CPE 231 Algorithms

Iterative Improvement

Dr. Taweechai Nuntawisuttiwong

Contents

1. Iterative Improvement
2. The Maximum Matching in Bipartite Graphs
3. The Stable Marriage Problem

1. Greedy Method → Local optimal → no trace back → near optimal
2. Dynamic Programming → trace back to find the best solution
   old step / Recast step → best solution → optimal
3. Iterative improvement → initial solution → improve → optimal
4. Genetic Algorithm (Evolutionary method)

Recap Find exact answer
- Brute Force
- Decrease-and-Conquer
- Divide-and-Conquer
- Trasforms-and-Conquer
- Space and Time Trade off

Recap Optimization Problem
- Dynamic Programming
- Greedy Method
- Iterative Improvement

- objective fn. (Find max, min)
- constraints
- Feasible solution

Iterative Improvement

Iterative Improvement

- A technique for optimization problems that starts with a feasible solution and improves it step-by-step.
- Repeatedly apply a simple change to improve the objective function.
- When no further improvement is possible, the last solution is considered optimal.

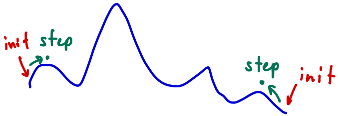
Comparison to Greedy Strategy

- **Greedy Strategy**: Builds a solution piece-by-piece, choosing locally optimal choices.
- **Iterative Improvement**: Starts with a complete feasible solution and enhances it iteratively.
- **Key Difference**: Iterative improvement modifies a complete solution, unlike greedy methods that construct one from scratch.

Challenges in Iterative Improvement

1. **Finding an Initial Solution**: Requires an initial feasible solution, which can sometimes be as challenging as solving the problem.
2. **Efficient Modification**: Must efficiently identify and apply beneficial changes.
3. **Local vs. Global Optimum**: Risk of stopping at a local optimum rather than a global one.
   - Example: Hiking in a hilly area with fog.
   - Climbing to the highest local point doesn't guarantee finding the global maximum.
   - Iterative improvement may converge to a local optimum, especially without knowledge of the global landscape.

Exploitation



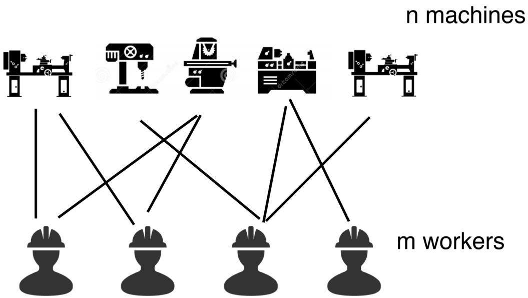
Application

1. Linear Programming and the Simplex Method
   - Developed by George Dantzig (1947), it's a classic algorithm that iteratively improves solutions for linear programming problems.
2. Network Flow and the Ford-Fulkerson Algorithm
   - Maximizing flow through a network with limited capacities.
3. Bipartite Matching
   - Pair elements from two disjoint sets, maximizing matched pairs or ensuring stability.
4. Other Applications
   - Traveling Salesman and Knapsack Problems
   - Heuristic Search Methods

The Maximum Matching in Bipartite Graphs

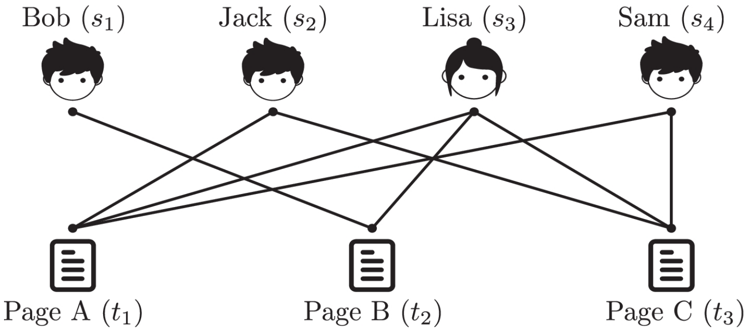
Maximum Matchiing

- A matching in a graph is a subset of its edges with the property that no two edges share a vertex.
- A maximum matching (maximum cardinality matching) is a matching with the largest number of edges.
- Applications: pairing workers with jobs, students with schools, etc.



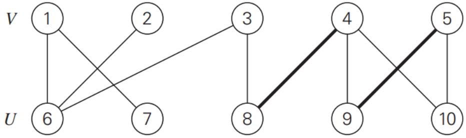
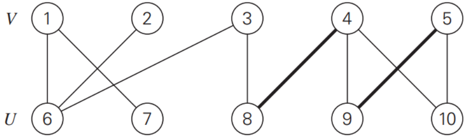
Bipartite Graphs

- A graph whose vertices can be split into two disjoint sets V and U.
- Every edge connects a vertex in V to one in U.
- Example: Pairing two distinct groups such as applicants and job openings.

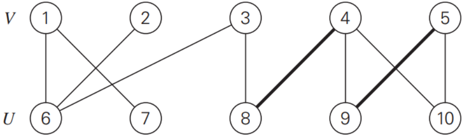
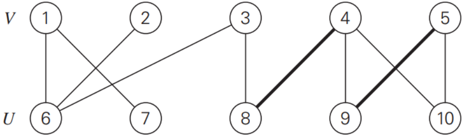


Augmenting paths

- A path that starts and ends with unmatched vertices and alternates between edges in and out of the matching.
- Allows us to increase the size of a matching by swapping edges along the path.

Bipartite Graph	Augmenting path: 1, 6
	

Augmenting paths

Augmenting path: 2, 6, 1, 7	Augmenting path: 3, 8, 4, 9, 5, 10
	

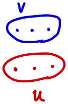
Theorem

A matching M is a maximum matching if and only if there exists no augmenting path with respect to M.

Maximum Matching Algorithm

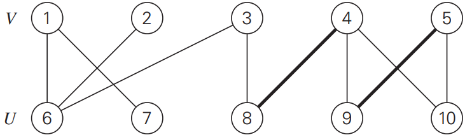
1. **Initialize** with an empty matching.
2. **Find Augmenting Path**: Use a BFS-like traversal starting from free vertices.
3. **Augment Matching**: Adjust the matching along the discovered path.
4. **Repeat Until No Augmenting Path Exists**: When no more paths can be found, the current matching is maximum.

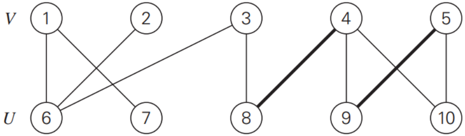
Maximum Matching Algorithm



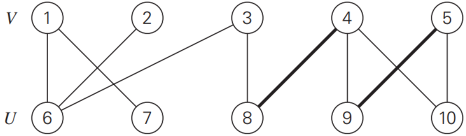

```
ALGORITHM  MaximumBipartiteMatching(G)
    //Finds a maximum matching in a bipartite graph by a BFS-like traversal
    //Input: A bipartite graph G = ⟨V, U, E⟩
    //Output: A maximum-cardinality matching M in the input graph
    initialize set M of edges with some valid matching (e.g., the empty set)
    initialize queue Q with all the free vertices in V (in any order)
    while not Empty(Q) do
        w ← Front(Q);   Dequeue(Q)
        if w ∈ V                                              Label → augmented path
            for every vertex u adjacent to w do
                if u is free
                    //augment
                    M ← M ∪ (w, u)                            } add argument matching
                    v ← w
                    while v is labeled do                              remove old matching
                        u ← vertex indicated by v's label;   M ← M − (v, u)   } Improve
                        v ← vertex indicated by u's label;   M ← M ∪ (v, u)
                    remove all vertex labels                           add augmenting matching
                    reinitialize Q with all free vertices in V
                    break   //exit the for loop
                else   //u is matched
                    if (w, u) ∉ M and u is unlabeled          } add augmenting matching
                        label u with w
                        Enqueue(Q, u)
        else   //w ∈ U (and matched)
            label the mate v of w with w                      } add augmenting matching
            Enqueue(Q, v)
    return M   //current matching is maximum
```

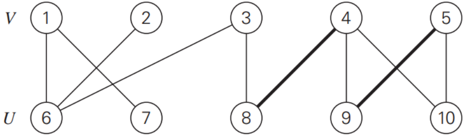
Maximum Matching Algorithm





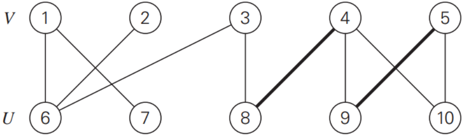
Queue:	Queue:



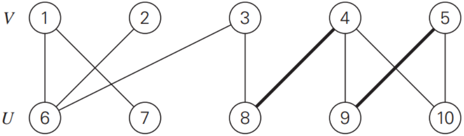


Queue:	Queue:

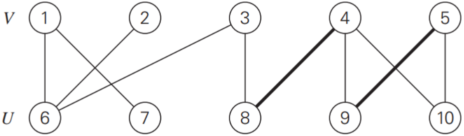
Maximum Matching Algorithm



Queue:



Queue:



Queue:

Algorithm Efficiency

- Let $n = |V| + |U|$ represent the total number of vertices in the bipartite graph.
- Let $m = |E|$ represent the total number of edges in the graph.
- In the worst case, there are $\lfloor n/2 \rfloor$ augmenting paths (i.e., half of the vertices need to be matched), so the algorithm requires up to $\lfloor n/2 \rfloor + 1$ iterations.
- Time complexity per iteration is $O(n + M)$.
- Since the maximum number of iterations is $O(n)$, and each iteration takes $O(n + m)$ time, the total complexity of the algorithm is $O(n(n + m))$.

The Stable Marriage Problem

The Stable Marriage Problem

- A problem of finding a stable matching between two sets, e.g., men and women.
- Each person in one set is paired with one in the other set.
- **Stability Condition**: No pair prefers each other over their assigned partners (i.e., no "blocking pairs").
- Each participant ranks the other set in order of preference, with no ties.
- **Blocking Pair**: A man and woman not matched to each other but who both prefer each other over their current partners.
- **Stable Matching**: A matching where no blocking pairs exist.

The Stable Marriage Problem

men's preferences

	1st	2nd	3rd
Bob:	Lea	Ann	Sue
Jim:	Lea	Sue	Ann
Tom:	Sue	Lea	Ann

women's preferences

	1st	2nd	3rd
Ann:	Jim	Tom	Bob
Lea:	Tom	Bob	Jim
Sue:	Jim	Tom	Bob

ranking matrix

	Ann	Lea	Sue
Bob			
Jim			
Tom			

Stable Marriage Algorithm

Stable marriage algorithm

Input: A set of n men and a set of n women along with rankings of the women by each man and rankings of the men by each woman with no ties allowed in the rankings

Output: A stable marriage matching

Step 0 Start with all the men and women being free.

Step 1 While there are free men, arbitrarily select one of them and do the following:

Proposal The selected free man m proposes to w, the next woman on his preference list (who is the highest-ranked woman who has not rejected him before).

Response If w is free, she accepts the proposal to be matched with m. If she is not free, she compares m with her current mate. If she prefers m to him, she accepts m's proposal, making her former mate free; otherwise, she simply rejects m's proposal, leaving m free.

Step 2 Return the set of n matched pairs.

Stable Marriage Algorithm

men's preferences

	1st	2nd	3rd
Bob:	Lea	Ann	Sue
Jim:	Lea	Sue	Ann
Tom:	Sue	Lea	Ann

women's preferences

	1st	2nd	3rd
Ann:	Jim	Tom	Bob
Lea:	Tom	Bob	Jim
Sue:	Jim	Tom	Bob

	Ann	Lea	Sue
Bob	2, 3	1, 2	3, 3
Jim	3, 1	1, 3	2, 1
Tom	3, 2	2, 1	1, 2

Free men:

Stable Marriage Algorithm

Free men:

	Ann	Lea	Sue
Bob	2, 3	1, 2	3, 3
Jim	3, 1	1, 3	2, 1
Tom	3, 2	2, 1	1, 2

Free men:

	Ann	Lea	Sue
Bob	2, 3	1, 2	3, 3
Jim	3, 1	1, 3	2, 1
Tom	3, 2	2, 1	1, 2

Stable Marriage Algorithm

Free men:

	Ann	Lea	Sue
Bob	2, 3	1, 2	3, 3
Jim	3, 1	1, 3	2, 1
Tom	3, 2	2, 1	1, 2

Free men:

	Ann	Lea	Sue
Bob	2, 3	1, 2	3, 3
Jim	3, 1	1, 3	2, 1
Tom	3, 2	2, 1	1, 2

Stable Marriage Algorithm

Free men:

	Ann	Lea	Sue
Bob	2, 3	1, 2	3, 3
Jim	3, 1	1, 3	2, 1
Tom	3, 2	2, 1	1, 2

Termination, Stability, and Characteristics

Theorem

The stable marriage algorithm terminates after no more than n^2 iterations with a stable marriage output.

- Stability: By construction, no unmatched man and woman will both prefer each other over their current partners.
- Man-Optimal Matching: The solution favors the proposing group (men in this case).
  - Each man is paired with the best partner he could have in any stable matching.
- Gender Bias: Reversing roles (letting women propose) would create a woman-optimal solution.