FUNDAMENTAL PROBLEMS AND ALGORITHMS

Graph Theory and Combinational

© Giovanni De Micheli
Stanford University
Shortest/Longest path problem

- **Single-source shortest path problem.**
- Model:
  - Directed graph $G(V, E)$ with $N$ vertices.
  - Weights on each edge.
  - A source vertex.
- **Single-source shortest path problem.**
  - Find shortest path from the source to any vertex.
  - Inconsistent problem:
    - Negative-weighted cycles.
Shortest path problem

Bellman’s equations: $G(V, E)$ with $N$ vertices

- $s_j = \min_{k \neq j} (s_k + w_{kj}); \quad j = 1, 2, \ldots, N$

- Acyclic graphs:
  - Topological sort $O(N^2)$.
  - $s_j = \min_{k < j} (s_k + w_{kj}); \quad j = 1, 2, \ldots, N$

- All positive weights:
  - Dijkstra’s algorithm.
Dijkstra’s algorithm

DIJKSTRA\((G(V, E, W))\)
\[
\begin{align*}
s_0 &= 0; \\
\text{for } (i = 1 \text{ to } N) &
\text{repeat }
\{ \\
& \quad \text{select unmarked } v_q \text{ such that } s_q \text{ is minimal; } \\
& \quad \text{mark } v_q; \\
& \quad \text{foreach (unmarked vertex } v_i ) \\
& \quad \quad s_i = \min \{s_i, (s_q + w_{q,i})\},
\}
\text{until (all vertices are marked)}
\end{align*}
\]

Apply to Korea’s map, robot tour, etc

G(V, E) with N vertices
**Bellman-Ford’s Algorithm**

Given a graph $G(V, E, W)$, the Bellman-Ford’s algorithm finds the shortest paths from a source vertex $s$ to all other vertices in the graph. The algorithm assumes that the edge weights do not change over time.

The pseudocode for the Bellman-Ford’s algorithm is as follows:

```plaintext
BELLMAN_FORD(G(V, E, W))
{
    $s_0 = 0$;
    for ($i = 1$ to $N$)
        $s_i = w_{0,i}$;
    for ($j = 1$ to $N$)
        for ($i = 1$ to $N$)
            $s_{j+1,i} = \min\{ s_{j,i}, (s_{j,k} + w_{q,i}) \}$, $k \neq i$
        if ($s_{j+1,i} = s_{j,i} \ \forall i$) return (TRUE);
    return (FALSE)
}
```

This algorithm iterates through all the edges of the graph for $N-1$ times, updating the shortest paths. If after $N-1$ iterations, there are no further updates, it indicates that there are no negative weight cycles in the graph.
Longest path problem

- Use shortest path algorithms:
  - by reversing signs on weights.
- Modify algorithms:
  - by changing min with max.
- Remarks:
  - Dijkstra’s algorithm is not relevant.
  - Inconsistent problem:
    - Positive-weighted cycles.
**Example – Bellman-Ford**

- **Iteration 1:** $l_0 = 0, l_1 = 3, l_2 = 1, l_3 = \infty$.
- **Iteration 2:** $l_0 = 0, l_1 = 3, l_2 = 2, l_3 = 5$.
- **Iteration 3:** $l_0 = 0, l_1 = 3, l_2 = 2, l_3 = 6$.

Use shortest path algorithms: by reversing signs on weights.
LIAO WONG(G( V, E∪ F, W))
{
    for ( i = 1 to N)
        1^i = 0;
    for ( j = 1 to |F| + 1) {
        foreach vertex v_i
            1^j+1 i = longest path in G( V, E,W_E ) ;
        flag = TRUE;
        foreach edge ( v_p, v_q) ∈ F {
            if ( 1^j+1 q < 1^j+1 p + w_{p,q}){
                flag = FALSE;
                E = E ∪ ( v_0 , v_q) with weight ( 1^j+1 p + w_{p,q})
            }
        }
        if ( flag ) return (TRUE) ;
    }
    return (FALSE)
Example – Liao-Wong

Looking for longest path from node 0 to node 3

**Iteration 1:** \( l_0 = 0, l_1 = 3, l_2 = 1, l_3 = 5 \).

Adjust: add edge \((v_0, v_1)\) with weight 2.

**Iteration 2:** \( l_0 = 0, l_1 = 3, l_2 = 2, l_3 = 6 \).

Only positive edges from (a) adjusted by adding longest path from node 0 to node 2.
Vertex cover

• Given a graph $G(V, E)$
  – Find a subset of the vertices
    • covering all the edges.

• Intractable problem.

• Goals:
  – Minimum cover.
  – Irredudant cover:
    • No vertex can be removed.
Example
Heuristic algorithm vertex based

VERTEX_COVERV(G(V; E))
{
    C = Ø;
    while (E ≠ Ø) do {
        select a vertex v ∈ V;
        delete v from G(V, E);
        C = C ∪ {f_v};
    }
}
Heuristic algorithm edge based

\[ \text{VERTEX\_COVER}(G(V, E)) \]

{ 

\[ C = \emptyset ; \]

\textbf{while} \ (E \neq \emptyset) \ \textbf{do} \ { }

\textbf{select an edge} \ \{u, v\} \in E ; \n
\[ C = C \cup \{u, v\} ; \]

\textbf{delete from} \ G(V, E) \ \textbf{any edge incident} \ 
\textbf{to either} \ u \ \textbf{or} \ v ; \n
\}

}
Graph coloring

- Vertex labeling (coloring):
  - No edge has end-point with the same label.
- Intractable on general graphs.
- Polynomial-time algorithms for chordal (and interval) graphs:
  - Left-edge algorithm.
Graph coloring heuristic algorithm

\text{VERTEX\_COLOR}(G(V, E))

\{ 
\qquad \text{for } (i = 1 \text{ to } |V| ) \{ 
\qquad \quad c = 1 \\
\qquad \quad \text{while } (\exists \text{ a vertex adjacent to } v_i \\
\qquad \quad \quad \text{with color } c) \text{ do } \{ 
\qquad \quad \quad \quad c = c + 1; \\
\qquad \quad \quad \quad \text{color } v_i \text{ with color } c ; \\
\qquad \quad \quad \}\n\qquad \}\n\}
EXACT_COLOR( G( V, E), k) 
{
  repeat {
    NEXT VALUE( k) ;
    if ( c_k == 0) return ;
    if ( k == n)
      c is a proper coloring;
    else
      EXACT_COLOR( G( V, E), k+ 1)
  }
}
Graph coloring exact algorithm

\text{NEXT VALUE}(k) \\
\{ \\
\text{repeat} \{ \\
\quad c_k = c_k + 1; \\
\quad \text{if} \ ( \text{there is no adjacent vertex to } v_k \\
\quad \quad \text{with the same color } c_k ) \\
\quad \quad \text{return} ; \\
\} \text{ until } (c_k \leq \text{maximum number of colors}) ; \\
\quad c_k = 0; \\
\}
Interval graphs

• Edges represent interval intersections.
• Example:
  – Restricted channel routing problem with no vertical constraints.
• Possible to sort the intervals by left edge.
Example
Example
**Left-edge algorithm**

**LEFT EDGE**(*I*)

{ 

Sort elements of *I* in a list *L* with ascending order of \( l_i \);

c = 0;

while (Some interval has not been colored ) do {

    \( S = \emptyset \);

    repeat {

        s = first element in the list *L* whose left edge \( l_s \) is higher than the rightmost edge in *S*.

        \( S = S \cup \{ s \} \);

    } until ( an element *s* is found );

    c = c + 1;

    color elements of *S* with color *c*;

    delete elements of *S* from *L*;

} }
Clique partitioning and covering

- A clique partition is a cover.
- A clique partition can be derived from a cover by making the vertex subsets disjoint.
- Intractable problem on general graphs.
- Heuristics:
  - Search for maximal cliques.
- Polynomial-time algorithms for chordal graphs.
CLIQUEPARTITION( G( V, E) )
{
    =∅;
    while ( G( V, E) not empty ) do {
        compute largest clique C V in G( V, E) ;
        =∪ C;
        delete C from G( V, E) ;
    }
}

CLIQUE( G( V, E) )
{
    C = seed vertex;
    repeat {
        select vertex v ∈ V , v∉ C
        and adjacent to all vertices of C;
        if (no such vertex is found) return
        C = C ∪ {v} ;
    }
}
Covering and Satisfiability

• Covering problems can be cast as satisfiability.
• Vertex cover.
  – Ex1: \((x_1 + x_2)(x_2 + x_3)(x_3 + x_4)(x_4 + x_5)\)
  – Ex2: \((x_3 + x_4)(x_1 + x_3)(x_1 + x_2)(x_2 + x_4)(x_4 + x_5)\)
• Objective function:
• Result:
  – Ex1: \(x_2 = 1, x_4 = 1\)
  – Ex2: \(x_1 = 1, x_4 = 1\)
Covering problem

• Set covering problem:
  – A set S.
  – A collection C of subsets.
  – Select fewest elements of C to cover S.

• Intractable.

• Exact method:
  – Branch and bound algorithm.

• Heuristic methods.
Example
vertex-cover of a graph

(a) 1 2 3 4 5

(b) 1 2 3 4 5

(c) 1 2 3 4 5

(d) 1 2 3 4 5

3 4 5

1 2 3 4 5

3 4 5

1 2 3 4 5
Example
edge-cover of a hypergraph
Matrix representation

- Boolean matrix: \textbf{A}.
- Selection Boolean vector: \textbf{x}.
- Determine \textbf{x} such that:
  - \( \textbf{A} \textbf{x} \geq 1 \).
  - Select enough columns to cover all rows.
- Minimize cardinality of \textbf{x}.

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
1 \\
1 \\
0 \\
1 \\
1 \\
\end{bmatrix}
= 
\begin{bmatrix}
1 \\
2 \\
1 \\
1 \\
1 \\
\end{bmatrix}
\]
Branch and bound algorithm

- Tree search of the solution space:
  - Potentially exponential search.

- Use bounding function:
  - If the lower bound on the solution cost that can be derived from a set of future choices exceeds the cost of the best solution seen so far:
    - Kill the search.

- Good pruning may reduce run-time.
BRANCH_AND_BOUND{
Current best = anything;
Current cost = 1;
S= s_0 ;

while (S6 = ; ) do {
    Select an element in s 2S;
    Remove s from S ;
    Make a branching decision based on s
    yielding sequences f s_i ; i= 1; 2; ...; mg ;
    for ( i = 1 to m) {
        Compute the lower bound b_i of s_i ;
        if ( b_i Current cost) Kill s_i ;
        else {
            if ( s_i is a complete solution ) {
                Current best = s_i ,
                Current cost = cost of s_i ,
            }
        }
    }
    else
    Add s_i to set S;
}
}
}
Example

(a) 

(b) 

Bound = 6

Killed subtree