Predicate Detection to Solve Combinatorial Optimization Problems

Vijay K. Garg
The University of Texas at Austin
Austin, Texas, USA
garg@ece.utexas.edu

ABSTRACT

We present a method to design parallel algorithms for constrained combinatorial optimization problems. Our method solves and generalizes many classical combinatorial optimization problems including the stable marriage problem, the shortest path problem and the market clearing price problem. These three problems are solved in the literature using Gale-Shapley algorithm, Dijkstra's algorithm, and Demange, Gale, Sotomayor algorithm. Our method solves all these problems by casting them as searching for an element that satisfies an appropriate predicate in a distributive lattice. Moreover, it solves generalizations of all these problems — namely finding the optimal solution satisfying additional constraints called *lattice-linear* predicates. For stable marriage problems, an example of such a constraint is that Peter's regret is less than that of Paul. For shortest path problems, an example of such a constraint is that cost of reaching vertex v_1 is at least the cost of reaching vertex v_2. For the market clearing price problem, an example of such a constraint is that $item_1$ is priced at least as much as $item_2$. Our algorithm, called Lattice-Linear Predicate Detection (LLP) can be implemented in parallel without any locks or compare-and-set instructions. It just assumes atomicity of reads and writes.

CCS CONCEPTS

• **Theory of computation → Parallel algorithms**;

KEYWORDS

distributive lattices; predicate detection; optimization problems

ACM Reference Format:
Vijay K. Garg. 2020. Predicate Detection to Solve Combinatorial Optimization Problems. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '20), July 15–17, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3350755.3400235

1 INTRODUCTION

We present a method called *lattice-linear predicate detection* that can solve many combinatorial optimization problems. We use this method to solve generalizations of three of the most fundamental problems in combinatorial optimization — the stable marriage problem [11], the shortest path problem [9], and the assignment problem [18]. The classical algorithms to solve these problems are Gale-Shapley algorithm for the stable marriage problem [11], Dijkstra's algorithm for the shortest path problem [9], and Kuhn's Hungarian method to solve the assignment problem [18] (or equivalently, Demange-Gale-Sotomayor auction-based algorithm [7] for market clearing prices). Could there be a single efficient parallel algorithm that solves all of these problems?

In this paper, we describe a technique that solves not only these problems but more *general* versions of each of the above problems. We seek the optimal solution for these problems that satisfy additional constraints modeled using a *lattice-linear* predicate [4]. When the set of constraints is empty, we get back the classical problems. Our technique requires the underlying search space to be viewed as a distributive lattice [3, 6, 12]. Common to all these seemingly disparate combinatorial optimization problems is the structure of the *feasible* solution space. The set of all stable matchings, the set of all feasible rooted trees for the shortest path problem, and the set of all market clearing prices are all closed under the meet operation of the lattice. If the order is appropriately defined, then finding the optimal solution (the man-optimal stable marriage, the shortest path cost vector, the minimum market clearing price vector) is equivalent to finding the infimum of all feasible solutions in the lattice.

We note here that it is well-known that the set of stable matching and the set of market clearing price vectors form distributive lattices. The claim that the set of stable matchings forms a distributive lattice is given in [16] where this observation is attributed to Conway. The set of market clearing price vectors forms a distributive lattice is given in [19]. However, the algorithms to find the man-optimal stable matching and the minimum market clearing price vectors are not derived from the lattice property. In our method, once the lattice-linearity of the feasible solution space is established, the algorithm to find the optimal solution falls out as a consequence. To the best of our knowledge, this is the first paper to derive Gale-Shapley's algorithm, Dijkstra's algorithm and Demange-Gale-Sotomayor's algorithm from a single algorithm by exploiting a lattice property. In fact, we derive parallel version of all these algorithms. Our algorithms do not use any synchronization (locks, compare-and-sets, or barriers) assuming read-write atomicity of memory locations.

The lattice-linear predicate detection method to solve the combinatorial optimization problem is as follows. The first step is to define a lattice of vectors L such that each vector is *assigned* a point in the search space. For the stable matching problem, the vector corresponds to the assignment of men to women (or equivalently, the choice number for each man). The second step in our method is to define a boolean predicate B that models feasibility of the

SPAA '20, July 15–17, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-6935-0/20/07…$15.00
https://doi.org/10.1145/3350755.3400235

vector. For the stable matching problem, an assignment is feasible iff it is a matching and there is no blocking pair. The third step is to show that the feasibility predicate is a lattice-linear predicate [4]. Lattice-linearity allows one to search for a feasible solution efficiently. If any point in the search space is not feasible, it allows one to make progress towards the optimal feasible solution without any need for exploring multiple paths in the lattice. Moreover, multiple processes can make progress towards the feasible solution independently. This property of lattice-linearity allows the search algorithm to be parallel.

In summary, this paper makes the following contributions to the constrained combinatorial optimization problem. First, we present a unifying framework and the lattice-linear predicate detection algorithm (LLP) for such problems. By applying the lattice-linear predicate detection algorithm to unconstrained problems, we get parallel versions of Gale-Shapley algorithm for the stable matching problem, Dijkstra's algorithm and Bellman-Ford algorithm [1, 10] for the shortest path problem and Demange, Gale, Sotomayor's algorithm for the minimum market clearing price.

Note that our technique does not necessarily lead to the most-efficient parallel algorithm. Additional techniques may be necessary to optimize the algorithm further. It does provide a unifying framework for analyzing a large class of algorithms. In a rough sense, our framework is similar to Linear Programming [2] formulations. Most problems discussed in the paper can also be formulated as (integer) linear programs but those formulations may not result in most efficient algorithms.

Second, we get solutions for the constrained version of each of these problems, whenever the constraints are lattice-linear. We solve the *Constrained Stable Matching Problem* where in addition to men's preferences and women's preferences, there may be a set of lattice-linear constraints. For example, we may require that Peter's regret [14] should be less than that of Paul, where the *regret* of a man in a matching is the choice number he is assigned. We note here that some special cases of the constrained stable marriage problems have been studied. Dias et al [5, 8] study the stable marriage problem with restricted pairs. A restricted pair is either a *forced* pair which is required to be in the matching, or a *forbidden* pair which must not be in the matching. Both of these constraints are *lattice-linear* and therefore can be modeled in our system.

Third, by applying a constructive version of Birkhoff's theorem on finite distributive lattices [3, 6, 12, 17], we give an algorithm that outputs a succinct representation of all feasible solutions. In particular, the join-irreducible elements [6] of the feasible sublattice can be determined efficiently (in polynomial time). For the constrained stable matching problem, we get a concise representation of all stable matchings that satisfy given constraints. Thus, our method yields a more general version of rotation posets [14] to represent all *constrained* stable matchings. Analogously, we get a concise representation of all constrained integral market clearing price vectors.

The paper is organized as follows. Section 2 gives the definition of Lattice-Linear predicates and the LLP algorithm for detecting such predicates. The LLP Algorithm is a parallel algorithm that can be applied to all the problems mentioned earlier. Section 3 shows lattice-linearity of stability in the stable matching and gives a parallel algorithm for the constrained stable matching problem. Gale-Shapley algorithm can be viewed as a special case of this algorithm. Section 4 shows how Bellman-Ford, Dijkstra and Johnson's algorithm [15] can be viewed as special cases of LLP algorithm. Section 6 shows how Gale-Demange-Sotomayor algorithm for the assignment problem can be viewed as a special case of LLP algorithm. Section 8 compares this work with the related work. Finally, Section 9 presents conclusions and future work.

2 LATTICE-LINEAR PREDICATES

Let L be the lattice of all n-dimensional vectors of reals greater than or equal to zero vector and less than or equal to a given vector T where the order on the vectors is defined by the component-wise natural $\leq$. The lattice is used to model the search space of the combinatorial optimization problem. The combinatorial optimization problem is modeled as finding the minimum element in L that satisfies a boolean *predicate* B, where B models *feasible* (or acceptable solutions). We are interested in parallel algorithms to solve the combinatorial optimization problem with n processes. We will assume that the system maintains as its state the current candidate vector $G \in L$ in the search lattice, where $G[i]$ is maintained at process i. We call G, the global state, and $G[i]$, the state of process i.

Finding an element in a lattice that satisfies the given predicate B, is called the *predicate detection* problem. Finding the *minimum* element that satisfies B (whenever it exists) is the combinatorial optimization problem. We now define *lattice-linearity* which enables efficient computation of this minimum element. Lattice-linearity is first defined in [4] in the context of detecting global conditions in a distributed system where it is simply called linearity. We use the term *lattice-linearity* to avoid confusion with the standard usage of linearity.

A key concept in deriving an efficient predicate detection algorithm is that of a *forbidden* state. Given a predicate B, and a vector $G \in L$, a state $G[i]$ is *forbidden* (or equivalently, the index i is forbidden) if for any vector $H \in L$, where $G \leq H$, if $H[i]$ equals $G[i]$, then B is false for H. Formally,

Definition 2.1 (Forbidden State [4]). Given any distributive lattice L of n-dimensional vectors of $\mathbf{R}_{\geq 0}$, and a predicate B, we define $\text{forbidden}(G, i, B) \equiv \forall H \in L : G \leq H : (G[i] = H[i]) \Rightarrow \neg B(H)$.

We define a predicate B to be *lattice-linear* with respect to a lattice L if for any global state G, B is false in G implies that G contains a *forbidden state*. Formally,

Definition 2.2 (lattice-linear Predicate [4]). A boolean predicate B is *lattice-linear* with respect to a lattice L iff $\forall G \in L : \neg B(G) \Rightarrow (\exists i : \text{forbidden}(G, i, B))$.

We now give some examples of lattice-linear predicates.

(1) **Job Scheduling Problem**: Our first example relates to scheduling of n jobs. Each job j requires time t_j for completion and has a set of prerequisite jobs, denoted by $pre(j)$, such that it can be started only after all its prerequisite jobs have been completed. Our goal is to find the minimum completion time for each job. We let our lattice L be the set of all possible completion times. A completion vector $G \in L$ is feasible iff $B_{jobs}(G)$ holds where $B_{jobs}(G) \equiv \forall j : (G[j] \geq t_j) \wedge (\forall i \in pre(j) : G[j] \geq G[i] + t_j)$. B_{jobs} is lattice-linear because if it is false, then there exists j such that either

$G[j] < t_j$ or $\exists i \in pre(j) : G[j] < G[i] + t_j$. We claim that forbidden(G, i, B_{jobs}). Indeed, any vector $H \geq G$ cannot be feasible with $G[j]$ equal to $H[j]$. The minimum of all vectors that satisfy feasibility corresponds to the minimum completion time.

(2) **Shortest Path Problem**: We are given a weighted directed graph and a fixed vertex s. We are required to find the cost of the shortest path from s to all vertices. Let the input be specified as $w[i, j]$ as the cost of going from i to j. Here our objective is to output maximum $G[j]$ subject to constraints that $G[j]$ is less than or equal to $G[i] + w[i, j]$ for all $i \in pre(j)$. One can view $G[j]$ as an upper bound on the cost of reaching j. We assume that there are no negative cycles and thus $G[s]$ equals zero. For this problem, the order on the underlying lattice is inverted. The lattice is defined on the value of $G[j]$ for all vertices except the source vertex. The minimum element is the vector with all components as ∞. It is easy to check that the predicate $G[j] \leq \min\{G[i] + w[i, j] \mid i \in pre(j)\}$ is lattice-linear. If $G[j] > G[i] + w[i, j]$ for some (i, j) then it will continue to hold until G is advanced on j, i.e., the value of $G[j]$ is reduced at least to $G[i] + w[i, j]$.

(3) **Continuous Optimization Problem**: We are required to find minimum nonnegative x and y such that $B \equiv (x \geq 2y^2 + 5) \wedge (y \geq x - 4)$. We view this problem as finding minimum (x, y) pair such that B holds. It is easy to verify that B is lattice-linear. If the first conjunct is false, then x is forbidden. Unless x is increased the predicate cannot become true, even if other variables (y for this example) increase. If the second conjunct is false, then y is forbidden.

(4) **A Non Lattice-Linear Predicate** As an example of a predicate that is not lattice-linear, consider the predicate $B \equiv \sum_j G[j] \geq 1$ defined on the space of two dimensional vectors. Consider the vector G equal to $(0, 0)$. The vector G does not satisfy B. For B to be lattice-linear either the first index or the second index should be forbidden. However, none of the indices are forbidden in $(0, 0)$. The index 0 is not forbidden because the vector $H = (0, 1)$ is greater than G, has $H[0]$ equal to $G[0]$ but it still satisfies B. The index 1 is also not forbidden because $H = (1, 0)$ is greater than G, has $H[1]$ equal to $G[1]$ but it satisfies B.

The following Lemma is useful in proving lattice-linearity of predicates.

Lemma 2.3. *Let B be any boolean predicate defined on a lattice L of vectors.*
(a) Let $f : L \rightarrow \mathbf{R}_{\geq 0}$ be any monotone function defined on the lattice L of vectors of $\mathbf{R}_{\geq 0}$. Consider the predicate $B \equiv G[i] \geq f(G)$ for some fixed i. Then, B is lattice-linear.
(b) Let L_B be the subset of the lattice L of the elements that satisfy B. Then, B is lattice-linear iff L_B is closed under meets.
(c) If B_1 and B_2 are lattice-linear then $B_1 \wedge B_2$ is also lattice-linear.

Proof. (a) Suppose B is false for G. This implies that $G[i] < f(G)$. Consider any vector $H \geq G$ such that $H[i]$ is equal to $G[i]$. Since $G[i] < f(G)$, we get that $H[i] < f(G)$. The monotonicity of f implies that $H[i] < f(H)$ which shows that $\neg B(H)$.
(b) This is shown in [4]. Assume that B is not lattice-linear. This implies that there exists a global state G such that $\neg B(G)$, and $\forall i : \exists H_i \geq G : (G[i] = H_i[i])$ and $B(H_i)$. Consider $Y = \cup_i \{H_i\}$. All elements of $Y \in L_B$. However, $inf\ Y$ which is equal to G is not an element of L_B. This implies that L_B is not closed under the meet operation. Conversely, let $Y = \{H_1, H_2, \ldots, H_k\}$ be any subset of L_B such that its meet G does not belong to L_B. Since G is the meet of Y, for any i, there exists $j \in \{1 \ldots k\}$ such that $G[i] = H_j[i]$. Since $B(H_j)$ is true for all j, it follows that there exists a G for which lattice-linearity does not hold.
(c) Follows from the equivalence of meet-closed predicates with lattice-linearity and that meet-closed predicates are closed under conjunction. For a more direct proof, suppose that $\neg(B_1 \wedge B_2)$. This implies that one of the conjuncts is false and therefore from the lattice-linearity of that conjunct, a forbidden state exists. □

For the job scheduling example, we can define B_j as $G[j] \geq max(t_j, max\{G[i]+t_j \mid i \in pre(j)\})$. Since $f_j(G) = max(t_j, max\{G[i]+t_j \mid i \in pre(j)\})$ is a monotone function, it follows from Lemma 2.3(a) that B_j is lattice-linear. The predicate $B_{jobs} \equiv \forall j : B_j$ is lattice-linear due to Lemma 2.3(c). Also note that the problem of finding the minimum vector that satisfies B_{jobs} is well-defined due to Lemma 2.3(b).

We now discuss detection of lattice-linear predicates which requires an additional assumption called the *efficient advancement property* [4] — there exists an efficient (polynomial time) algorithm to determine the forbidden state. This property holds for all the problems considered in this paper. Once we determine j such that $forbidden(G, j, B)$, we also need to determine how to advance along index j. To that end, we extend the definition of forbidden as follows.

Definition 2.4 (α-forbidden). Let B be any boolean predicate on the lattice L of all assignment vectors. For any G, j and a positive real $\alpha > G[j]$, we define forbidden(G, j, B, α) iff $\forall H \in L : H \geq G : (H[j] < \alpha) \Rightarrow \neg B(H)$.

Given any lattice-linear predicate B, suppose $\neg B(G)$. This means that G must be advanced on all indices j such that forbidden(G, j, B). We use a function $\alpha(G, j, B)$ such that forbidden$(G, j, B, \alpha(G, j, B))$ holds whenever forbidden(G, j, B) is true. With the notion of $\alpha(G, j, B)$, we have the algorithm *LLP* shown in Fig. 1. The algorithm *LLP* has two inputs — the predicate B and the top element of the lattice T. It returns the least vector G which is less than or equal to T and satisfies B (if it exists). Whenever B is not true in the current vector G, the algorithm advances on all forbidden indices j in parallel. This simple parallel algorithm can be used to solve a large variety of combinatorial optimization problems by instantiating different forbidden(G, j, B) and $\alpha(G, j, B)$.

We note here that *LLP* algorithm has a single variable G. Suppose that we maintain $G[j]$ on a separate thread for all j. Then, all these threads can evaluate $forbidden(G, j, B)$ in parallel and also advance $G[j]$ to $\alpha(G, j, B)$ in parallel. Since only thread j can update $G[j]$, there cannot be any write-write conflict. Moreover, assuming that we have linearizable reads and writes, if one thread j is updating $G[j]$ and thread i reads the value of $G[j]$ and gets the old value of $G[j]$, the algorithm continues to behave correctly. In other words, LLP algorithm does not require any locks. We do assume that any update to G is eventually visible to all threads and that the algorithm terminates only when G does not have any forbidden component.

```
vector function getLeastFeasible(T: vector, B: predicate)
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
```

Figure 1: Parallel Algorithm *LLP* to find the minimum vector less than or equal to T that satisfies B

Theorem 2.5. *Suppose there exists a fixed constant $\delta > 0$ such that $\alpha(G, j, B) - G[j] \geq \delta$ whenever* forbidden(G, j, B). *Then, the parallel algorithm LLP finds the least vector $G \leq T$ that satisfies B, if one exists.*

Proof. Since $G[j]$ increases by at least δ for at least one forbidden j in every iteration of the *while* loop, the algorithm terminates in at most $\sum_i \lceil T[i]/\delta \rceil$ number of steps.

We show that the algorithm maintains the invariant $(I1)$ that for all indices j, any vector V such that $V[j]$ is less than $G[j]$ cannot satisfy B. Formally, the invariant $(I1)$ is

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

Initially, the invariant holds trivially because G is initialized to 0. Suppose forbidden(G, j, B). Then, we increase $G[j]$ to $\alpha(G, j, B)$. We need to show that this change maintains the invariant. Pick any V such that $V[j] < \alpha(G, j, B)$. We now do a case analysis. If $V \geq G$, then $\neg B(V)$ holds from the definition of $\alpha(G, j, B)$. Otherwise, there exists some k such that $V[k] < G[k]$. In this case $\neg B(V)$ holds due to $(I1)$.

We now show Theorem 2.5 using the invariant. First, suppose that the algorithm *LLP* terminates because $\alpha(G, j, B) > T[j]$. In this case, there is no feasible vector in L due to the invariant (because the predicate B is false for all values of $G[j]$). Now suppose that the algorithm terminates because there does not exist any j such that forbidden(G, j, B). This implies that G satisfies B due to lattice-linearity of B. It is also the least vector that satisfies B due to the invariant $(I1)$. □

2.1 Simple Examples

We now derive parallel algorithms for some simple examples.

(1) *Job Scheduling*: For the job scheduling example, we get a parallel algorithm to find the minimum completion time by using forbidden$(G, j, B_{jobs}) \equiv (G[j] < t_j) \vee (\exists i \in pre(j) : G[j] < G[i] + t_j)$, and $\alpha(G, j, B_{jobs}) = \max\{t_j, \max\{G[i] + t_j | i \in pre(j)\}\}$.
The resulting algorithm is shown in Fig. 2. Instead of initializing $G[j]$ with 0, we initialize it with t_j to simplify the forbidden predicate. The program returns the least G that satisfies the predicate B_{jobs}. Any j that does satisfies $forbidden(j)$ triggers the **advance** section. Each thread can evaluate whether it is forbidden (possibly using stale values of other components) in parallel and then advance itself. Computation of the forbidden predicate at node j takes time proportional to the size of $pre(j)$. If all threads evaluate $forbidden$ in parallel, the number of iterations required equals the length of the critical path in the prerequisite graph.

(2) *Shortest Path Problem: Parallel Bellman-Ford*: For this problem our goal is to maximize $G[j]$ subject to constraints that $G[j]$ is less than or equal to $G[i] + w[i, j]$ for all $i \in pre(j)$. The variable $G[i]$ is initialized to ∞ for all indices except for the source vertex which is initialized to 0. Since the predicate $G[j] \leq \min\{G[i] + w[i, j] \mid i \in pre(j)\}$ is lattice-linear, the program returns the optimal cost vector. Each thread evaluates whether it is forbidden in G in parallel and advances in a manner similar to the job scheduling problem.
In this example, $G[j]$ corresponds to an upper-bound on the cost of the shortest path from the source to node j. This is the standard method of edge-relaxation algorithms for shortest paths. In section 4, we discuss LLP algorithms based on maintaining lower bounds instead of the upper bounds.

```
P_j: Code for thread j
// common declaration for all the programs below
shared var G: array[1..n] of 0..maxint;
job-scheduling:
    input: t[j] : int, pre(j): list of 1..n;
    init: G[j] := t[j];
    forbidden: G[j] < max{G[i] + t[j] | i ∈ pre(j)};
    advance: G[j] := max{G[i] + t[j] | i ∈ pre(j)};

shortest path from node s: Parallel Bellman-Ford
    input: pre(j): list of 1..n; w[i, j]: int for all i ∈ pre(j)
    init: if (j = s) then G[j] := 0 else G[j] := maxint;
    forbidden: G[j] > min{G[i] + w[i, j] | i ∈ pre(j)}
    advance: G[j] := min{G[i] + w[i, j] | i ∈ pre(j)}
```

Figure 2: LLP Parallel Program for the job scheduling and the shortest path problems

We now show, on account of Lemma 2.3(c), that if we have a parallel algorithm for a problem, then we also have one for the constrained version of that problem.

Lemma 2.6. *Let LLP be the parallel algorithm to find the least vector G that satisfies B_1 if one exists. Then, LLP can be adapted to find the least vector G that satisfies $B_1 \wedge B_2$ for any lattice-linear predicate B_2.*

Proof. The algorithm *LLP* can be used with the following changes: forbidden$(G, j, B_1 \wedge B_2) \equiv$ forbidden$(G, j, B_1) \vee$ forbidden(G, j, B_2), and $\alpha(G, j, B_1 \wedge B_2) = \max\{\alpha(G, j, B_1), \alpha(G, j, B_2)\}$.

□

For example, suppose that we want the minimum completion time of jobs with the additional lattice-linear constraint that $B_2(G) \equiv (G[1] = G[2])$. B_2 is lattice-linear with forbidden$(G, 1, B_2) \equiv (G[1] < G[2])$ and forbidden$(G, 2, B_2) \equiv (G[2] < G[1])$. By applying, Lemma 2.6, we get a parallel algorithm for the constrained version.

3 CONSTRAINED STABLE MATCHING PROBLEM

In this problem, we are given as input n men and n women. We are also given a list of men preferences as $mpref$ where $mpref[i][k]$ denotes k^{th} top choice of man i. The women preferences are more convenient to express as a $rank$ array where $rank[i][j]$ is the rank of man j by woman i. A matching between man and woman is stable if there is no *blocking pair*, i.e., a pair of woman and man such that they are not matched and prefer each other to their spouses.

The underlying lattice for this example is the set of all n dimensional vectors of $1..n$. We let $G[i]$ be the choice number that man i has proposed to. Initially, $G[i]$ is 1 for all men. For convenience, let $\rho(G, i)$ denote the woman $mpref[i][G[i]]$.

Definition 3.1. An assignment G is feasible for the stable marriage problem if (1) it corresponds to a perfect matching (all men are paired with different women) and (2) it has no blocking pairs.

We show that the predicate "G is a stable marriage" is a lattice-linear predicate.

Lemma 3.2. *The predicate that a vector G corresponds to a stable marriage is lattice-linear.*

Proof. Let z be $\rho(G, j)$, the woman that corresponds to choice $G[j]$ for man j. We define j to be forbidden in G if there exists a man i such that z prefers man i to man j and either man i has also been assigned z in G or he prefers z to his current choice, i.e., man i and woman z would form a blocking pair in G. Formally, $forbidden(G, j)$ is defined as $(\exists i : \exists k \leq G[i] : (z = mpref[i][k]) \wedge (rank[z][i] < rank[z][j]))$.

It is easy to see that G is not a stable marriage iff $\exists j : forbidden(G, j)$. If G is not a perfect matching then there must be at least one woman who is assigned to two men. In that case, the less preferred man is forbidden. If G is a perfect matching but has a blocking pair, then the partner of the woman in the blocking pair is forbidden. Conversely, $forbidden(G, j)$ implies that either G is not a perfect matching or has a blocking pair.

We only need to show that if $forbidden(G, j)$ holds, then there is no proposal vector H such that $(H \geq G)$ and $(G[j] = H[j])$ and H is a stable marriage.

Consider any H such that $(H \geq G)$ and $(G[j] = H[j])$. We show that H is not a stable marriage. Since $G[j]$ is equal to $H[j]$, $\rho(G, j)$ is equal to $\rho(H, j)$. Let i be such that $\exists k \leq G[i] : (z = mpref[i][k]) \wedge (rank[z][i] < rank[z][j]))$. Since $G \leq H$, $G[i] \leq H[i]$, we get that $\exists k \leq H[i] : (z = mpref[i][k]) \wedge (rank[z][i] < rank[z][j]))$. Hence, $forbidden(H, i)$ also holds.

□

The parallel LLP algorithm is shown in Fig. 3. The **always** section defines variables which are derived from G. These variables can be viewed as macros. For example, in the stable marriage problem, for any thread $z = mpref[j][G[j]]$. This means that whenever $G[j]$ changes, so does z (just like a formula in a spreadsheet).

If man j is forbidden, it is clear that any vector in which man j is matched with z and man i is matched with his current or a worse choice can never be a stable marriage. Thus, it is safe for man j to advance to the next choice.

P_j: Code for thread j
Man-optimal stable marriage

input: $mpref[i, k]$: int for all i, k; $rank[k][i]$: int for all k, i;
init: $G[j] := 1$;
always: $z = mpref[j][G[j]]$;

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

Figure 3: A Parallel LLP Algorithm for Stable Matching

The LLP algorithm has a single variable G. Any thread j (simulating code for man j), can only change its own component $G[j]$ although it can read $G[i]$ for any i. When thread i is updating its own component, and thread j reads that component, we assume that it either gets the old value of $G[i]$ or the new value of $G[i]$. Assuming such read-write atomicity for a single entry of the array G, there is no lock required in the algorithm shown in Fig. 3. Thus, we have

Theorem 3.3. *Assuming read-write atomicity, there exists a parallel algorithm to solve the stable matching problem with n threads that does not use any synchronization.*

Observe that with n cores, the function $forbidden(G, j)$ can be computed in $O(1)$ time. Since each component can be advanced at most $O(n)$ times, LLP algorithm may take $O(n^2)$ time in the worst case (when exactly one man can advance in every time step). Our goal is not to come up with the most efficient parallel algorithm but a unifying framework for designing parallel algorithms.

Another subtle issue is the trigger for evaluation of forbidden predicates. If $forbidden(G, j)$ holds for some G, then unless $G[j]$ is advanced, it will continue to hold. However, if $forbidden(G, j)$ is false, it can become true when G advances on other components. One possibility is for thread j to signal all threads i to evaluate $forbidden(G, i)$ whenever $G[j]$ is advanced and $\rho(G, j)$ equals $\rho(G, i)$.

One can easily derive an efficient sequential LLP algorithm from the parallel LLP algorithm in Fig. 3. We can maintain the list of men that are forbidden at the current value of G. At each iteration, we remove a man from this list and advance him to his next choice. If advancing this man makes some other man forbidden, then he is added to the list. Doing this efficiently, requires us to maintain current partners for all women. The algorithm derived in this manner is identical to Gale-Shapley deferred acceptance algorithm.

3.1 Properties of the LLP Algorithm

We note here some useful properties of the LLP algorithm. These properties are applicable to all the problems in this paper.

(1) **Nondeterminism in Evaluation of Forbidden Predicate**: Given a global state G, there may be multiple indices j for which $G[j]$ is forbidden. The LLP algorithm is correct irrespective of the order in which these indices are updated. The efficiency of the algorithms may differ depending upon the order in which these indices are updated, but the correctness is independent of the order. In the stable marriage problem,

the final answer returned is independent of the order in which men propose.

(2) **Parallel Evaluation of Forbidden Predicate without locks**: Suppose that G is shared among different threads such that thread j is responsible for evaluating $forbidden(G, j)$. While this thread is evaluating this predicate other threads may have advanced on other indices, i.e., thread j may have old information of $G[i]$ for $i \neq j$. However, this would still keep the algorithm correct. In the stable marriage problem, men can propose to women in parallel. In the shortest path algorithm, multiple vertices can update the estimate $G[i]$ in parallel.

(3) **No Lookahead Required for evaluation of Forbidden Predicate**: The LLP algorithm determines whether an index j is forbidden depending upon only the current global state G (and the history). This means that these algorithms are applicable in online settings where the future part of the lattice is revealed only when a forbidden index needs to advance. In the stable marriage problem, when we are computing the man-optimal stable marriage, a man may not reveal his preference list. Only when he is rejected (his state is forbidden), he needs to advance on his choices and therefore reveal the next woman on his list.

3.2 Additional Constraints on Stable Matchings

We now present an algorithm to find stable marriages that satisfy additional constraints. Due to Lemma 2.6, we can use LLP algorithm to find the least stable marriage satisfying these constraints. The following lemma proves lattice-linearity of many such constraints.

Lemma 3.4. *The following constraints are lattice-linear.*

(1) *The regret of man i is at most that of the regret of man j.*
(2) *Man i cannot be married to woman j.*
(3) *The regret of man i is equal to that of man j.*

Proof. Let B be the predicate that G is a stable marriage and it satisfies the corresponding additional constraint.

(1) Suppose that G is a stable marriage but it does not satisfy B. This means that regret of man i is more than the regret of man j. In this case, we have $forbidden(G, j)$, because unless $G[j]$ is advanced, the predicate cannot become true.
(2) If $\rho(G, i) = j$, then $forbidden(G, i)$ holds.
(3) This condition is a conjunction of two lattice-linear conditions of type in part (1).

□

4 CONSTRAINED SINGLE SOURCE SHORTEST PATH ALGORITHM

Consider a weighted directed graph with n vertices numbered 0 to $n-1$. We assume that all edge weights are *strictly positive*. We are required to find the minimum cost of a path from a distinguished *source* vertex v_0 to all other vertices where the cost of a path is defined as the sum of edge weights along that path. For any vertex v, let $pre(v)$ be the set of vertices u such that (u, v) is an edge in the graph. To avoid trivialities, assume that every vertex v (except possibly the source vertex v_0) has nonempty $pre(v)$ and that all nodes in the graph are reachable from the source vertex.

As the first step of the predicate detection algorithm, we define the lattice for the search space. We assign to each vertex v_i, $G[i] \in \mathbf{R}_{\geq 0}$ with the interpretation that $G[i]$ is the cost of reaching vertex v_i. We call G, the *assignment* vector. The invariant maintained by our algorithm is: for all i, the cost of any path from v_0 to v_i is greater than or equal to $G[i]$. The vector G only gives the lower bound on the cost of a path and there may not be any path to vertex v_i with cost $G[i]$. To capture that an assignment is feasible, we define *feasibility* which requires the notion of a *parent*. We say that v_i is a parent of v_j in G (denoted by the predicate $parent(j, i, G)$) iff there is a direct edge from v_i to v_j and $G[j]$ is at least $(G[i] + w[i, j])$, i.e., $(i \in pre(j)) \wedge (G[j] \geq G[i] + w[i, j])$. A node may have multiple parents.

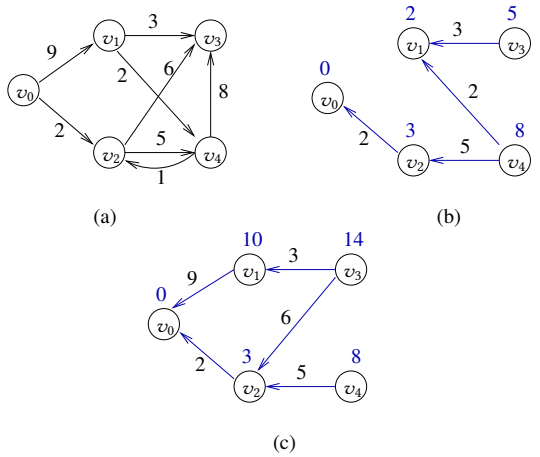

Figure 4: (a) A Weighted Directed Graph (b) The parent structure for $G = (0, 2, 3, 5, 8)$ (c) The parent structure for $G = (0, 10, 3, 14, 8)$. Since every non-source node has at least one parent, G is feasible.

In Fig. 4, let G be the vector $(0, 2, 3, 5, 8)$. Then, v_0 is a parent of v_2 because $G[2]$ is greater than $G[0]$ plus $w[0, 2](i.e., 3 \geq 0 + 2)$. Similarly, v_1 is a parent of v_4 because $G[4] \geq G[1] + 2$. A node may have multiple parents. The node v_2 is also a parent of v_4 because $G[4] \geq G[2] + 5$.

Since $w[i, j]$ are strictly positive, there cannot be a cycle in the parent relation. Now, feasibility can be defined as follows.

Definition 4.1 (Feasible for paths). An assignment G is *feasible for paths* iff every node except the source node has a parent. Formally, $B_{path}(G) \equiv \forall j \neq 0 : (\exists i : parent(j, i, G))$.

Hence, an assignment G is feasible iff one can go from any non-source node to the source node by following any of the parent edges. We now show that feasibility satisfies lattice-linearity.

Lemma 4.2. *For any assignment vector G that is not feasible, $\exists j$:* forbidden$(G, j, B_{path}(G))$.

Proof. Suppose G is not feasible. Then, there exists $j \neq 0$ such that v_j does not have a parent, i.e., $\forall i \in pre(j) : G[j] < G[i] + w[i, j]$. We show that forbidden$(G, j, B_{path}(G))$ holds. Pick any H such that $H \geq G$. Since for any $i \in pre(j)$, $H[i] \geq G[i]$, $G[j] < G[i] + w[i, j]$ implies that $G[j] < H[i] + w[i, j]$. Therefore, whenever $H[j] = G[j]$, v_j does not have a parent. □

Since B_{path} is a lattice-linear predicate, it follows from Lemma 2.3(c), that the set of feasible assignment vectors are closed under meets (the component-wise min operation). Hence, we can use LLP algorithm with $\alpha(G, j, B_{path}) = \min\{G[i] + w[i, j] \mid i \in pre(j)\}$.

For an unweighted graph (i.e., each edge has weight equal to 1), the above parallel algorithm requires time equal to the distance of the farthest node from the root. The LLP algorithm derived from B_{path}, however, may take time that depends on the weights because the advancement along a forbidden process may be small.

We now give an alternative feasible predicate that results in an algorithm that takes bigger steps. We first define a node j to be *fixed* in G if either it is the source node or it has a parent that is a fixed node, i.e., $fixed(j, G) \equiv (j = 0) \vee (\exists i : parent(j, i, G) \wedge fixed(i, G))$.

Observe that node v_0 is always fixed. Any node v_j such that one can reach from v_j to v_0 using parent relation is also fixed. We now define another feasible predicate called B_{rooted}, as $B_{rooted}(G) \equiv \forall j : fixed(j, G)$.

Even though it may first seem that the predicate B_{rooted} is strictly stronger than B_{path}, the following Lemma shows otherwise.

Lemma 4.3. *$B_{path}(G)$ iff $B_{rooted}(G)$.*

Proof. If G satisfies B_{rooted}, then every node other than v_0 has at least one parent by definition of $fixed$, hence $B_{path}(G)$. Conversely, suppose that every node except v_0 has a parent. Since parent edges cannot form a cycle, by following the parent edges, we can go from any node to v_0. □

It follows that the predicate B_{rooted} is also lattice-linear. Then, the following threshold $\beta(G)$ is well-defined whenever the set of edges from the fixed vertices to non-fixed vertices is nonempty.

$$\beta(G) = \min_{(i,j): i \in pre(j)} \{G[i] + w[i, j] \mid fixed(i, G), \neg fixed(j, G)\}.$$

If the set of such edges is empty then no non-fixed vertex is reachable from the source. We call these set of edges *Heap* (because we would need the minimum of this set for advancement).

We now have the following result in advancement of G.

Lemma 4.4. *Suppose $\neg B_{rooted}(G)$.*
Then, $\neg fixed(j, G) \Rightarrow$ forbidden$(G, j, B_{rooted}, \beta(G))$.

Proof. Consider any assignment vector H such that $H \geq G$ and $H[j] < \beta(G)$. We show that H is not B_{rooted}. In particular, we show that j is not fixed in H. Suppose j is fixed in H. This implies that there is a path W from v_0 to v_j such that all nodes in that path are fixed. Let the path be the sequence of vertices $w_0, w_1, \ldots w_{m-1}$, where $w_0 = v_0$ and $w_{m-1} = v_j$. Let $w_l = v_k$ be the first node in the path that is not fixed in G. Such a node exists because w_{m-1} is not fixed in G. Since w_0 is fixed, we know that $1 \leq l \leq m - 1$. The predecessor of w_l in that path, w_{l-1} is well-defined because $l \geq 1$. Let $w_{l-1} = v_i$.

We show that $H[k] \geq \beta(G)$ which contradicts $H[j] < \beta(G)$ because $H[k] \leq H[j]$ as the cost can only increase going from k to j along the path W. We have $H[k] \geq H[i] + w[i, k]$ because i is a parent of k in H. Therefore, $H[k] \geq G[i] + w[i, k]$ because $H[i] \geq G[i]$. Since i is fixed in G and k is not fixed in G, from the definition of $\beta(G)$, we get that $\beta(G) \leq G[i] + w[i, k]$. Hence, $H[k] \geq \beta(G)$. □

By using the advancement Lemma 4.4, we get the algorithm *ShortestPath* shown in Fig. 5. In this algorithm, in every iteration we find all nodes that are forbidden (not fixed) and advance them. All nodes are advanced to $\alpha(G, j)$ that combines $\beta(G)$ with $\min\{G[i]+w[i, j] \mid i \in pre(j)\}$. Note that if a node is fixed, its parent is fixed and therefore any algorithm that advances $G[j]$ only for non-fixed nodes j maintains that once a node becomes fixed it stays fixed.

Theorem 4.5. *Assuming read-write atomicity, there exists a parallel algorithm to solve the shortest path problem with one thread per node of the graph which does not use any synchronization.*

We remark here that the parallel algorithm assumes that the variables in **always** section are maintained using G and therefore is not work efficient.

By removing certain steps, we get Dijkstra's algorithm from the algorithm *ShortestPath*. It is clear that the algorithm stays correct if $\alpha(G, j)$ uses just $\beta(G)$ instead of $\max\{\beta(G), \min\{G[i] + w[i, j] \mid i \in pre(j)\}\}$. Secondly, the algorithm stays correct if we advance G only on the node j such that (i, j) are in Heap edges and the node j minimizes $G[i] + w[i, j]$. Finally, to determine such a node and $\beta(G)$, it is sufficient to maintain a min-heap of all non-fixed nodes j, along with the label that equals $\min_{i \in pre(j), fixed(i,G)} G[i] + w[i, j]$. On making all these changes to *ShortestPath*, we get Dijkstra's algorithm (modified to run with a heap).

It is illustrative to compare the algorithm *ShortestPath* with Dijkstra's algorithm. In Dijkstra's algorithm, the nodes become fixed in the order of the cost of the shortest path to them. In the proposed algorithm, a node may become fixed even when nodes with shorter cost have not been discovered. In Fig. 4, node v_1 becomes fixed earlier than nodes v_3 and v_4. This feature is especially useful when we are interested in finding the shortest path to a single destination and that destination becomes fixed sooner than it would have been in Dijkstra's algorithm.

Dijkstra's algorithm maintains a distance vector $dist$ such that it is always feasible, i.e., for any vertex v there exists a path from source to v with cost less than or equal to $dist[v]$. We maintain the invariant that the cost of the shortest path from source to v is guaranteed to be at least $G[v]$. Therefore, in Dijkstra's algorithm, $dist[v]$ is initialized to ∞ whereas we initialize G to 0. Dijkstra's algorithm and indeed many algorithms for combinatorial optimization, such as simplex, start with a feasible solution and march towards the optimal solution, our algorithm starts with an extremal point in search space (even if it is infeasible) and marches towards feasibility. Also note that in Dijkstra's algorithm, $dist[v]$ can only decrease during execution. In our algorithm, G can only increase with execution.

The *ShortestPath* algorithm and indeed all the algorithms in this paper have a single variable G. All other predicates and functions are defined using this variable. This is because the goal of the paper is to show effectiveness of using lattice-linear predicates and deriving work efficient algorithms is out of the scope of the paper.

4.1 Closure under Meets and Joins

We get the following structural result on the assignment vectors that are feasible.

Shortest path from node s: LLP algorithm

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

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

Shortest path from node s: a variant of Dijkstra's algorithm

same as above except
advance: $G[j] := \min Heap$

Figure 5: Algorithm *ShortestPath* and Dijkstra's Algorithm to find the minimum cost assignment vector less than or equal to T.

Lemma 4.6. *Let G and H be two assignment vectors such that they satisfy B_{rooted}, then $min(G, H)$ also satisfies B_{rooted}.*

Proof. Follows directly from Lemma 2.3, because B_{rooted} is a lattice-linear predicate. □

The set of feasible assignment vectors is not closed under the join operation. In Fig. 4, the vectors (0, 10, 3, 14, 8) and (0, 9, 10, 12, 11) are feasible, but their join (0, 10, 10, 14, 11) is not feasible.

4.2 Constrained Shortest Path Algorithm

We now consider the generalization of the shortest path algorithm with constraints. We assume that all constraints specified are lattice-linear. For example, consider the constraint that the cost of vertex i is at most cost of vertex j. The predicate $B \equiv G[j] \geq G[i]$ is easily seen to be lattice-linear. If any cost vector G violates B, then the component j is forbidden (with $\alpha(G, j)$ equal to $G[i]$). The predicate $(G[i] = G[j])$ is also lattice-linear because it can be written as a conjunction of two lattice-linear predicates $(G[i] \geq G[j])$ and $(G[j] \geq G[i])$. The predicate $B \equiv (G[i] \geq k) \Rightarrow (G[j] \geq m)$ is also lattice-linear. If any cost vector violates B, then we have $(G[i] \geq k) \wedge (G[j] < m)$. In this case, the component j is forbidden with $\alpha(G, j)$ equal to m. Again, from Lemma 2.6, the algorithm *LLP* can be used to solve the constrained shortest path algorithm by combining forbidden and α for constraints with B_{rooted}. An application of the constrained shortest path problem is as follows. Suppose that there are n dispatch trucks that start from the source vertex at time 0. Let $w[i, j]$ denote the time it takes for a truck to go from node i to node j. An assignment vector G is feasible if it is possible to design a tree rooted at the source vertex such that the path from the source vertex to vertex i takes $G[i]$ units of time and G satisfies specified constraints. In Fig. 4, the vector (0, 9, 2, 8, 7) is feasible, but does not satisfy the constraint that $G[1]$ equals $G[2]$. The least vector that satisfies this additional constraint is (0, 9, 9, 12, 11). LLP algorithm can be used to find this vector. When the set of additional constraints is empty, we get back the standard shortest path problem.

An example of a predicate that is not lattice-linear is $B \equiv G[i] + G[j] \geq k$. If the predicate is false for G, then we have $G[i]+G[j] < k$. However, neither i nor j may be forbidden. The component i is not forbidden because if $G[i]$ is fixed but $G[j]$ is increased, the predicate B can become true. Similarly, j is also not forbidden.

5 GRAPHS WITH NEGATIVE WEIGHTS: JOHNSON'S ALGORITHM

We now consider directed graphs which have zero or negative weight edges. We assume that even though there are edges with negative costs, there are no negative cost cycles. We show how a parallel version of Johnson's algorithm can be derived using lattice-linear predicates. Our strategy for finding the shortest path in such a graph X is to convert it into another graph Y on the same set of vertices such that Y has all strictly positive edges and it preserves all shortest paths, i.e., a path is shortest in X iff it is also a shortest path in Y. The graph Y has the same set of vertices and edges as X. The weight of any edge (i, j) is updated as follows:

$$w'[i, j] = w[i, j] + G[j] - G[i] \tag{1}$$

where G is a *price* vector associated with vertices. A price vector G is a non-negative vector such that when we compute new costs of edges, called *reduced* costs, we get that the new cost of every edge is at least 0. The advantage of updating weights using Equation 1 is that it preserves shortest paths.

Lemma 5.1. *Let s and t be any two vertices in the graphs. The weight of any path in the graph Y equals the weight in the graph X plus $(G[t] - G[s])$.*

Since the cost of all paths between s and t are changed by the same amount, it follows that any shortest path in X is a shortest path in Y and vice-versa. Now our task is reduced to finding a price vector such that $w'[i, j]$ is at least 0 for all edges. We use LLP algorithm to find such a vector. Our feasibility predicate B for pricing vector is

$$\forall (i, j) \in E : w[i, j] + G[j] - G[i] \geq 0$$

Furthermore, we require $G[i] \geq 0$ for all i. We first show that B is lattice linear.

Lemma 5.2. *Let X be any graph such that the edge (i, j) has weight $w[i, j]$ and every vertex i has price $G[i]$. Consider the lattice of all non-negative price vectors. Then, the predicate*

$$B \equiv \forall (i, j) \in E : w[i, j] + G[j] - G[i] \geq 0$$

is lattice linear.

Proof. Since lattice linearity is closed under conjunction, it is sufficient to show that $B_e \equiv w[i, j] + G[j] - G[i] \geq 0$ is lattice linear for arbitrary edge $e = (i, j)$. B_e can be rewritten as $G[j] \geq G[i] - w[i, j]$. The right hand side of this inequality is a monotone function on G and hence from the Lemma 2.3 of lattice-linearity, we get that B_e is lattice linear. □

By applying LLP algorithm, we get the parallel algorithm in Fig. 6 to find the price vector.

The algorithm obtained is same as the parallel version of Johnson's algorithm.

Graph Transformation: Johnson's algorithm

input: $pre(j)$: list of $1..n$; $w[i,j]$: int for all $i \in pre(j)$
init: $G[j] := 0$;
forbidden: $G[j] < \max\{G[i] - w[i,j] \mid i \in pre(j)\}$
advance: $G[j] := \max\{G[i] - w[i,j] \mid i \in pre(j)\}$

Figure 6: Algorithm $Price_b$ to find the minimum price vector.

6 CONSTRAINED MARKET CLEARING PRICE

In this section, we apply our technique to the problem of finding a market clearing price with constraints. This problem is equivalent to the heavily studied problem of weighted bipartite matching. Let I be a set of n indivisible items, and U, a set of n bidders. Every item $i \in I$ is given a *valuation* $v_{b,i}$ by each bidder $b \in U$. The valuation of any item i is a number between 0 and $T[i]$. Each item i is given a price $G[i]$ which is also a number between 0 and $T[i]$. We are assuming integral costs for simplicity — the algorithm is easily extensible to real costs.

Given a price vector G, we define the bipartite graph $(I, U, E(G))$ as

$$(j,b) \in E(G) \equiv \forall i : (v_{b,j} - G[j]) \geq (v_{b,i} - G[i]).$$

Informally, an edge exists between item i and bidder b if the payoff for the bidder (the bid minus the price) is maximized with that item. Given any set $U' \subseteq U$, let $N(U', G)$ denote all the items that are adjacent to the vertices in U' in the graph $(I, U, E(G))$. A price vector G is a *market clearing price*, denoted by $B_{clearingPrice}(G)$ if the bipartite graph $(I, U, E(G))$ has a perfect matching. We now generalize the problem of finding a market clearing price to that of finding a constrained market clearing price. For example, the constraint $G[i] \geq G[j]$ is lattice-linear. Given any set of valuations, and a boolean predicate B that is a conjunction of lattice-linear constraints, a price vector G is a *constrained market clearing price*, denoted by $constrainedClearing(G)$ iff $clearing(G) \wedge B(G)$. From Lemma 2.6, it is sufficient to give an algorithm for $clearing(G)$.

We now claim that

Lemma 6.1. *The predicate $B_{clearingPrice}(G)$ is a lattice-linear predicate on the lattice of price vectors.*

Proof. Let J be a minimal over-demanded set in G. If J is a singleton, then it is clear that unless price on j is advanced it will stay over-demanded. Now suppose that J is not singleton. We show that each of the elements in J is forbidden. Let $j \in J$. We set $I_0 = \{j\}$. Consider any H that is greater than G such that $G[j] = H[j]$. Let $bidders(G, j)$ be the set of bidders for item j at the price vector G. If $bidders(G, j)$ is a singleton, then J cannot be the minimal overDemanded set because we can remove j from J and the resulting set is also over-demanded as exactly one bidder is eliminated. Since $bidders(G, j)$ have items assigned in H, and at most one of them could be assigned item j, the remaining items must be assigned to bidders from J. Since these bidders preferred item j in G and the price of j has not changed, we get that these assigned items must be most preferred in G as well. The price of these items could not have increased in H; otherwise, these bidders will not prefer these items to j. Hence, all items that $bidders(G, j)$ are assigned in H cannot have their price changed. Let us call this set of items, I_1. We now repeat this procedure armed with the knowledge that price of I_1 is same in G and H. We consider $bidders(G, I_1)$. If this set has the same size as I_1 then by removing I_1 from J we get a smaller over-demanded set than J. Otherwise, we get $I_2 \subseteq J$ such that I_2 has same price in G and H and I_2 is bigger than I_1. By repeating this procedure, we must either find a smaller over-demanded set than J or find that none of the items in J has any price change in H. In the former case we get a contradiction to minimality of J and in the latter case we get a contradiction to H being a clearing price because the size of $bidders(H, J)$ is greater than J. □

P_j: Code for thread j
shared var G: array$[1..n]$ of $0..maxint$;
Market Clearing Prices: Demange Gale Sotomayor algorithm
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: $(\exists minimal\ J : OverDemanded(J) \wedge (j \in J)$
advance: $G[j] := G[j] + 1$;

Figure 7: Algorithm ConstrainedMarketClearingPrice to find the minimum cost assignment vector

It follows that the set of constrained market clearing price vectors is closed under meets. By applying the lattice-linear predicate detection, we get an algorithm to compute the least constrained market clearing price shown in Fig. 7. In conjunction with Lemma 2.6, we get a generalization of Demange, Gale and Sotomayor's exact auction mechanism [7] to incorporate lattice-linear constraints on the market clearing price. In Fig. 7, we have used $\alpha(G, j)$ as simply one unit of price. For any item j that is part of a minimal over-demanded set of items, we can increase its price by the minimum amount to ensure that some bidder b can switch to her second most preferred item.

7 DUAL OF LATTICE-LINEARITY

Just as a lattice-linear predicate allows us to start with the bottom element of the lattice and advance in the forward direction, its dual allows us to start with the top element and advance in the backward direction.

Given any distributive lattice L of n-dimensional vectors, and any predicate B, we say reverse-forbidden$(G, i, B) \equiv \forall H \in L : H \leq G : (G[i] = H[i]) \Rightarrow \neg B(H)$. We define a predicate B to be *dual-lattice-linear* if for any $G \in L$, B is false in G implies that G contains a *reverse-forbidden* component.

It can be shown that $B_{marriage}$ is not only lattice-linear but also dual-lattice-linear.

This property allows us to find the man-pessimal stable matching. We start with G such that $G[i]$ equals the last choice proposal for P_i. If G is a stable matching, we are done. Otherwise, we can find i such that unless $G[i]$ goes backward, there cannot be any stable matching.

These i can be found in parallel. By repeating this procedure, we get the man-pessimal stable matching.

Since stable matching is a dual-lattice-linear predicate, from the dual of Lemma 2.3(b) it follows that the *feasible set*, the set of assignments satisfying $B_{marriage}$, is also closed under joins. Therefore, the feasible set forms a sublattice of the lattice of all assignments. A similar result holds for $B_{clearingPrice}$ (but not for B_{path}).

Since a sublattice of a distributive lattice is also distributive, the set of assignments that satisfy (constrained) stable marriage forms a finite distributive lattice. From Birkhoff's theorem [6] we know that a finite distributive lattice can be equivalently represented using the poset of its join-irreducible elements.

The set of all elements of L satisfying B can be generated as the ideals of the poset $(\{J(B,e)|e \in E\}, \subseteq)$ where $J(B,e)$ is the least ideal of $(E, \rightarrow)$ that satisfies B and contains e. It can be verified that $J(B,e)$ is a join-irreducible element and that every join-irreducible element is of this form.

To determine $J(B,e)$ it is sufficient to use the algorithm for detecting a lattice-linear predicate by using the following predicate for every e: $B_e(G) \equiv B(G) \wedge (e \in G)$. Since B_e is a conjunction of two lattice-linear predicates, it is also lattice-linear. Therefore, by using the LLP algorithm, we also get an algorithm to compute the poset that generates all ideals that satisfy B. For stable marriage, this would be equivalent to rotation poset [14]. However, we can now also generate posets for constrained stable marriages or constrained price vectors.

8 RELATED WORK

In this section, we compare LLP Algorithm to other related techniques. We note here that a preliminary version of this work appeared in [13].

8.1 Linear Programming

Linear programming can also be viewed as a search for an optimal feasible solution. However, there are many important differences. First, the underlying space in linear programming is the set of real valued vectors whereas the underlying space in the lattice-linear predicate detection method is a distributive lattice. In the domain of distributive lattices, we do not have addition or the scalar multiplication as in vector spaces. All of lattice-linear predicate algorithms use the following two operations: meet of the underlying lattice and the "advance" operation. The advance operation maps an element of the lattice to a bigger element in the lattice. In linear programming, the feasible space is characterized by a polyhedron (or the set of vectors x such that $Ax \leq 0$ for some matrix A). There is no lattice structure required on the feasible space. It is not guaranteed that if two vectors are feasible, then their component-wise minimum vector is also feasible. Lattice-linear predicate detection requires the feasible space to be closed under meets. Finally, even though many problems studied in this paper can also be solved via linear programming, the algorithms derived in that manner are not as efficient or parallel as the LLP algorithm.

8.2 Relationship with Knaster-Tarski's Theorem

The algorithm in Fig. 1 can also be viewed as repeated iteration of a monotone function on the bottom element of a lattice similar to a constructive version of Knaster-Tarski's theorem [20]. Our work differs from such earlier work in many respects. First, B may not have the form $\forall i : G_i \geq f_i(G)$; instead we only require B to be closed under meets. Second, Knaster-Tarski's fixed point theorem (and many variants) requires the function to be from the lattice L to itself. In that case, the solution to the equation $x \geq f(x)$ always exists for a complete lattice because $\top \geq f(\top)$. We do not assume that the range of the function is the lattice itself. Therefore, there is no guarantee of the existence of the fixed point. Indeed, for the job scheduling example, if the prerequisites have a cycle and weights are positive, then there is no solution and the algorithm *LLP* returns null. Third, the goal of this paper is to develop techniques to reach the fixed point with an efficient parallel algorithm and to show that many standard and non-standard parallel algorithms for combinatorial optimization can be derived in this framework.

8.3 Relationship with Predicate Detection in Distributed Systems

Although the notion of forbidden and detection of lattice-linear predicates is from [4], there are many important differences from their work. The focus of their work in [4] is for detecting a global condition is a distributed system. Our work is focused on developing parallel algorithms for the optimization problems. The predicates used in [4] are simple conjunction of local predicates in a distributed system (or monotonic channel predicates). The predicates discussed in this paper are more general and motivated by optimization problems.

9 CONCLUSIONS AND FUTURE WORK

We have shown that many discrete optimization problems can be cast as searching for an element satisfying a lattice-linear predicate in a distributive lattice. The algorithms that can be derived using this framework include Gale-Shapley algorithm, Dijkstra's algorithm, Gale-Demange-Sotomayor algorithm, Bellman-Ford algorithm, and Johnson's algorithm. All of these algorithms have a single vector as the variable of the program and assuming read-write atomicity, the proposed algorithm uses different threads to update different components of the vector without use of any lock.

All of our examples include problems in class P. Are there techniques to find approximation algorithms for problems that are not in P based on lattice-linearity? We have exploited lattice linearity of the predicate in LLP algorithm. What if the problem requires searching an element that satisfies a condition which is not lattice-linear?

ACKNOWLEDGEMENTS.

I would like to thank David Alves, Rohan Garg, Changyong Hu, Calvin Ly, and Xiong Zheng for discussions on this topic and anonymous reviewers for useful comments. This work was supported in parts by the National Science Foundation Grants CNS-1812349, CNS-1563544, and the Cullen Trust Endowed Professorship.

REFERENCES

[1] Richard Bellman. 1958. On a routing problem. *Quarterly of applied mathematics* 16, 1 (1958), 87–90.

[2] Dimitris Bertsimas and John N Tsitsiklis. 1997. *Introduction to linear optimization*. Vol. 6. Athena Scientific Belmont, MA.

[3] G. Birkhoff. 1967. *Lattice Theory*. Providence, R.I. third edition.

[4] Craig M Chase and Vijay K Garg. 1998. Detection of global predicates: Techniques and their limitations. *Distributed Computing* 11, 4 (1998), 191–201.

[5] Agnes Cseh and David F. Manlove. 2016. Stable Marriage and Roommates problems with restricted edges: Complexity and approximability. *Discrete Optimization* 20 (2016), 62 – 89.

[6] B. A. Davey and H. A. Priestley. 1990. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, UK.

[7] Gabrielle Demange, David Gale, and Marilda Sotomayor. 1986. Multi-item auctions. *Journal of Political Economy* 94, 4 (1986), 863–872.

[8] Vânia M.F. Dias, Guilherme D. da Fonseca, Celina M.H. de Figueiredo, and Jayme L. Szwarcfiter. 2003. The stable marriage problem with restricted pairs. *Theoretical Computer Science* 306, 1 (2003), 391 – 405.

[9] E. W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Math.* 1, 1 (01 Dec 1959), 269–271. https://doi.org/10.1007/BF01386390

[10] L. A. Ford. 1956. *Network Flow Theory*. Technical Report.

[11] David Gale and Lloyd S Shapley. 1962. College admissions and the stability of marriage. *The American Mathematical Monthly* 69, 1 (1962), 9–15.

[12] Vijay K Garg. 2015. *Lattice Theory with Computer Science Applications*. Wiley, New York, NY.

[13] Vijay K. Garg. 2018. Applying Predicate Detection to the Constrained Optimization Problems. *CoRR* abs/1812.10431 (2018). arXiv:1812.10431 http://arxiv.org/abs/1812.10431

[14] Dan Gusfield and Robert W Irving. 1989. *The stable marriage problem: structure and algorithms*. MIT press.

[15] Donald B Johnson. 1977. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM)* 24, 1 (1977), 1–13.

[16] Donald Ervin Knuth. 1997. *Stable marriage and its relation to other combinatorial problems: An introduction to the mathematical analysis of algorithms*. Vol. 10. American Mathematical Soc.

[17] Neeraj Mittal and Vijay K Garg. 2001. Computation slicing: Techniques and theory. In *International Symposium on Distributed Computing*. Springer, 78–92.

[18] James Munkres. 1957. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics* 5, 1 (1957), 32–38.

[19] Lloyd S Shapley and Martin Shubik. 1971. The assignment game I: The core. *International Journal of game theory* 1, 1 (1971), 111–130.

[20] Alfred Tarski. 1955. A Lattice-Theoretic Fixed Point Theorem and its Applications. *Pacific J Math* 5 (1955), 285–309.