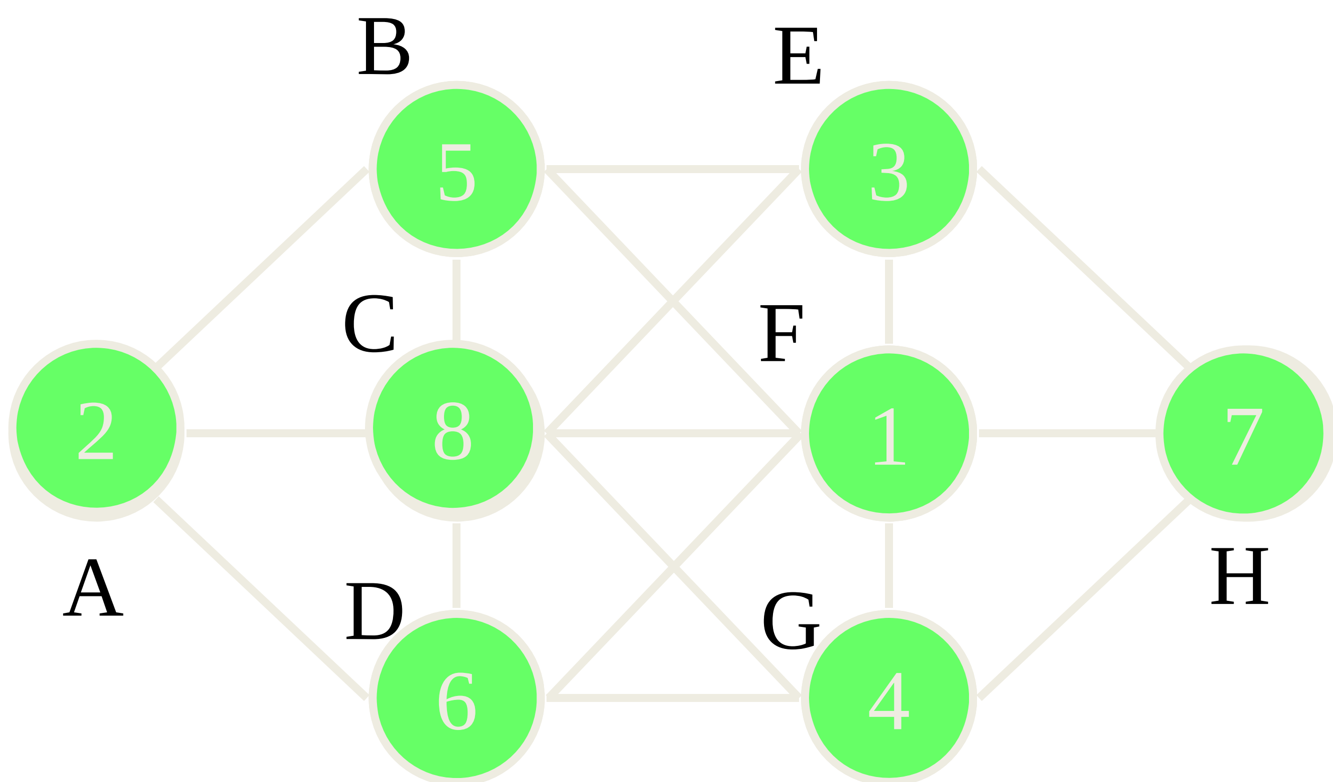
– Tabu: [(2G,4D),(1E,5F),(5H,7F),(1B,3E)]

Swap 2 & 6: Cost 1



– Tabu: [(2D,6A),(2G,4D),(1E,5F),(5H,7F)]

Swap 3 & 5: Cost 0



– Tabu: [(3B,5E),(2D,6A),(2G,4D),(1E,5F)]

Learn by Example !

Suppose we're solving SAT problem with n = 8 variables.
Initial assignment for x = (x1,…,x8) , x = (0,1,1,1,0,0,0,1).

function for evaluation : calculate a weighted sum of a number of satisfied clauses where the weights depend on the number of variables in the clause.
Evaluation function should be maximized.
Eval(Init State) = 27.

Each state consist of eight neighborhoods , each of which can be obtained by flipping a single bit in the vector x.

What about Memory ?!

New facet of tabu search : Memory. In order to keep a record of out actions , we'll need some memory structures for book keeping. We need memory M to keep time of flipping . So we can difference between older and more recent flips.

M(i) = j (when j != 0)

"j is the most recent iteration when the i-th bit was flipped" after some period of time , the information stored in memory is erased.

What about Memory ?!

Assuming that any piece of information can stay in a memory for at most, say, five iterations, a new interpretation of an entry :
M(i) = j (when j != 0)
“the i-th bit was flipped 5-j iterations ago”
Another interpretation , only requiring updating a single entry in the memory per iteration, and increasing iteration counter.

0	0	5	0	0	0	0	0

The contents of the memory after iteration 1

What about Memory ?!

The contents of memory after five iterations

3	0	1	5	0	4	2	0

Bits 2, 5, 8 are available to be flipped any time. Bit 1 is not available for he next three iterations, bit 3 isn't available but only for the next iteration, bit 4(which was just flipped) is not available for the next five iterations, and so on. Thus at the next iteration (iteration 6) it's impossible to flip bits 1,3,4,6 and 7, since all of these bits were tried "recently". These forbidden (tabu) solutions are not considered.

After one iteration the contents of the memory change as follows : all nonzero values are decreased by one to reflect the fact that all of the recorded flips took place one generation earlier.

2	0	0	4	5	3	1	0

The contents of the memory after iteration 6

Idea : at any stage there is a current solution being processed which implies a neighborhood, and from this neighborhood , tabu solution are eliminate from possible exploration.

Make search more flexible !

If we find an outstanding solution, we might forget about the principles!

In our life: great opportunity → forget principle

normal circumstance : selects a non-tabu solution as the next current solution , whether or nor this non-tabu solution has a better evaluation score than the current solution.

circumstance that aren't normal : an outstanding tabu solution is found in the neighborhood, such a superior solution is taken as the next point. This override of the tabu classification occurs when a so-called *aspiration criterion* is met.

Make search more flexible !

We could change the previous deterministic selection procedure into a probabilistic method where better solution have an increased chance of being selected.

We could change the memory horizon during the search. (what we do in our life....)

We might connect this memory horizon to the size of problem

We might connect the memory horizon/size to iteration

Another Memory-based improvements

Recency-bases memory (short term memory) : It only record some actions of the last few iterations.

Frequency-based memory (long term memory) : which operates over a much longer horizon, for example, a vector H may serve as a long-term memory.
H(i) = j is interpreted as "during the last h iterations of the algorithm, the i-th bit was flipped j times"

5	7	11	3	9	8	1	6

The contents of the frequency-based memory after 100 iterations (horizon = 50)

The principles of the tabu search indicate that this type of memory might be useful to diversify the search.

This memory concerning which flips have been under-represented (less frequent) or not represent at all and we can diversify the search by exploring these possibilities.

Long term memory

The use of long term memory in Tabu search is usually restricted to some special cases.

For Example:

When All non Tabu moves lead to worse situation.

How?

- Make more frequent moves less attractive

Penalty conditions

Makes the most frequent moves less attractive.

New circumstance : when all directions lead to lower solutions .
Assume the evaluation function for a new solution in such circumstances is :

$$eval(x') - penalty(x'),$$

where eval return the value of the original evaluation function and

$$penalty(x') = 0.7*H(i)$$

H(i) is the value taken from long term memory H.

Example

Tabu list/ short term memory

2	0	0	4	5	3	0	1

Frequency-based memory / long term memory

5	7	11	3	9	8	1	6

Vc = 35 current solution
Non-Tabu flips (2/3/7) → value: 30/33/31
Maximum Tabu flips → value 37 → we can not use aspiration criterion
→ Using Frequency based memory
Bit#2 → 30 – 0.7 * 7 = 25.1
Bit#3 → 33 – 0.7 * 11 = 25.3
Bit#7 → 31 – 0.7 * 1 = 30.3
→ We flip the bit 7

TSP Problem

- We conclude this section by providing an additional example of possible structures used in tabu search while approaching the TSP. For this problem , we can consider moves that swap two cities in a particular solution. The following solution (for an eight- city TSP),

  (2, 4, 7, 5, 1, 8 , 3, 6) ,

has 28 neighbors, since there are 28 different pairs of cities , that we can swap (two from eight).

TSP Continue

Recency-based memory

The swap of cities i and j is recorded in the i-th row and the j-th column (for $i < j$).
Note that we interpret i and j as cities, and not as their positions in the solution vector, but this might be another possibility to consider .

Note that the same structure can also be used for Frequency- based memory.

TSP Tabu

For clarity, we 'll maint ain the number of remaining it erations for which a given swap stays on the tabu list (recency-based memory) as with the previous SAT problem , while the frequency- based memory will indicate the totals of all swaps that occurred within some horizon h.

TSP Tabu

Assume both memories were initialized to zero and 500 iterations of the search have been completed. The current status of the search then might be as follows . The current solution is
(7, 3, 5, 6, 1, 2, 4, 8)
with the total length of the tour being 173. The best solution encountered during these 500 it erat ions yields a value of 171.

The contents of the recency-based memory M for the TSP after 500 iterations . The horizon is five iterations

	2	3	4	5	6	7	8
1	0	0	1	0	0	0	0
2		0	0	0	5	0	0
3			0	0	0	4	0
4				3	0	0	0
5					0	0	2
6						0	0
7							0

TSP Tabu

The current status of the search then might be as follows . The current solution is

(7, 3, 5, 6, 1, 2, 4, 8)

it 's easy to interpret the numbers in these memories . The value M(2, 6) = 5 indicates that the most recent swap was made for cities 2 and 6, i.e. , the previous current solution was (7, 3, 5, 2, 1, 6, 4, 8)

2	3	4	5	6	7	8	
0	0	1	0	0	0	0	1
	0	0	0	5	0	0	2
		0	0	0	4	0	3
			3	0	0	0	4
				0	0	2	5
					0	0	6
						0	7

Therefore , swapping cities 2 and 6 is tabu for the next five iterations .

Note that only 5 swap s (out of 28 possible swaps) are forbidden (tabu) .

TSP Tabu

The frequency- based memory provides some additional statistics of the search. It seems that swapping cities 7 and 8 was the most frequent (it happened 6 times in the last 50 swaps) , and there were pairs of cities (like 3 and 8) that weren't swapped within the last 50 iterations .

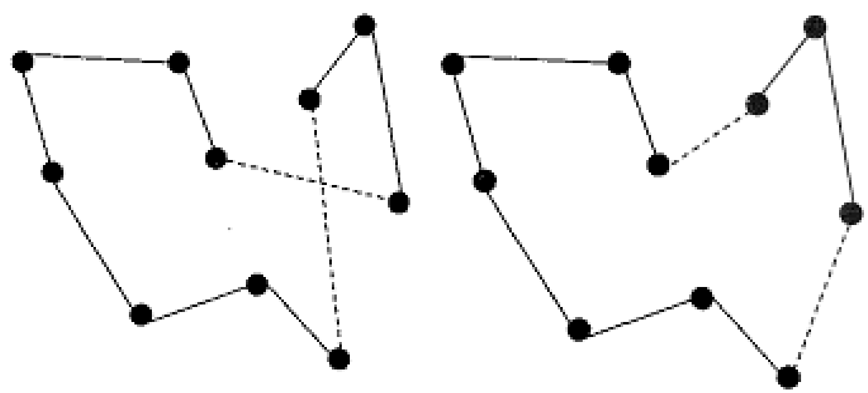
	2	3	4	5	6	7	8
1	0	2	3	3	0	1	1
2		2	1	3	1	1	0
3			2	3	3	4	0
4				1	1	2	1
5					4	2	1
6						3	1
7							6

The contents of the frequency-based memory F for the TSP after 500 iterations . The horizon is 50 iterations

TSP Tabu

The neighorhood of a tour was defined by a swap operation between two cities in the tour . This neighborhod is not the best choice either for tabu search or for simulated annealing. Many researchers have seleced larger neighborhoods .

There's a huge variety of local search algorithms for the TSP. The simplest is called *2-opt*. It starts with a random permutation of cities (call this tour T) and tries to improve it. The neighborhood of T is defined as the set of all tours that can be reached by changing two nonadjacent edges in T. This move is called a *2-interchange* and is illustrated in figure 3.6.



(a) (b)

Ref

- Slides adapted from Advanced Algorithms course, presented by Dr. kourosh ziarati