Predicate Detection to Solve Combinatorial Optimization Problems

Vijay K. Garg

Parallel and Distributed Systems Lab,
Department of Electrical and Computer Engineering,
The University of Texas at Austin.

Motivation

Consider the following problems:

- Shortest Path Problem:
  Input: a weighted directed graph and a source vertex
  Output: Least Cost of reaching any vertex i
  Dijkstra's algorithm for graph with non-negative weights,
  Bellman-Ford algorithm for graphs with no negative cycles
- Stable Marriage Problem:
  Input: ordered preferences of n men and n women
  Output: Man-optimal stable marriage
  Gale-Shapley's algorithm
- Assignment Problem:
  Input: n items, n bidders with valuation for items
  Output: Least market clearing prices
  Hungarian Algorithm (or Gale-Demange-Sotomayor's Auction)

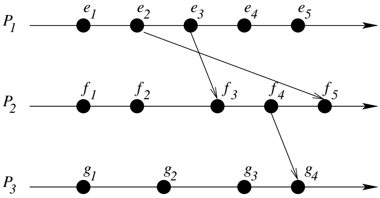
Could there be a single algorithm that solves all of these problems?
Lattice-Linear Predicate (LLP) Algorithm

Steps of Using LLP Algorithm

- Step 1: Model the underlying search space. A Distributive Lattice of State Vectors. The order on the lattice is based on the optimization objective of the problem.
- Step 2: Define the feasibility predicate B. An element is feasible if it satisfies constraints of the problem
- Step 3: Check whether the feasibility predicate B is Lattice-Linear. If B is lattice-linear, LLP Algorithm will return the optimal feasible solution.

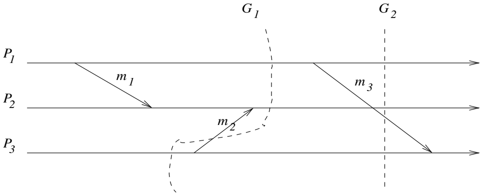
Step 1: Modeling the underlying search space

Model the problem as n processes choosing their component in a vector of size n. The choice for a single process is total ordered.



computation: poset $(E, \rightarrow)$
candidate solution: a possible global state of the system.

Consistent Global State



A subset G of E is a consistent global state if

$$\forall e, f \in E : (f \in G) \wedge (e \rightarrow f) \Rightarrow (e \in G)$$

The set of all consistent global states forms a finite distributive lattice.
The order is component-wise comparison.

Step 1: Examples

G: Global State Vector where $G[i]$ is the component for process i.

- Shortest Path: $G[i]$: cost of reaching vertex i from the source vertex
  initially 0
- Stable Marriage: $G[i]$: index in the preference list for man i
  initially 1 // top choice
- Market Clearing Prices: $G[i]$: price of item i
  initially 0

Step 2: Defining Feasibility Predicate

- Shortest Path: Every non-source node has a parent. For any node $j \neq 0$,

$$\exists i \in pre(j) : G[j] \geq G[i] + w[i,j]$$

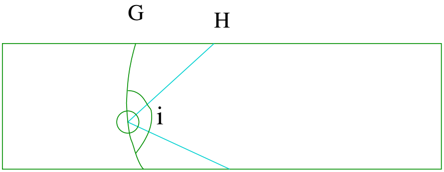
- Stable Marriage: Every man must be matched to a different woman and there must not be any blocking pair. For any man j, let $z = mpref[j][G[j]]$; //current woman assigned to man j

$$\neg\exists i : \exists k \leq G[i] : (z = mpref[i][k]) \wedge (rank[z][i] < rank[z][j]))$$

- Market Clearing Prices: There is no overdemanded item at that pricing vector. For any item j,

$$\neg\exists J : minimalOverDemanded(J, G) \wedge (j \in J)$$

Lattice-Linearity for Predicate Detection



Forbidden State The state at P_i is forbidden at G with respect to B if unless P_i is advanced B cannot become true.

$$\text{forbidden}(G, i, B) \equiv \forall H : G \subseteq H : (G[i] = H[i]) \Rightarrow \neg B(H)$$

Lattice-Linear Predicates A predicate B is lattice-linear if for all consistent cuts G,

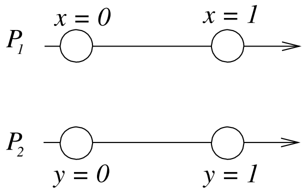
$$\neg B(G) \Rightarrow \exists i : \text{forbidden}(G, i, B).$$

Examples of Lattice-Linear Predicates

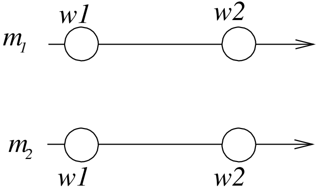
- A conjunctive predicate
  $l_1 \wedge l_2 \wedge \ldots \wedge l_n$, where l_i is local to P_i.
  Suppose G is not feasible. Then, there exists j such that l_j is false in G. The index j is forbidden in G.
- Shortest Path
  Any j such that v_j does not have a parent,
  $(\forall i \in pre(j) : G[j] < G[i] + w[i, j])$ is forbidden in G.
- Stable Marriage
  j is forbidden in G if
  $\exists i : \exists k \leq G[i] : (z = mpref[i][k]) \wedge (rank[z][i] < rank[z][j]))$
- Market Clearing Price
  $(\neg\exists J : minimalOverDemanded(J, G) \wedge (j \in J))$
  Any j in a minimal overDemanded set is forbidden.

Example of Predicates that are not Lattice-Linear

Example 1: $B(G) \equiv x + y \geq 1$



Example 2: $B(G) \equiv G$ is a matching.



LLP Algorithm

How much to advance: j is forbidden in G until α iff

$$\forall H \in L : H \geq G : (H[j] < \alpha) \Rightarrow \neg B(H).$$

```
vector function getLeastFeasible(T: vector, B: predicate)
//T: top element of the lattice
 var G: vector of reals initially ∀i : G[i] = 0;
 while ∃j: forbidden(G, j, B)  do
      for all j such that forbidden(G, j, B) in parallel:
           if (α(G, j, B) > T[j]) then return null;
           else G[j] := α(G, j, B);
 endwhile;
 return G; // the optimal solution
```

All processes can asynchronously evaluate forbidden and advance in parallel. Only P_j updates $G[j]$.

LLP Algorithm: Stable Marriage Problem

P_j:

var G: array[1..n] of 1..n;

input: $mpref[i,k]$: int for all i,k; // men preferences
$rank[k][i]$: int for all k,i; // women ranking
init: $G[j] := 1$;
always: $w = mpref[j][G[j]]$;

forbidden:
$(\exists i : \exists k \leq G[i] : (w = mpref[i][k]) \wedge (rank[w][i] < rank[w][j]))$
advance: $G[j] := G[j] + 1$;

Slightly more general than Gale-Shapley Algorithm:
instead of starting from $(1, 1, \ldots, 1)$, can start from any choice vector.

LLP Algorithm: Shortest Path Problem

input: $pre(j)$: list of $1..n$;
$w[i,j]$: positive int for all $i \in pre(j)$
$s : 1..n$; // source node;
init: $G[j] := 0$;
always:
$parent[j,i] = (i \in pre(j)) \wedge (G[j] \geq G[i] + w[i,j])$;
$fixed[j] = (j = s) \vee (\exists i : parent[j,i] \wedge fixed[i])$
$Q = \{(G[i] + w[i,k]) | (i \in pre(k)) \wedge fixed(i) \wedge \neg fixed(k)\}$;

forbidden: $\neg fixed[j]$
advance: $G[j] := \max\{\min Q, \min\{G[i] + w[i,j] \mid i \in pre(j)\}\}$

By ignoring the second part of advance, we can get Dijkstra's algorithm.

LLP Algorithm: Shortest Path Problem Revisited

Assume no negative cost cycle.

input: $pre(j)$: list of $1..n$;
  $w[i,j]$: int for all $i \in pre(j)$
init: if $(j = s)$ then $G[j] := 0$ else $G[j] :=$ maxint;
forbidden: $G[j] > \min\{G[i] + w[i,j] \mid i \in pre(j)\}$
advance: $G[j] := \min\{G[i] + w[i,j] \mid i \in pre(j)\}$

Lattice is reversed: the bottom element is $(maxint, maxint, \ldots, maxint)$
This is just Bellman-Ford's algorithm.

LLP Algorithm: Market Clearing Prices

input: $v[b, i]$: int for all b, i
init: $G[j] := 0$;
always: $E = \{(k, b) \mid \forall i : (v[b, k] - G[k]) \geq (v[b, i] - G[i])\}$;
$demand(U') = \{k \mid \exists b \in U' : (k, b) \in E\}$;
$overDemanded(J) \equiv \exists U' \subseteq U : (demand(U') = J) \wedge (|J| < |U'|)$

forbidden: $\exists J : minimal - OverDemanded(J) \wedge (j \in J)$
advance: $G[j] := G[j] + 1$;

This is just Demange-Gale-Sotomayor exact auction algorithm.

Constrained Optimization

If B_1 and B_2 are lattice-linear then $B_1 \wedge B_2$ is also lattice-linear.

- least stable marriage such that regret of Peter is less than or equal to regret of John
- least feasible path such that the cost of reaching x equals cost of reaching y
- least clearing prices such that $item_1$ is priced at least 5 more than $item_2$.

All of the additional constraints are also lattice-linear.

Conclusions

How to Solve Many Combinatorial Optimization Problems
Find the least feasible element

- View State space as the set of consistent global states
- Each process starts with the most desirable choice and moves to less desirable
- Define a "feasibility" predicate B
- Check if B satisfies the lattice-linearity condition

Other algorithms as special cases of the LLP Algorithm:

- Gale's Top Trading Cycle Algorithm,
- Horn's satisfiability algorithm,
- Johnson's algorithm to transform graphs with negative cost edges

Future Work

- Techniques when the feasibility predicate is not lattice-linear.